Detecting the Knowledge Frontier: An Error Predicting Knowledge Based System

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. Knowledge Based Systems (KBS) have long wrestled with the problem of incomplete knowledge that occasionally causes them to make ridiculous conclusions. Knowledge engineers have searched for methodologies that allow for less brittle systems. Additionally, KBS systems for general knowledge have been developed to try and build background information that a system can fall back on when they cannot find a conclusion in their specific domain. However, it is next to impossible to include all the required knowledge to completely eradicate the inherent brittleness of these systems. This paper presents a method for predicting when a case being presented to the KBS is outside its current knowledge. When the system notices such a case it provides a warning allowing the user to investigate the case further. This preliminary study of the system has been tested using a simulated expert with randomly generated data sets. It shows that this system has great potential for predicting almost every error and rarely issuing warnings for correct conclusions. Such a system could significantly reduce the knowledge acquisition task for an expert. These results clearly show the potential of such a system and that further investigation with recognised data sets and real user tests should be performed.

Introduction

Regardless of how intelligent we think we are, there will occasionally be that slip up - that embarrassing moment when we do or say something that reveals an extreme lack of knowledge about one particular detail that *everyone* else knows. This, fairly *rare* human ailment, has long been recognised as being a *frequently* occurring flaw in knowledge based systems (KBS). While, such an error may make the person that made the mistake uncomfortable, it can usually be covered up or laughed at and then simply put aside. However, when a knowledge base makes this error it highlights a major error in a system's knowledge base and causes the system's user to lose faith in the computer's ability to give an accurate and meaningful conclusion. The famous apocryphal example is the expert system that diagnosed a man as being pregnant [1].

[1] Collaborative research project between both institutions

This *brittleness*, often associated with KBSs, is founded in the system's inability to realise when its knowledge base is inadequate to provide an accurate and meaningful conclusion. The cause of such inadequacies is generally recognised as being due to the concentration of specialised knowledge in the target domain for the particular system [2]. Within this particular domain they may perform exceptionally well, however, the moment some form of knowledge is needed from just outside of this domain their *competence* drops off quickly to complete *incompetence*. As the introductory paragraph alluded, people are also subject to this difficulty. However, they have the ability to fall back on layer upon layer of general knowledge providing a much less precipitous slope [2].

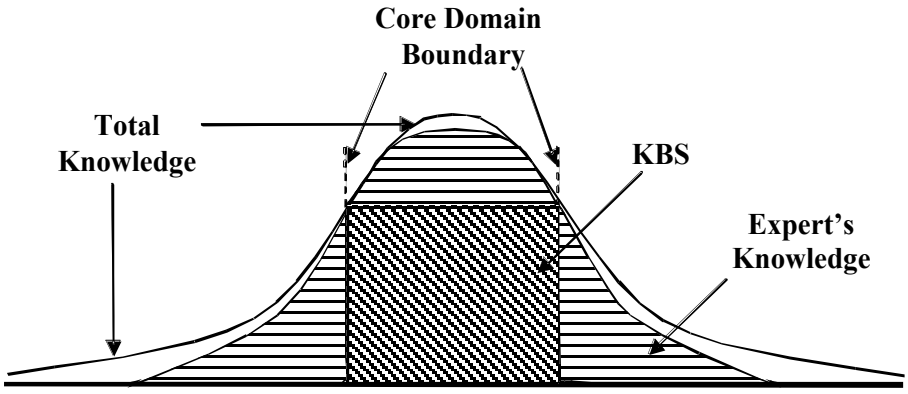

Figure 1: Conceptual diagram of knowledge distribution within a particular domain.

This can be viewed in the conceptual diagram, Figure 1, where the full knowledge needed for a particular domain is represented by the normal distribution. The choice of the normal distribution is to highlight that, while the vast majority of the knowledge required is in the core of the domain, the potential knowledge needed is actually infinite. Therefore, at no time can either a person or computer know everything required. It also shows that the further you move away from the core domain knowledge the less relevant and less likely that knowledge will actually be required. It can be seen in this diagram that the KBS generally has a very poor coverage of the general knowledge beyond the core domain, as well as, an incomplete knowledge base for the domain itself. This is primarily due to the inherit difficulties of current KA methodologies being unable to extract all of the experts knowledge.

While researchers and designers have sought methods of improving the knowledge coverage of KBSs, it should be realised that a *perfect* KBS in a domain may never be achieved. Therefore, researchers have been simultaneously investigating methods for checking if a knowledge-base is *complete* (as complete as possible for a KBS), commonly referred to as validation and verification [3]. However, they tend not to be particularly useful or cover a large range of data patterns. The larger the data pattern the more frequently the expert will be asked to speculate about hypothetical situations that they may have no experience [1, 4, 5] and that may never occur in reality.

Rather than attempting to investigate the completeness of the knowledge base as a whole; this paper presents a rarely attempted approach of identifying inadequacies in the knowledge base for only the case currently being presented. Therefore, when a case is presented, which the system detects as being beyond its current ability to accurately assess, it provides a warning. This is a highly useful tool in systems where the knowledge base is being constructed incrementally by the expert. Multiple Classification Ripple Down Rules (MCRDR)[6-8] is one such system.

MCRDR has previously been shown to be a highly effective incremental learning Knowledge Based System (KBS)[6-8]. It allows a domain expert to add rules online by providing justifications identifying the differences between cases within the context provided. The one major inherent problem with MCRDR is that the human expert must be in a position to review each and every case to ensure that it is classified correctly. This can be excessively time consuming, especially in later stages of development when there are only a tiny percentage of misclassifications requiring correction. Therefore, the aim of this work was to reduce the need for case review by the expert without reducing the system's accuracy.

Thus, this paper describes an augmented hybrid system, referred to as Rated MCRDR (RM) and how it can be applied to Knowledge Acquisition Warnings (KAW). In this system, the conclusions and their justifications from MCRDR for each case are fed into a purpose built resource-allocating radial basis function (RBF) neural network and trained using the single-step-Δ-update-rule [9]. RM is capable of classifying cases into single or multiple classifications. Additionally, RM has been shown to be able to provide a prediction or evaluation for continuous value ranges [9]. However, this paper will specifically investigate RM's ability to predict the likelihood that a classification or value prediction by the system is correct.

Previously, a system was developed for predicting errors in the single classification predecessor to MCRDR, known as Ripple Down Rules (RDR)[1]. The approach taken was based simply on the idea of keeping a record of every case that passed through the system and then issuing a warning when a case presented was sufficiently different from all the previously seen cases within the particular classification path followed [1]. The system was tested using a simulated expert on three datasets. The best results were found on the Garvan dataset, where the system was able to predict approximately 92% of errors. However, it achieved this by provided a warning on 17% of cases, where there were actually only 2.6% of errors [1]. While this reduces the load on the expert significantly, there is still much opportunity for further improvement. In conclusion the system was found to be a "…major advance" [1], and potentially a "…useful aid in incremental knowledge acquisition"[1]. However, its performance was not sufficient by itself for most applications.

As will be discussed further in this paper, preliminary testing of RM has been able to significantly improve on these results, with the added advantage of not being required to store, retrieve and compare every case; significantly reducing the system's memory and computational load on the computer. The assumption behind RMs Knowledge Acquisition Warning (RM-KAW) technique is that if a case presented to the system follows a significantly different pattern-of-paths (POPs) through the MCRDR structure, then the expert should be warned of a potential error. Therefore, the system is using the structure of the MCRDR n-ary tree itself to determine when a case is outside its experience and, thus, requiring the expert to verify the conclusion.

Effectively, RM-KAW is keeping track of the knowledge the system has for a particular domain and, thus, tries to identify when a case being classified or rated is outside that area. For instance, it identifies when a case is using knowledge from outside the KBSs area of knowledge, figure 1. This allows the system to slowly expand its knowledge frontier into very specific specialised areas of a domain, along with gathering general background information. This provides a method for significantly reducing the brittleness of a knowledge based system.

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 [8]. The system then lists the collection of classifications and the paths they followed.

Knowledge Acquisition 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 [10]. 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 [10].

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. 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. In the case of RM-KAW this value can be used to identify a confidence-*like*-factor. This could then be used to identify when the system should issue a warning that the case is unfamiliar and the conclusions should be checked by the expert.

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, importance or confidence in its own classification for the case presented. Therefore, a mapping must be found from the set $C\_\bar{v}, \forall C \in \wp(C^*)$. 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 [11].

The neural network was integrated into MCRDR by linking each possible rule or class to an input neuron. Then, for each rule or classification found by the MCRDR system, an associated neuron will fire. In this implementation of RM a purpose built resource-allocating radial basis function (RBF) network was used. The output nodes use the standard sigmoid thresholding function, equation 1, using a *modified* generalised delta rule. A subset of the input nodes is selected by the hidden layer by using the Gaussian function, equation 2, where the distance measure r is taken to be Euclidean, equation 3.

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

$$\phi(r) = e^{(-r^2)} \tag{2}$$

$$\|x - y\| = \sum_i (x_i - y_i)^2 \tag{3}$$

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 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 2 and shown diagrammatically in Figure 3, 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, $\overline{v}$, 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.

For example, in Figure 3, 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 value, which in the case of RM-KAW, can be used as a measure of confidence in the systems conclusion.

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
$\overline{i}$ ← Generate input vector from *l*.
NN ← $\overline{i}$
v ← *NN output value.*

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

Figure 2: Inference Algorithm for RM.

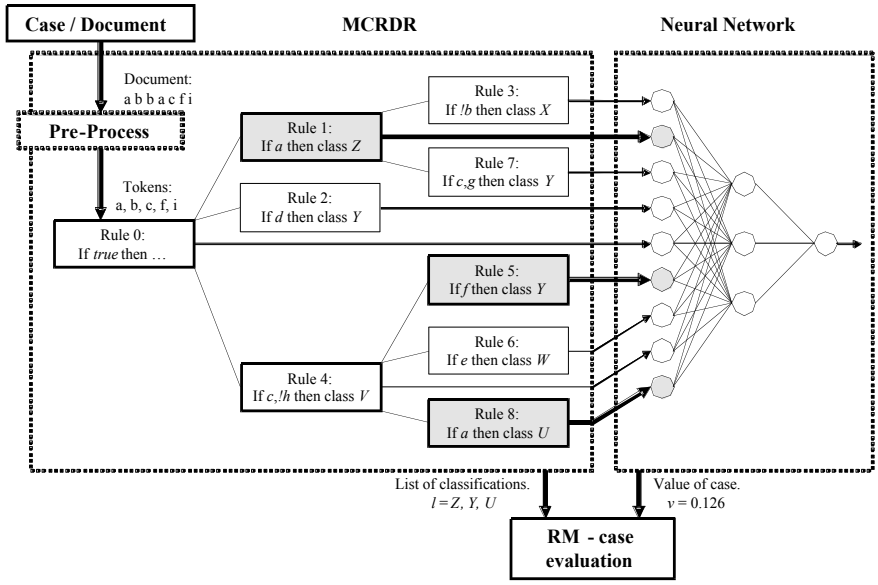

Figure 3: RM illustrated diagrammatically.

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 RM-KAW feedback with a high value may be given if the conclusion produced by the system is not changed by the expert. 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.

Neural Network Structure and Learning

The neural network selected in developing RM for KAW was the standard radial basis function where the output nodes use the standard sigmoid thresholding function, equation 1, and the hidden layer uses the Gaussian function, equation 2, with a Euclidean distance measure, equation 3. Further experimentation still needs to be carried out to determine if other functions would provide better results. For instance, due to the discrete inputs currently used, it is expected that a broader fitness function at the hidden layer may provide better generalisation.

Some variations to the standard RBF network were required, due to the creation of additional rules and classifications, described earlier. Primarily, the capability to increase its number of input nodes to ensure one input node for each possible rule or

class. Likewise, due to the growing nature of the input space, it can be expected that the number of significant patterns would also increase. Thus, a system was also developed for automatically allocating addition resources, in the form of hidden nodes.

A new input node is added to the system only when the user has corrected a conclusion. This, therefore, also means the user has identified a new significant pattern (whatever the new input vector is after the correction), which should automatically be captured in the hidden layer. Thus, a new hidden node is added with appropriate input weights assigned that will produce a Euclidean distance of zero only for that input sequences. However, the weight assigned to determine the contribution of this pattern to the overall confidence of future cases also must be found.

The simplest approach is to assign a random start up weight in the same fashion used when first initialising the network. However, we already have an accurate measure of the confidence for the new pattern, because the expert just created the new conclusion. Thus, we can be reasonably sure it is correct and assign the maximum confidence level for the new pattern.

Furthermore, in this implementation, additional hidden node resources are also added even when input nodes aren't added. This is done when a significant error is found, generally when a warning was made and the user decided that the conclusion was correct. If, when such a situation occurs, the system will check to see whether there are any nodes providing a reasonably close representation of the input pattern and if not a new hidden node is added with a Euclidean distance of zero. Once again a valid estimate can be made from the user's behaviour, providing a recommended value for the new patterns contribution to the system's overall confidence.

Single-Step-_-Update-Rule

In order to calculate the weight needed for a new connection from a just created hidden node and each output node, w_{no}, the system must first calculate the error in the weighted-sum, $\_{ws}$. This is then divided by the input at the newly created hidden node, h_n, which is always one in this implementation due to it being assigned a Euclidean distance of zero for the given input vector, where there are $n>0$ hidden nodes and $o>0$ output nodes.

$$w_{no} = \frac{\delta_{ws}}{x_n} \tag{4}$$

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

$$\delta_{ws} = R_{ws} - net \tag{5}$$

The value of *net* for each output node, o, was previously calculated by the network during the feed forward operation, and is shown in Equation 6, where there are $n>1$ hidden nodes and the n^{th} hidden node is our new input node.

$$net_o = \left(\sum_{h=0}^{m} x_h w_{ho} \right) \quad (6)$$

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 7, where *f(net)* is the original thresholded value that was outputted from that neuron.

$$R_{ws_o} = \frac{\log\left[\frac{f(net)_o + \delta_o}{1 - \left(f(net)_o + \delta_o\right)}\right]}{k} \quad (7)$$

Thus, the full *single-shot-_-update-rule*, used for each new hidden node's connection with each output node is given in Equation 8. 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) + _) > 1* as this will cause an error.

$$w_{no} = \frac{\left(\log\left[\frac{f(net)_o + \delta_o}{1 - \left(f(net)_o + \delta_o\right)}\right] \Big/ k \right) - \left[\sum_{h=0}^{m} x_h w_{ho} \right]}{x_n} \quad (8)$$

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 [6, 12, 13]. 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 [6, 8].

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.

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. When classifying each case the case also gauged the level of confidence the system has in its conclusion. The test carried out at this stage has not used this confidence directly. Instead, it simply gathered statistics on how accurate its predictions actually were. Thus, the simulated user actually still checked every case, ignoring any warnings, and created new rules when ever it found a case incorrectly

classified. Rewards were given to the RM system depending on what it predicted and the action taken by the expert. 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.

Results and Discussion

In the preliminary test discussed in this paper, one of the following four details where recorded for each case as it was processed:

- If the conclusion was correct and RM gave a warning then a *False Positive* was recorded.
- If the conclusion was correct and RM did not give a warning then a *True Negative* was recorded.
- If the conclusion was incorrect and RM gave a warning then a *True Positive* was recorded.
- If the conclusion was incorrect and RM did not give a warning then a *False Negative* was recorded.

In Figure 4, the black line shows how many times the user corrected rules overtime. While the white line shows when a rule that the user corrected was warned about by RM; *true positives*. It can clearly be observed that RM was able to identify nearly all, approximately 97.73%, incorrect classifications. Therefore, these results indicate that an expert using this system to identify when a case is misclassified may be confident that the majority of errors would generate a warning.

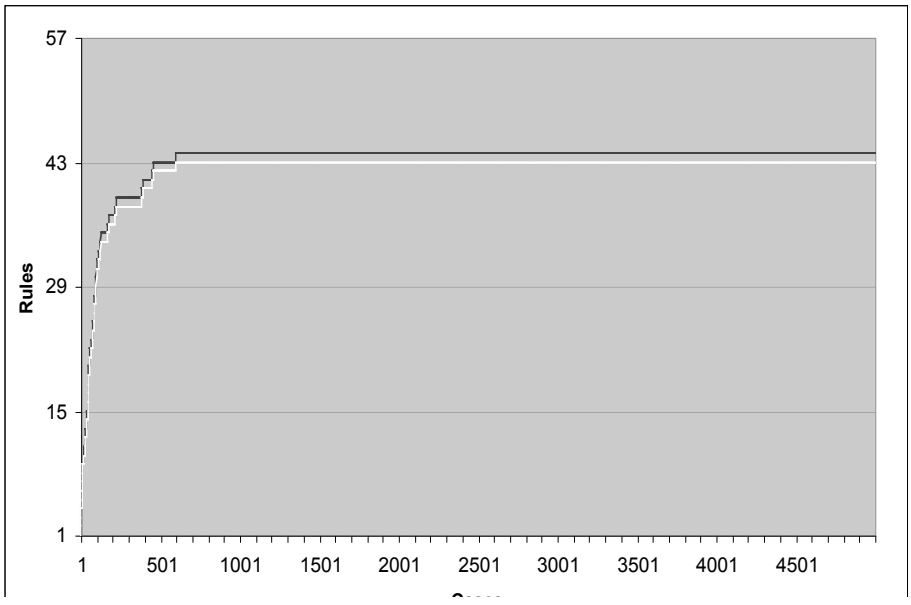

Figure 4: Comparison of the total rules created and the total rules first warned about and then created.

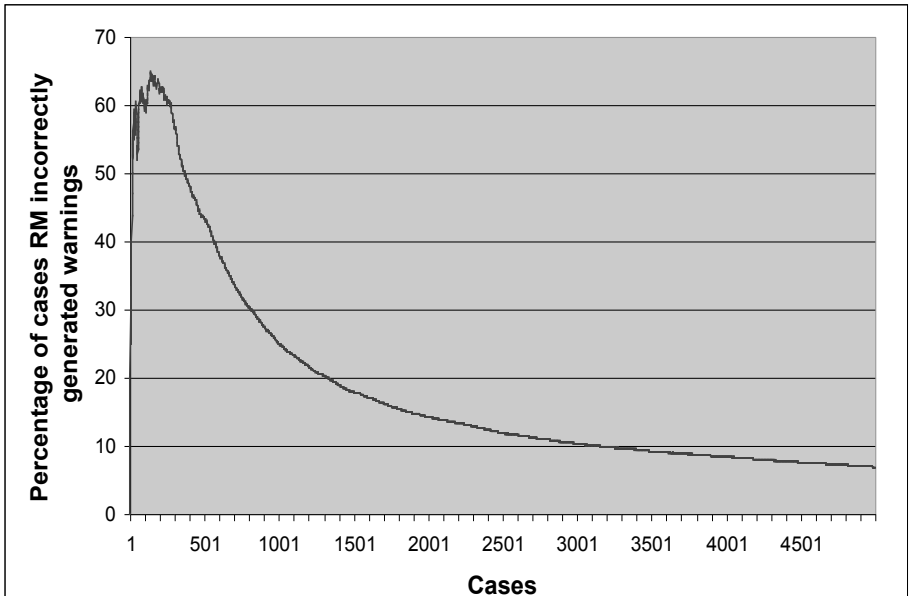

Figure 5: Percentage of cases that recieved a warning over time.

However, if the system simply provides a warning for every case then this result is meaningless. Therefore, RM would also need to minimise the amount of warnings generated when a case is correct, without reducing the amount of correct warnings. Figure 5, illustrates how RM is highly successful at reducing the amount of incorrect warnings, down to only 6.8% after 5000 cases, being generated as the knowledge base grows and RM learns. Furthermore, it has accomplished this without loosing its ability to identify the incorrect cases.

While these results are only preliminary, and direct comparisons with Compton et al's [1] work of identifying warnings in RDR cannot be made at this stage, they do show that RM has lots of potential when used for generating warnings. With 97.73 % accuracy and only generating warnings in less than 8 % of cases shows that it could possibly be used for knowledge acquisition and reducing the load on the user having to review every case. It would be particularly useful after the majority of the knowledge base has been formed and we are primarily interested in only identifying those rare cases that require knowledge from outside the core domain already known to the system.

Conclusion and Future Work

The system described in this paper was developed to provide a means for identifying knowledge that sits outside the current knowledge held by the knowledge based system. The system was designed to learn which patterns of paths followed were likely to be correct and which were unusual. When an unusual classification pattern was found then the system provides a warning bring the case to the user's attention,

allowing them to verify the correctness of the conclusions found. The system used Multiple Classification Ripple Down Rules (MCRDR) as its incremental Knowledge Based System. The patterns of rules followed in generating a set of classifications, then fed into a purpose built resource allocating radial basis function neural network using the single-step-Δ-update-rule for faster learning.

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. Additionally, testing using data sets used by Compton et al's work and real world user tests need to also be performed.

Acknowledgements

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

References

1. P. Compton, P. Preston, G. Edwards and B. H. Kang. Knowledge based systems that have some idea of their limits. in *Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*. 1996
2. D. B. Lenat and E. A. Feigenbaum, On the threshold of knowledge, *Artificial Intelligence*, (1991). **47**(1-3): p. 185-250.
3. A. D. Preece, Validation of knowledge-based systems: Current trends and issues., *The Knowledge Engineering Review*, (1995). **10**(1): p. 69 - 71.
4. R. Colomb and J. Sienkiewicz. Analysis of Redundancy in Expert Systems Case Data. in *AI'95 Eighth Australian Joint Conference on Artificial Intelligence*. 1996
5. P. Compton and R. Jansen, A philosophical basis for knowledge acquisition., *Knowledge Acquisition*, (1990). **2**: p. 241-257.
6. 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
7. B. H. Kang, Validating Knowledge Acquisition: Multiple Classification Ripple Down Rules. 1996, University of New South Wales: Sydney.
8. 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
9. R. Dazeley and B. H. Kang. Rated MCRDR: Finding non-Linear Relationships Between Classifications in MCRDR. in *3rd International Conference on Hybrid Intelligent Systems*. 2003. Melbourne, Australia: IOS Press

10.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
11.R. Beale and T. Jackson, *Neural Computing: An Introduction.* 1992, Bristol, Great Britain: IOP Publishing Ltd
12.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
13.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.