EVALUATION OF RULE EXTRACTION ALGORITHMS

Tiruveedula GopiKrishna

Department of Computer Science, Sirt University, Hoon, Libya

ABSTRACT

For the data mining domain, the lack of explanation facilities seems to be a serious drawback for techniques based on Artificial Neural Networks, or, for that matter, any technique producing opaque models In particular, the ability to generate even limited explanations is absolutely crucial for user acceptance of such systems. Since the purpose of most data mining systems is to support decision making, the need for explanation facilities in these systems is apparent. The task for the data miner is thus to identify the complex but general relationships that are likely to carry over to production data and the explanation facility makes this easier. Also focused the quality of the extracted rules; i.e. how well the required explanation is performed. In this research some important rule extraction algorithms are discussed and identified the algorithmic complexity; i.e. how efficient the underlying rule extraction algorithm is.

KEYWORDS

Extraction Algorithms; taxonomy of rule extraction; evolution of rule extraction algorithms; scalability; comprehensibility.

1. INTRODUCTION

For the data mining domain, the lack of explanation facilities seems to be a serious drawback for techniques based on Artificial Neural Networks (ANNs), or, for that matter, any technique producing opaque models. Within the field of symbolic AI, the term explanation refers to an explicit structure which is used internally for reasoning and learning and externally for the explanation of the results to the user. Normally, the explanation facility in symbolic AI includes intermediate steps of the reasoning process, like trace of rules firing and proof structures. Generally speaking, the explanation facilities are capable of answering the "how" and "why" questions. The answer to a how-question is an explanation of how the result was found. A why-question is supplied by the user during execution of the system and the answer specifies why the system performed a certain operation; e.g. queried the user.

Experience from the field of expert systems has shown that an explanation capability is a vital function provided by symbolic AI systems. In particular, the ability to generate even limited explanations is absolutely crucial for user acceptance of such systems [1]. Since the purpose of most data mining systems is to support decision making, the need for explanation facilities in these systems is apparent. Nevertheless many systems (especially those using ANN techniques, but also ensemble methods like boosting) must be regarded as black boxes; i.e. they are opaque to the user.

In [2] the authors Andrews, Diederich and Tickle highlight this deficiency of ANNs, and argue for rule extraction; i.e. to create more transparent representations from trained ANNs:

DOI : 10.5121/ijdkp.2014.4302

It is becoming increasingly apparent that the absence of an explanation capability in ANN systems limits the realizations of the full potential of such systems, and it is this precise deficiency that the rule extraction process seeks to reduce.

Andrews, Diederich and Tickle also list five more reasons for the importance of being able to interpret trained ANNs:

- For ANNs to be used in safety-critical problem domains, the explanation facility must be considered mandatory.
- If ANNs are integrated into larger software systems without an explanation facility, the entire systems would be very hard to verify.
- An explanation facility may provide an experienced user with the capability to anticipate or predict a set of circumstances under which the ANN is likely to generalize poorly.
- ANNs may discover previously unknown dependencies, but without an explanation facility these dependencies are incomprehensibly encoded in the model.
- Extracted rules from ANNs could be directly added to the knowledge base of a symbolic AI system.

It should be noted that an explanation facility also offers a way to determine data quality, since it makes it possible to examine and interpret relationships found. If the discovered relationships are deemed doubtful when inspected by a human, they are less probable to actually add value. "Nonsense" relationships found would, if used on a production set, most likely produce poor results. The task for the data miner is thus to identify the complex but general relationships that are likely to carry over to production data and the explanation facility makes this easier.

There are basically two methods that can be used to gain understanding of the relationship found by a trained ANN; sensitivity analysis and rule extraction. Sensitivity analysis does not produce a new model, but is used to gain some basic understanding of the relationship between input variables and the output. Rule extraction is an activity where the trained ANN is transformed into another, more comprehensible model, representing the same relationship.

2. SENSITIVITY ANALYSIS

Sensitivity analysis does not provide explicit rules, but is used to find the relative importance of the inputs to the output of the neural net. There are some small variations in how the analysis is performed, but the overall procedure is to record changes in the output following changes in specific input attributes. Normally, the average value for each input is chosen as the starting point and the changes should vary from small changes all the way up to the extreme values. If the difference in output is small even for large changes in a specific attribute, this attribute is probably not very important; i.e. the network is insensitive to that attribute. Other attributes might have a large effect on the output and the network is then said to be sensitive to these attributes.

Obviously, there could be combinations of features that are very important for the network, which would require the sensitivity analysis to be performed on two or more attributes at the same time, in order to find these particular patterns.

It is safe to say that sensitivity analysis is a good tool to get some basic understanding of the underlying problem, but also that it normally is unable to produce explanations. The purpose of performing a sensitivity analysis is thus usually not to actually explain the relationship found. Instead sensitivity analysis is normally used either as a tool to find and remove unimportant input attributes or as a starting point for some more powerful explanation technique.

3. RULE EXTRACTION FROM TRAINED NEURAL NETWORKS

The knowledge acquired by an ANN during training is encoded as the architecture and the weights. The task of extracting explanations from the network is therefore to interpret, in a comprehensible form, the knowledge represented by the architecture and the weights.

Craven and Shavlik [3] coined the term representation language for the language used to describe the network's learned model. They also used the expression extraction strategy for the process of transforming the trained network into the new representation language.

4. TAXONOMY OF RULE EXTRACTION APPROACHES

In [2] the authors proposed a classification schema for rule extraction approaches. The presentation below intentionally follows the one given in the paper closely. In the paper, the method of classification is based on five dimensions:

- The expressive power of the extracted rules; i.e. the chosen representation language.
- The translucency of the view taken within the rule extraction technique of the underlying ANN units; i.e. does the technique look inside the trained neural net and utilize knowledge about connections and weights or is the network treated as an oracle.
- The extent to which the underlying ANN incorporates specialized training regimes.
- The quality of the extracted rules; i.e. how well the required explanation is performed.
- The algorithmic complexity; i.e. how efficient the underlying rule extraction algorithm is.

Representation languages typically used include (if-then) inference rules, M-of-N rules, fuzzy rules, decision trees and finite-state automata. In the translucency dimension there are two fundamentally different approaches; decompositional

(open-box or white-box) and pedagogical (black-box).

Decompositional approaches focus on extracting rules at the level of individual units within the trained ANN; i.e. the view of the underlying ANN is one of transparency. According to Andrews, Diederich and Tickle, a basic requirement for this category of rule extraction is that the computed output from each unit must be mapped into a binary outcome, corresponding to a rule consequent. Each unit can be interpreted as a step function, meaning that the problem is reduced to finding a set of incoming links whose summed weights guarantee that the unit's bias is exceeded regardless of other incoming links. When such a combination of links is found, this is readily translated into a rule where the output of that unit is a consequent of the inputs. The rules extracted at the individual unit level are then aggregated to form the composite rule set for the ANN as a whole. Pedagogical approaches treat the trained ANN as a black box; i.e. the view of the underlying ANN is opaque. The core idea in the pedagogical approach is to treat the ANN as an oracle and view the rule extraction as a learning task, where the target concept is the function learnt by the

ANN. Hence the rules extracted directly map inputs to outputs. Black-box techniques typically use some symbolic learning algorithm, where the ANN is used to generate the training examples. The easiest way to understand the process is to regard black-box rule extraction as an instance of predictive modeling, where each input-output pattern consists of the original input vector and the corresponding prediction from the opaque model. From this perspective, black-box rule extraction becomes the task of modeling the function from the (original) input variables to the opaque model predictions; see Figure 1.

The first two dimensions (i.e. expressive power and translucency) are in [2] suggested to be the primary classifiers of rule extraction algorithms.

5. EVALUATION OF RULE EXTRACTION ALGORITHMS

There are several criteria used for evaluating rule extraction algorithms. In [4] Craven and Shavlik listed five criteria:

Comprehensibility

The extent to which extracted representations are humanly comprehensible.

Fidelity

The extent to which extracted representations accurately model the networks from which they where extracted.

Accuracy

The ability of extracted representations to make accurate predictions on previously unseen cases.

Scalability

The ability of the method to scale to networks with large input spaces and large numbers of weighted connections.

Generality

The extent to which the method requires special training regimes or places restrictions on network architectures.

Most researchers have evaluated their rule extraction methods using the first three criteria but, according to Craven and Shavlik, scalability and generality have often been overlooked. In the paper, scalability is defined in the following way:

Scalability refers to how the running time of a rule extraction algorithm and the comprehensibility of its extracted models vary as a function of such factors as network, feature-set and training-set size.

Craven and Shavlik argue that methods that scale well in terms of running time, but not in terms of comprehensibility will be of little use. The reason is obvious from the fact that the overall purpose of rule extraction always is to produce a comprehensible model available for human interpretation. If this becomes impossible for larger problems it must be considered a serious limitation for a proposed method.

It should be noted that scaling is an inherent problem, regarding both running time and comprehensibility, for decompositional methods. The potential size of a rule for a unit with n inputs each having k possible values is kn, meaning that a straightforward search for possible rules is normally impossible for larger networks. This, and the problem with continuous inputs, are normally to some extent handled by clustering inputs into disjoint ranges. Craven and Shavlik also highlight that the size of rule sets produced by decompositional algorithms tend to be proportional to the network size.

Craven and Shavlik recommend rule extraction researchers to pursue different lines of research that have not been explored to a large extent, to try to overcome the problem of scalability:

- Methods for controlling the comprehensibility vs. fidelity trade-off; i.e. the possibility to improve the comprehensibility of an extracted rule set by compromising on its fidelity and accuracy.

- Methods for anytime rule extraction; i.e. the ability to interrupt the rule extraction at any time and still get a solution.

Regarding generality, Craven and Shavlik argue that rule extraction algorithms must exhibit a high level of generality to have a large impact. In particular, algorithms requiring specific training regimes or algorithms limited to narrow architectural classes are deemed less interesting. Craven and Shavlik finally say that rule extraction algorithms ideally should be so general that the models they are trying to describe must not even be ANNs. Obviously there is also a need to explain complex models like ensembles or classifiers using boosting, so it is natural to extend the task of rule extraction to operate on these models. A rule extraction algorithm capable of coping with different underlying kinds of models would therefore be of high value.

Yet another important criterion, often overlooked but recognized in [5], is consistency. A rule extraction algorithm is consistent if it extracts similar rules every time it is applied to a specific data set. According to Towell and Shavlik, consistency is important since it would be very hard to give any significance to a specific rule set if the extracted rules vary significantly between runs.

Craven and Shavlik also pointed out another issue they believe to be a key to the success of rule extraction methods, namely software availability; i.e. researchers should make their methods available to potential users and fellow researchers to receive testing and evaluation.

An interesting discussion about the purpose of rule extraction is found in [6], where Zhou argues that rule extraction really should be seen as two very different tasks; rule extraction using neural networks and rule extraction for neural networks. The first task prioritizes accuracy while the second focuses on fidelity. Rule extraction using neural networks thus is aimed at finding a comprehensible model with higher accuracy than a comprehensible model created directly from the data set using, for instance, a decision tree algorithm. Rule extraction for neural networks, on the other hand, is solely aimed at understanding the inner workings of a trained neural network. The claim made by Zhou is that it is never important to obtain both high fidelity and high accuracy; the goal is always one of them, and, consequently the other should be ignored.

6. RELATED WORK CONCERNING RULE EXTRACTION

Since one key contribution of this thesis is a novel method for rule extraction from opaque models, one very important related research area is previously presented rule extraction algorithms. In this section some important rule extraction algorithms are discussed.

Below, three specific algorithms for rule extraction are presented. The motivation for including RX [7] and TREPAN [8] is that they are well-known, general techniques, representing very different approaches.

6.1 Rule Extraction(RX)

With the RX algorithm [7], Lu, Setino and Liu present a decompositional rule extraction algorithm producing Boolean rules from feed-forward ANNs. The description below is a slight simplification of the presented method; for more details see the original paper. Lu, Setino and Liu use an approach consisting of three steps to create classification rules:

1. Network training: A three-layer ANN is trained using standard techniques; e.g. back propagation. The coding used for the classes is localist; i.e. there is one output unit per class. To facilitate the following pruning, it is desirable to have many weights with values so small that they can be set to zero. This is accomplished by adding a penalty function to the error function; for details see the original paper.

2. Network pruning: The fully connected network of step 1 is pruned to produce a much smaller net without raising the classification error "too much". More specifically, links with small weights are iteratively removed and the smaller network is retrained until the accuracy on the training set falls below an acceptable level.

3. Rule Extraction: Rules are extracted from the pruned network. The rules generated are of the form if (a_1 θ v_1) and (a_2 θ v_2)… and (a_n θ v_n) then C_j where a_i's are the attributes of an input instance, v_i's are constants, C_j is one of the class labels and θ is a relational operator.

The process for the actual rule extraction is given in pseudocode below:

Input:
D // Training data
N //Pruned neural network
Output:
R // Extracted rules

Algorithm (RX):

Cluster hidden nodes activation values; Generate rules that describe the output values in terms of the discretized hidden activation values; Generate rules that describe the discretized hidden output values in terms of input values; Combine the two sets of rules; It should be noted that continuous inputs must first be discretized, here by dividing their range into subintervals. Normally, continuous inputs are then coded using thermometer coding. The RX algorithm relies heavily on the success of the pruning since, if a node has n input links, there could be as many as 2n distinct input patterns. Another problem is the fact that the activation value of a hidden node could be anywhere in the range [-1, 1], (assuming a hyperbolic tangent activation function), depending on the input instance. For a large training set this makes the activation function almost continuous. The RX algorithm handles this by discretizing the activation values of hidden nodes into a "manageable" number of discrete values. A small set of discrete activation values makes it possible to determine the dependencies between hidden node and output node values, as well as the dependencies between input and hidden node values. From these dependencies, rules can be generated; in the RX algorithm this is done by using the X2R rule generator [10].

Although the RX algorithm is often cited, and even sometimes used as an example of typical rule extraction; see e.g. [11], it is not widely used in practice. The source code is not publicly

available, which makes it hard to correctly evaluate the performance. It is obvious that RX most likely will fail regarding scalability. The case study reported in the original paper uses a fairly small data set (1000 instances with initially 7 attributes each) and the pruning is very successful; only 17 of 386 links remain after the pruning, while the accuracy is still over 90%. Although most criteria regarding rule quality (comprehensibility, accuracy and fidelity) seem to be well met, this is very hard to judge from just one problem.

Regarding generality, RX is tightly bound to feed-forward ANNs. The demands for repeated training during the pruning phase, a tailored error function and the many "tricks" to get inputs, outputs and activation values into suitable formats also make RX a very specialized algorithm. It should, in addition, be noted that RX extracts rules from one network only. If, as often is the case, the predictive model consists of an ensemble of networks, RX would have to extract from a less accurate model. This is a disadvantage compared to pedagogical methods, which would operate directly on the actual predictive model.

A rule extraction technique based on RX is CaST (Cluster and See Through) [12]. The main idea of CaST is to apply a clustering technique similar to the one used by RX, but to the activation values of the input nodes; i.e. to the input values directly. This alteration in reality makes CaST a black-box rule extraction algorithm because the extracted rules now describe outputs in terms of inputs. Naturally, this makes it possible for CaST to extract rules from any opaque model, not just single feed-forward ANNs. In the study reported in [12], CaST is evaluated against NeuroRule (RX) and C5.0 on three data sets. The main result is that CaST and NeuroRule have almost identical accuracy on two problems, but the rules found by CaST are significantly more compact. On the third problem (Soybean) NeuroRule fails to extract rules since the pruning is not effective enough. CaST on the other hand, produces a fairly complex rule set having higher accuracy than C5.0.

6.2 TREPAN

TREPAN [8] is a pedagogical rule extraction algorithm for classification problems producing decision trees. Each internal node in the extracted tree represents a splitting criterion and each leaf represents a predicted class. TREPAN uses M-of-N expressions for its splits. M-of-N expressions are Boolean expressions in disjunctive normal form, with N conjuncts and the requirement that at least M of these should be true for the entire expression to be true.

TREPAN is a black-box method since it focuses exclusively on the input-output relationship, instead of looking inside the neural net. In a way, TREPAN uses the network as an oracle; i.e. its results are regarded as ground truth. TREPAN grows trees in a best-first fashion, since the node with the greatest potential to increase the fidelity of the extracted tree is expanded first.

The TREPAN algorithm is given in pseudocode below:

Input:
D // Training data
N // Trained neural network

Output:

DT // Extracted decision tree

Algorithm (TREPAN):

Initialize the tree as a leaf node;

While stopping criteria not met.

Pick the most promising leaf node to expand Draw a sample of instances;

Use the network to label the instances;
Select splitting test for the node;
For each possible outcome of the test make a new leaf node;
The task of TREPAN is, according to Craven and Shavlik, to induce the function represented by the trained ANN; i.e. fidelity is the basis of the score function.

Craven and Shavlik [4] describe TREPAN as similar to conventional decision tree algorithms such as C4.5 with some basic differences. Below is a summary of the main properties of TREPAN:

- TREPAN uses the ANN to label all instances. This also means that TREPAN can use the ANN to label new instances and thus learn from arbitrarily large samples.

- In order to decide which node to expand next, TREPAN uses an evaluation function to rank all of the leaves in the current tree. The evaluation function used for node N is: f (N) = reach (N) - (1 − fidelity (N)) where reach (N) is the estimated fraction of instances that reach node N and fidelity (N) is the estimated fidelity of the tree to the network for those instances.

- TREPAN gets a sample of instances from two sources to find the logical test with which to partition the instances that reach the node and to determine the class labels for the leaves. First, it uses the ANN's training examples that reach the node. Second, TREPAN constructs a model (using the training examples) of the underlying distribution and uses this model to draw instances. These instances are randomly drawn but are subject to the constraint that they would reach the node being expanded if classified by the tree. In both cases TREPAN queries the ANN to get the class label.

- TREPAN uses information gain as the evaluation measure when selecting splitting tests.

TREPAN is publicly available, the authors report several case studies [13], and has also been extended in different ways; for instance to return fuzzy decision trees [14]. TREPAN performs well, both in reported studies and in the experiments conducted in this thesis. The accuracy is normally higher than that of models generated directly from the data set; e.g. by C5.0. ANN fidelity is naturally high, since this is the purpose of the algorithm. Regarding comprehensibility it can be argued that decision trees automatically produce good explanation, but for more complex and bushy trees this is questionable. TREPAN handles this by extracting the tree in a best-first fashion and allows the user the option to stop the growth at any level; an example of anytime extraction.

Nevertheless there is still a trade-off between accuracy and comprehensibility. Actually for some of the extracted trees reported [15]. As well as some found during the research for this thesis, the ease of human inspection is questionable. This is arguably partly due to the use of M-of-N rules, which for most people are tricky to read.

TREPAN was, according to Craven and Shavlik, designed with scalability and generality in mind. The scalability criterion naturally favors black-box approaches, since the computational complexity for black-box methods does not depend directly on the network architecture. The node expansion itself is of polynomial computational complexity in the sample size, the number of features and the maximum number of values for a discrete feature. Thus, TREPAN is likely to

scale up well to larger problems, at least regarding computational complexity. Since scalability in comprehensibility requires that growth of the tree is stopped early, the accuracy of the extracted tree is obviously very dependent on that the evaluation function used constantly finds the best splits first. It should be noted that TREPAN in itself does not balance accuracy against comprehensibility, but leaves this decision to the user. The user must choose when to stop the growth of the tree, or put in another way, which tree of several, all with different complexity, to actually use on novel data.

Regarding generality, TREPAN does not really require the underlying model to be an ANN. There are no reports, though, of studies where the original TREPAN program is used for rule extraction from anything else than ANNs. In the third-party implementation later used for experimentation in this thesis, it was, however fairly easy to convert TREPAN into a true black-box rule extraction algorithm.

6.3 Rule extraction using evolutionary algorithms

Dorado et al. in [9] present a novel approach to rule extraction based on evolutionary algorithms. One should note that the two methods were developed independently1.

The algorithm suggested by Dorado et al. is a black-box method where the extraction strategy is based on GP. The algorithm can handle different representation languages with ease, since the choice of representation language corresponds to the choice of function and terminal sets. In the paper, Dorado et al. give three examples, two where Boolean rules are extracted for classification problems and one where a mathematical function, similar to, but more complex than a Box-Jenkins model, is derived for a time series forecasting problem. The suggested approach is also compared (with good results) to several existing techniques on well-known problems. A key result reported by Dorado et al. is the comparison between the rule extraction algorithm and GP applied directly on the data set. The accuracy of the rule extraction is slightly higher, supporting the claim that the neural net in a sense is a better (more general) representation of the data than the data set itself.

The purpose of the proposed method is to use GP for the search process. More specifically, candidate rules (which could be Boolean rules, decision trees, regression trees etc.) are continuously evaluated according to how well they resemble the ANN. The best rules are kept and combined using genetic operators to raise the fitness (performance) over time. After many iterations (generations) the most fit program (rule) is chosen as the extracted rule.

It should be noted that the fitness function is crucial for determining what to optimize, and that the choice here is to solely optimize the fidelity of the extracted representation to the neural net.

Dorado et al. do not state the algorithm explicitly, so the description below is based on an interpretation of their paper.

Input:
D // Training data
N // Trained neural network

F // Function set
T // Terminal set

Output:
R // Extracted representation

Algorithm: (Rule extraction using EA)
Initialize a first generation of candidate rules;
While number of generations not reached
Evaluate all candidate rules using the fitness function (fidelity);
Choose a number of candidate rules for reproduction;
Combine chosen candidate rules using genetic operators (crossover) to create offspring rules;
Replace old candidate rules with offspring rules;
The reported accuracy for the approach reported by Dorado et al. is high. Since GP is a recognized tool for optimization problems and fidelity to the neural net is the optimization criteria here, it is no surprise that the fidelity is high. The problem for the algorithm of Dorado et al. is clearly comprehensibility.

Regarding time scalability, the proposed method is likely to perform well. Although GP is rather computationally intensive, this applies even more for the original training of the neural net, making it unlikely that the rule extraction would be the bottle-neck. The computational complexity of a GP approach is dependent on parameters like the number of programs in each generation, and to a lesser degree, the size of each program. This could potentially be a difficulty for very complex problems, where large generations and/or programs are needed. This is obviously an issue that should be looked in to.

How well comprehensibility scales up is a totally different matter. Since Dorado et al. do not try to enforce short rules, complex data sets with many variables will inevitably produce long and complicated rules.

The generality of the proposed approach is very high and is actually the most appealing property of the method. Dorado et al. apply the rule extraction on both feed-forward and recurrent networks and extract rules for both classification and regression. It is also obvious that, even if the authors do not explicitly point this out, the algorithm does not require the underlying model to be a neural net.

3. CONCLUSIONS

The main contribution of this paper was to show the benefit of using test set data instances, together with predictions from the opaque model, when performing rule extraction. The technique evaluated means that the same novel data instances used for actual prediction also are used by the rule extraction algorithm. As demonstrated in the experiments, rules extracted using only oracle data were significantly more accurate than both rules extracted by the same rule extraction algorithm (using training data only) and standard decision tree algorithms. The overall implication is that rules extracted in this way will have higher accuracy on the test set; thus explaining the predictions made on the novel data better than rules extracted in the standard way; i.e. using training data only.

ACKNOWLEDGEMENTS

I would like to all our lab coordinators and Sirt University who helped in all to succeed this research paper.

REFERENCES

[1] Lee, S.hyun. & Kim Mi Na, (2008) “This is my paper”, ABC *Transactions on ECE*, Vol. 10, No. 5, pp120-122.

[2] R. Andrews, J. Diederich and A. B. Tickle, A survey and critique of techniques for extracting rules from trained artificial neural networks, Knowledge-Based Systems, 8(6).

[3] M. Craven and J. Shavlik, Using Neural Networks for Data Mining. Future Generation Computer Systems: Special Issue on Data Mining, pp.211-229.

[4] M. Craven and J. Shavlik, Rule Extraction: Where Do We Go from Here?, University of Wisconsin Machine Learning Research Group working Paper 99-1.

[5] G. Towell and J. Shavlik, The extraction of refined rules from knowledge based neural networks, Machine Learning, 13(1):71-101M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[6] Z.-H. Zhou, Rule Extraction: Using Neural Networks or For Neural Networks?, Journal of Computer Science & Technology, 19(2):249-253, 2004.

[7] H. Lu, R. Setino and H. Liu, Neurorule: A connectionist approach to data mining, Proceedings of the International Very Large Databases Conference, pp. 478-489, 1995. Article in a conference proceedings:

[8] M. Craven and J. Shavlik, Extracting Tree-Structured Representations of Trained Networks, Advances in Neural Information Processing Systems,8:24-30.

[9] J. Dorado, J. R. Rabunãl, A. Santos, A. Pazos and D. Rivero, Automatic Recurrent and Feed-Forward ANN Rule and Expression Extraction with Genetic Programming, Proceedings 7th International Conference onParallel Problem Solving from Nature, Granada,

[10]. H. Liu, X2R: A fast rule generator, Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vancouver.

[11]. M. Dunham, Data Mining – Introductory and Advanced Topics, Prentice Hall, 2003.

[12]. T. Löfström, U. Johansson, and L. Niklasson, Rule Extraction by Seeing Through the Model, 11th International Conference on Neural Information Processing, Calcutta, India, pp. 555-560, 2004.

[13]. M. Craven, Extracting Comprehensive Models from Trained Neural Networks, Ph.D. Thesis, University of Wisconsin-Madiso.

[14]. M. Faifer, C. Janikow, and K. Krawiec, Extracting Fuzzy Symbolic Representation from Artificial Neural Networks, Proceedings 18th International Conference of the North American Fuzzy Information Processing Society, New York, NY, pp. 600-604.

[15]. M. Craven and J. Shavlik, Understanding time-series networks: A case study in rule extraction, International Journal of Neural Systems, 8(4):373-384.

Author

Tiruveedula GopiKrishna
Lecturer,
Faculty of Arts and Science
Sirt University,
Department of Computer Science,
Hoon,Aljufra,
Libya