Problem Solving with Limited Knowledge: Pruning Attributes in Classification Rules

Kamran Karimi and Howard J. Hamilton Technical Report CS-2004-04 June, 2004

Copyright © 2004 Kamran Karimi and Howard J. Hamilton Department of Computer Science University of Regina Regina, Saskatchewan CANDA S4S 0A2

Problem Solving with Limited Knowledge: Pruning Attributes in Classification Rules

Kamran Karimi and Howard J. Hamilton

Department of Computer Science University of Regina Regina, Saskatchewan Canada S4S 0A2 {karimi, hamilton}@cs.uregina.ca

Abstract

Deriving algorithms to solve problems is a main activity in computer science. Elaborate techniques are often devised to efficiently handle specific problems. This paper proposes the use of a general method of solving problems, called the Iterative Multi-Agent (IMA) Method, which assumes little about the specific problem at hand. In IMA the problem is divided into overlapping sub-problems, and an agent is assigned the task of solving each of them. The agents iterate over the solutions to the sub-problems in the hope of finding a solution that satisfies the whole problem. If noticing a clash with another part of the problem that was solved by another expert, each expert revises its solution, hoping the new one will remove the clash. This is repeated unless the problem is solved with no clashes, or when a specific number of iterations are performed. In IMA no central authority knows about the whole problem, or even how each agent's work may affect other agents' work. By design, IMA agents can run in parallel, solving the often hard problem of coming up with a parallel algorithm. The agents need minimum amounts of communication and synchronisation, making the method scalable on hardware systems with many processors. To demonstrate IMA's adaptability, in this paper we tackle the problem of extracting rules by pruning (removing) the condition attributes from a series of classification records. This is also called attribute reduction. Each record contains an observed set of condition attributes, and the aim is to predict a class based on their values. In line with IMA's approach, there are many agents in the system, each responsible for a single class. These agents employ a very simple method, and transform the input records into rules by independently changing them to better comply with two laws: The rules should be short, and they should not misclassify. The agents try to make the rules that predict their assigned class as short as possible by deleting some of the condition attributes and turning them into don't-cares. As the condition attributes are deleted, the rules become more general, and match more input records. The agents also have to make sure that these generalised rules do not wrongly match the records with a different class value. They achieve this by undeleting some of the condition attributes, thus making the rules more specific, and hence removing the false classification. The deleting and undeleting of the attributes are the only operations performed. The aim of the paper is to make an argument in favour of simplicity and adaptability of IMA, and to show that simple algorithms can lead to acceptable results. We compare the results obtained with IMA to those of C4.5 and Random Pruning.

1. Introduction

Striving for simplicity of methods, both conceptual and at the implementation level, seems to be against the mainstream of research tendencies in computer science. The observed effect is that in most cases, elaborate and complex algorithms result in better run-time and accuracy. We argue that there is value in simplicity, and this opinion is shared by some other researchers [4]. We will present a general problem solving method that is not geared to any specific category of problems and use it to solve a certain problem in rule discovery. The simplicity of the method makes it easy

to be understood and adapted to new problems. It is a parallel method by design, making it easier for users to exploit parallel or distributed hardware systems [6]. The aim of the paper is to showcase the versatility and simplicity of the proposed method, and not to display its superiority to other existing, special-purpose tools and methods.

Trying to understand and therefore predict the behaviour of a system under investigation often requires extracting rules that describe the relationship between the input and the output of the system. Here we consider the system to have a single output. One example of such a system is a classifier, where a set of input values determine a single output (class). Seen as a black box, input values are provided to the system, and then the output is noted. The input values are called condition attributes, while the output value is called the decision attribute. We save the condition attributes and the corresponding decision attribute in individual records. The next step would be to extract shorter rules from these records.

Learning rules by pruning the condition attributes, also called attribute reduction, in a set of input records, has been the focus of a lot of attention. Not only it makes for shorter records, it may also allow us to reduce the number of records by deleting the less general ones. Many methods have been used to perform classification, including genetic [5] algorithms and neural networks [2, 3]. One well-known program to do this is C4.5 [10]. It uses the information contents of the condition attributes to select the best ones for building a decision tree. This decision tree is then used to generate rules. In many cases the resulting rules do not match the input records exactly, and so we will be faced with a loss of accuracy. Here we investigate a system of attribute pruning that is potentially loss-free, while giving a respectable predictive accuracy when applied to unseen data. The emphasis is on simplicity and reducing the complexity of the algorithm. We are interested in a high performance/complexity ratio, as opposed to high performance alone. Our criteria for measuring the complexity of an algorithm are its conceptual complexity (how long does it take to explain the algorithm to someone) and ease of implementation (how long does it take to write code for it).

The Iterative Multi-Agent (IMA) Method [6] is based on the idea that to solve a problem one may be able to segment it into possibly overlapping sub-problems and then assign each to an agent. Thus only the sub-problems are solved, hoping for the solution to the whole problem to emerge as a result. This is much like a team of experts working on a big project. Each expert by himself does not understand the whole problem, but can only solve a part of it. Figure 1 portrays this. To increase the chances of finding a solution to the whole problem, the designer should make sure the sub-problems cover the whole problem. IMA can function even if this condition is not met, but the results may not be correct. This is because IMA is a "best-effort" method, and does not guarantee valid and optimum results. Iteration is used since each agent by itself does not know what the effects of its actions are on the whole problem, or when it is solved.

There is usually a population of solution candidates, which are solutions to the sub-problems that may or may not be correct. These solution candidates are put into a workpool and are iteratively revised by independent agents, which inspect them and if needed, change them. The order in which the contents of the workpool are visited is random.

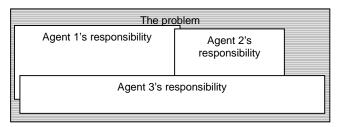


Fig. 1. Segmenting a problem into possibly overlapping sub-problems.

Having many solution candidates means that the agents can work in parallel. Each agent sleeps for a random amount of time before getting the next partial solution from the pool. This ensures that they do not work on the partial solutions always in the same order. In a parallel or distributed hardware system, this wait may be happening naturally because of synchronisation and network speed issues. Figure 2 shows a sample workpool and two agents.

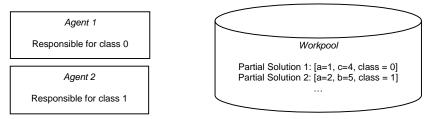


Fig. 2. An example workpool with two agents shown.

In [6] we tackled a general form of Constraint Satisfaction Problem (CSP) with IMA, and in [7] we solved the Travelling Salesperson Problem (TSP) and compared the results with the brute-force and the greedy methods. This shows the adaptability of the IMA method for solving very different optimisation problems.

IMA suggests a set of general guidelines for solving problems. This means that solving a specific problem requires an investigation as to how the IMA method should be applied to it. There are two requirements for IMA to be applicable. First, the problem should be decomposable into subproblems, otherwise the multi-agent nature of the method disappears. Each agent can use any algorithm to solve its assigned sub-problem. Second, at least some of the sub-problems should have more than one possible solution, otherwise the iteration element disappears. This is because one specific solution to a sub-problem may not be compatible with solutions to other sub-problems, hence the need to try another solution.

Having more than one possible solution for the sub-problems allows the system to try more than one solution to the whole problem. The number of iterations, determined by a domain expert, is very important because a higher number of iterations allows the system to explore more combinations of partial solutions. The program can stop executing either when the problem is solved or when the number of iterations reaches a limit set by the user.

Applying the IMA method to a problem requires answering the following five questions: 1. How the problem should be segmented. 2. How the solution candidates are to be represented. 3. Which algorithm should be used to solve each sub-problem 4. How the agents should verify that each subsolution is satisfactory. 5. How the whole solution is verified.

In this paper we apply IMA to the task of generating classification rules from observed input records. Each record has a set of condition attributes and a single decision attribute that takes on different values to represent different classes. All the attributes are integers, even though any other

symbolic values would do. An example record could be: (a = 1), (b = 5), (c = 4), (class = 12). The aim is to come up with a set of rules that numbers less than the number of the input records and that do the same classification job. An example rule could be: if $\{(a = 1) \text{ AND } (b = 5)\}$ then (class = 12), where the condition attribute c has been deleted and considered a don't-care. We do this transformation of the input records into rules with a very simple heuristic. A deleted attribute can be later undeleted. Records with unknown attribute values are not handled in this paper, but they can be considered as permanently deleted.

The aim in this paper is to know how far we can go in attribute pruning without using statistics or any other form of complex algorithms. The rest of the paper is organised as follows. Section 2 explains how the IMA method was applied to attribute pruning. In Section 3 we tackle two different problems: One with data concerning the movements of an artificial robot, produced by an Artificial Life simulator, and the other with more complex data from a letter recognition dataset. We present IMA's results along with those of C4.5. Since there are elements of randomness in IMA, we also compare it to a purely random method of pruning the attributes. Section 4 concludes the paper.

2. IMA'ing the Problem

We now have to answer the five questions in Section 1. We do not consider an input record and a rule to have any fundamental differences. The input to the program is a set of records, while the output is considered to be composed of rules. Instead of creating a decision tree as an intermediate step we will directly transform the input records into rules by deleting as many attributes as possible. These generalised records are then presented to the user as the results.

How the problem should be segmented? The segmentation is done according to the value of the decision attribute, or the class. If the decision attribute would take on values from 0 to 15, for example, then the input records will be virtually divided into 16 sets. 16 agents will be created and each assigned one class. Each agent tries to shorten the records that predict its class and at the same time prevent any other generalised records from matching its assigned class. To increase the potential for concurrency beyond the number of classes, we could simply add a new field to the input records, and assign it an agent's number. Thus the number of agents could be more than the number of classes. This would allow the user to utilise parallel execution means as efficiently as possible.

How the solution candidates are to be represented? The input records form a workpool, and each record is a solution candidate. They are represented as they are, but the system now adds more fields to the records, so it is now possible to flag a condition attribute as *deleted* or *present*. An attribute can become deleted and undeleted many times. Deleted attributes are don't-cares, meaning that they are skipped when verifying if the rule can be fired or not. At the beginning of a run, all the condition attributes are flagged as deleted. This means that all the rules apply all the time. As explained later, this deliberate act causes the agents to scramble at the start of the run to change the rules and keep them as consistent as possible.

Which algorithm should be used to solve each sub-problem? We are employing a very simple method here. The input records are copied into the workpool as the initial rules. They will gradually become more general by deleting some condition attributes from them. This means that initially they represent the worst-case scenario of each record being a single rule. As the rules become more general, they start to cover the condition attributes of other rules. This will allow us to prune the rules that are covered by the more general ones and end up with a less number of rules in the output.

Each agent is responsible for two things. First, the rules that predict its assigned class should be as short as possible. The second responsibility deals with misclassification: the condition attributes of a rule that predicts another class should not match the condition attributes of any rule that predicts its class. The following example makes this point clear. Consider the following two rules: Rule number 1: (a = 1), (b = 2), (c = 5), (class = 12). Rule number 2: (a = 1), (b = 3), (c = 5), (class = 8)

Now suppose the agent responsible for class 12 deletes (b = 2) from the first rule and makes it more general than the second rule because it now has the form (a = 1), (c = 5), (class = 12). The rule still predicts the correct class for the record, so the first agent has progressed towards the goal of shortening its rules.

After a number of iterations the agent responsible for class 8 will get the shortened rule number 1 from the workpool and starts to verify that it does not misclassify its assigned rules. It will then realise that the left hand side of rule number 2 matches the shortened rule 1, but the class predicted by rule 1 is 12, not 8, so rule 1 is misclassifying. The agent for class 8 undeletes a variable in rule number 1 to prevent this from happening. A rule that misclassifies a more general one could be considered an exception. However we avoid them because keeping the general rule requires dealing with multiple exceptions. Lacking any domain knowledge, we prefer to keep the rules from interfering with each other as much as possible.

The next time that rule number 1 is visited, another attribute is deleted by the agent responsible for class 12, and maybe that will not raise any conflicts. This process is repeated for all the rules in the workpool for a predetermined number of iterations. The rules thus keep changing and adjusting. As will be seen later, the higher the number of iterations, the more the chances of a better outcome.

If extracting a single record from the workpool proves to involve an unacceptable overhead in terms of time and hardware resources, each agent can retrieve a group of records and work on all of them before returning them to the pool.

How the agents should verify that each sub-solution is satisfactory? When an agent discovers a mismatching rule, it undeletes one variable in the hope that the mismatch will be resolved, and sets a flag. When the agent responsible for that rule encounters it, it notices the set flag. Realising that this rule has not *settled* yet, it deletes some other variables hoping to continue to shorten the rule, and knowing that given enough number of iterations, the newly changed rule will continue to pass more tests by other agents. The deleting and undeleting of the attributes is done in a round-robin fashion. This is better than a random method because it ensures that more of the search space is explored. Once a rule does not have any of its flags set, it is left unchanged. The agents gradually reduce the number of attributes they delete (compared to the number of attributes that are undeleted) so that the rules will converge to a stable state. This process is reminiscent of simulated annealing [9]. The side effect is that more emphasis is put on the undeleting process, resulting in the rules having a very good fit on the training data. The emphasise on undeleting also increases the number of rules at the output.

Since at the start of a run all the attributes in the workpool are flagged as deleted, there is great pressure on all the agents to stop the misclassifications. So a large number of attributes will be deleted or undeleted at the start. Things will slow down as the run continues.

How the whole solution is verified? After the number of iterations in the system reaches the limit, the rules are pruned so that only the more general ones remain. This is done only for the rules that predict the same class. The resulting rules are then tested against the original records and the testing records to calculate the accuracy values.

3. Experimental Results

A program was implemented in Java to try IMA on two sets of input records. IMA is suitable for parallel or distributed execution. Each of the agents in the program is a separate Java thread, and thus runs in parallel to other agents. This can result in automatic speedup if the program is run on a multi-processor computer that maps the Java threads to the native operating system's threads, allowing them to be executed on different processors. Alternatively, one could re-write the code in another language to achieve the same effects. Many of the support routines in the implementation were taken from [6] and [7] resulting in short coding and debugging phases. The first set of input records comes from a simple world of an Artificial Life simulator, and the second contains records from a "real world" letter recognition application which has been widely used for benchmarking purposes. In both cases we use C4.5 as a reference because it is a well-known program and uses an elaborate greedy heuristic, guided by information theory formulas, to prune attributes. C4.5 was invoked with default argument values. At the other end of the spectrum, we provide the results of purely random attribute pruning that works by deleting the attributes with certain probabilities. IMA stands in between these two methods both in terms of complexity of the algorithm and the ease of implementation.

3.1 Artificial Creature's Moves

In the first case study, we used IMA to prune the records produced in a simple artificial life environment called URAL [11]. The world in URAL is inhabited by one or more creatures. This world is a two dimensional, 15 by 15 board with 10 obstacles placed on it randomly. A creature moves from one position to the next using Up, Down, Left, and Right actions. The move is successful unless an obstacle is encountered, or if the edges of the world are reached. In both cases the creature remains in the same place as before the move. There are a total of 6 condition attributes: The previous x value, previous y value, an indication of whether or not food was present at that location, the humidity and temperature at that location, and the move direction. The decision attribute is the resulting x value. From the semantics of the domain we know that the next x value depends on the previous x and y values and also the move direction [8].

There were a total of 7,000 input records. The first 5,000 were used for training, and later to measure the training accuracy (T Accuracy). The remaining 2,000 were used to measure the predictive accuracy (P Accuracy). The number of rules after pruning is also counted. The results are shown in Table 1. C4.5's results for the same input are given as reference.

Method	Iterations	T Accuracy	P Accuracy	# of Rules
IMA	50,000	7.0%	6.6%	480
	100,000	15.2%	13.8%	370
	200,000	32.4%	30%	341
	500,000	98.8%	97.4%	362
	750,000	99.7%	98.3%	461
	1,000,000	100%	98.5%	458
C4.5	N/A	99.8%	99.8%	76

Table 1. IMA and C4.5's results for the URAL records

With IMA, the minimum possible number of rules is 15 (all condition attributes deleted) and the maximum is 5,000 (the rules are the same as the input records). Each row shows the results of a different run as opposed to the results of a single run at different iteration points. The accuracy starts to stabilise at around 500,000 iterations. With IMA the training accuracy reaches 100% while the predictive accuracy gets very close to C4.5's results. IMA stops modifying the rules after the training accuracy reached 100%, so increasing the number of iterations after that will not have any effects.

We also tried a purely random method to delete the attributes. The training and testing data were as before. For each rule, each attribute would be deleted with a certain probability. After this deleting phase, the rules that were covered by more general ones were purged out and the rest were used to measure the training accuracy and the predictive accuracy. 10 runs were tried for each probability value and the average results are shown in Table 2.

Method	Probability	T Accuracy	P Accuracy	# of Rules
Random	0%	100%	95.1%	818
	1%	95.4%	91.5%	810
	2%	94.0%	89.1%	782
	5%	89.9%	85.5%	758
	10%	81.5%	77.4%	723
	20%	59.5%	53.3%	638
	30%	21.4%	18.7%	500
	50%	7.0%	6.5%	178
	75%	3.8%	3.7%	16

Table 2. Results of randomly pruning the attributes

There are 818 unique records among the 5000 input records. Using them unchanged gives a predictive accuracy of 95.1%. Both the training accuracy and the predictive accuracy decline as the probability of deleting an attribute increases.

3.2 Letter Recognition Database

For the second set of tests we applied the same program to the Letter Recognition Database from University of California at Irvine's Machine Learning Repository [1]. It consists of 20,000 records that use 16 condition attributes to classify the 26 letters of the English alphabet. The first 16,000 records were used for training and measuring the training accuracy. The remaining 4,000 records were used to measure the predictive accuracy. C4.5's results are also provided as reference. The results come in Table 3.

Method	Iterations	T Accuracy	P Accuracy	# of Rules
IMA	100,000	4.8%	4.2%	528
	500,000	10.3%	9.8%	4497
	1,000,000	29.6%	25.6%	6355
	2,000,000	53.6%	42.2%	8403
	3,000,000	85.6%	63.2%	9462
	5,000,000	99.4%	70.3%	9688
	8,000,000	100%	70.4%	9694
C4.5	N/A	86.2%	73.5%	1483

Table 3. IMA and C4.5's results for the Letter Recognition Database

IMA can potentially output a maximum of 16,000, and a minimum of 26 rules. The same trends in Table 1 are seen here: The training accuracy reaches 100%, at which point the agents stop modifying the rules. The predictive accuracy gets close to that of C4.5, while the number of rules is higher than that of C4.5. This is because the agents in IMA actively try to prevent misclassification on training data, which is reflected in better training accuracy values. Using the same training and testing records, the authors of the database report achieving a predictive accuracy of about 80% with genetic classifiers [2]. The random method was used on the same data. 10 runs were tried for each probability value. The average values come in Table 4.

Method	Probability	T Accuracy	P Accuracy	# of Rules
Random	0%	100%	9.5%	15072
	1%	100%	9.6%	15042
	2%	99%	9.6%	14993
	5%	99%	9.7%	14911
	10%	98.2%	10.4%	14819
	20%	98%	13.7%	14734
	30%	97.3%	18.4%	14636
	50%	47.6%	23.5%	13616
	75%	3.1%	3.4%	26

Table 4. Results of randomly pruning the variables

The first row shows how the records perform in their original form. Note that there are only 15072 unique records. The random method of attribute pruning deteriorates with this more complex problem. The training accuracy remains at a high level for a while as the probability of deleting an attribute increases, but the predictive accuracy is low. The predictive accuracy gradually starts to get better with higher values of probability, but at the expense of the training accuracy's decline. Both collapse at very high probability values.

With IMA it is possible to change the rate of convergence. In the current implementation it is represented by a number bigger than 1 and determines the ratio of the undeleted attributes to the deleted attributes in a rule. No convergence happens with a ratio of 1. Changing the ratio causes the training accuracy to reach 100% after a different number of iterations. For example, increasing the convergence rate will cause the predictive accuracy to reach a lower peak, and we end up with a higher number of rules. This means that the user has a choice between the running time on one side and the quality of the results on the other. The results in Tables 1 and 3 were obtained with a convergence rate of 1.1. In the Letter Recognition database, 2,000,000 iterations with a convergence rate of 2 would result in a training accuracy of 94.7%, a predictive accuracy of 50.9%, and 10,391 rules. With more iteration the training accuracy would reach 100% while the predictive accuracy would peak at less than 53%. This represents a rapid convergence to the maximum training accuracy (100%) at the expense of the predictive accuracy and the number of rules.

A training accuracy of 100% is attributed to over-fitting of the rules. We do not stop the algorithm from reaching this stage because its rate of improvement is much better as it gets close to the saturation point. The IMA method seems like a good way to *compress* the training data in a way that the training accuracy is as high as needed while the predictive accuracy is not affected too badly. As an example, as seen in the first row of Table 4, there are 15,072 distinct records among the 16,000 input records. Considering this number, with IMA we are seeing a 36% reduction in the number of rules with no loss in the training accuracy.

4. Concluding Remarks

The main idea in IMA is that different agents, solving different sub-problems, can come up with a solution to the whole problem in parallel with no need for direct communication or knowledge sharing. This method is especially useful for solving vague or complex problems, where not enough information is centrally available.

We deliberately refrained from employing any statistical or knowledge-based methods in the IMA implementation in this paper. Much like the philosophy behind Genetic Algorithms, we used a set of simple operations to prune the attributes in sets of input classification records. We observed that IMA's results are reproducible in different runs with little or no variation.

We tried IMA, C4.5, and random variable pruning. Each of these methods has some strong and weak points (the random method is very fast, for example). Considering performance alone, in our tests IMA could not beat an elaborate algorithm like C4.5 in terms of the predictive accuracy, the number of rules generated, or (with larger iteration values) the running time. However its results are reasonable. We certainly did not spend much time deriving the algorithms and implementing the system. Considering the very simple algorithm that was behind the IMA method, it has a very good performance/complexity ratio.

It should be noted that the implementation presented in this paper is just one possible way of segmenting and solving the problem with the IMA method. The agents can apply any algorithm with any degree of complexity when solving their sub-problems. This implies that other implementations may have different results. For example, the current implementation obviously favours a higher training accuracy instead of lowering the number of rules. Changing this bias could reduce the number of rules and change the training accuracy and the predictive accuracy in the output.

As a general problem solving method, IMA can be of value in situations where a "good-enough" solution would do and the cost of designing and implementing an elegant algorithm is high, or a solution to the whole problem cannot be found. Its ability to benefit from parallel or distributed hardware with conceptual ease makes the utilisation of such hardware easier. The user can quickly come up with an implementation that solves a problem for which no suitable tool is already available. The gradual and consistent improvement in the output means that IMA could also be useful when an any-time algorithm is needed, where results may be needed at any time during the run of the system. The user can interrupt the run and use the results at any moment without having to wait any further for full convergence.

References

- 1. Blake, C.L and Merz, C.J., UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.
- 2. Compiani, M., Montanari, D., Serra, R., Valastro, G., Classifier Systems and Neural Networks, *Parallel architectures and Neural Networks*, World Science, 1990.
- 3. Frey, P. W. and Slate, D. J., Letter Recognition Using Holland-style Adaptive Classifiers, *Machine Learning*, Vol 6 (2), March 1991.
- 4. Janikow, C.Z., A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning 13*, pp. 189-228, 1993.
- 5. Mitchel, M., An Introduction to Genetic Algorithms, MIT Press, 1996.
- 6. Karimi, K., The Iterative Multi-Agent Method for Solving Complex Search Problems, *The Thirteenth Canadian Conference on Artificial Intelligence (AI'2000)*, Montreal, Canada, May 2000.
- 7. Karimi, K., Hamilton, H. J. and Goodwin, S. D., Divide and (Iteratively) Conquer!, *International Conference on Artificial Intelligence (IC-AI'2000)*, Las Vegas, Nevada, USA, June 2000.
- 8. Karimi, K. and Hamilton, H. J., Learning with C4.5 in a Situation Calculus Domain, *The Twentieth SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence (ES2000)*, Cambridge, UK, December 2000.
- 9. Kirkpatrick, S., Gerlatt Jr, C.D. and Vecchi, M.P., Optimization by Simulated Annealing, *Science*, 220, 1983.
- 10. Quinlan, J. R., C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.
- 11. http://www.cs.uregina.ca/~karimi/URAL.java