Parameter Selection Using Case-Based Reasoning Guided by Reinforcement Learning

Paul Juell
Computer Science Department
North Dakota State University
juell@plains.nodak.edu

Patrick Paulson
Computer Science Department
North Dakota State University
ppaulson@plains.nodak.edu

Abstract

This paper describes a system that uses case-based reasoning methods to choose parameters settings for industrial processes applications. The system uses reinforcement learning techniques to learn the cases which have the best parameter values to produce a desired set of outputs from the industrial process. The results of a simulation are presented which shows that the system can learn an arbitrary similarity function to a high degree of certainty.

Contents

List of Figures

1 Introduction

This paper describes a case-based learning system adaptable to many applications, including the selection of parameters for industrial processes. The type of problem this system solves can be illustrated by the following scenario. In an industrial process setting, a process has a number of parameters that affect the outcome of the process. The process has been run for a number of different parameter settings and the outcomes recorded. In order to produce a particular outcome, we would like to determine which set of parameters has produced an outcome which is closest, in some sense, to the desired outcome. There is, however, no explicit means of determining the 'closeness' of two outcomes. That is, there is some aspect of the outcome that is desired, and it is readily apparent to the users of the process, but they are unable to describe what it is about the outcome that is important.
The first section of this paper will present background material on case-based reasoning and reinforcement learning. Previous related work will also be described. The approach will be described in the next section. Then it will be applied to a simulated industrial process application.

2 Background

2.1 Case-based reasoning

2.1.1 *Description*

In *case based reasoning*, the current situation is compared against a library of cases. The most similar cases in the library are adapted to the current situation to provide a starting point to a solution for the problem at hand. A straight-forward case-based system for a parameter selection problem, for example, would store the correct selection of input parameters for each possible desired outcome. Then finding the correct parameters for any outcome would be a matter of looking them up in the case library.
One of the advantages of case based reasoning is that it can be used in areas where there is not a strong theoretical model of the domain, such as economics, medicine, and law. [6, page 27]. Previous approaches, such as expert systems, would require analyzing the problem domain in order to derive rules which could be used to develop solutions to problems. The needs of case-based systems are much simpler. They only need a library of good solutions to previously encountered problems.
Case-based reasoning can also be helpful in domains were there is a strong theoretical model. Cases can be used to quickly propose solutions to complex problems, rather then solving the problems from scratch using domain knowledge. So, for example, rather then coming up with a new design for a building that meets designated criteria, a case library can be searched for other buildings that meet the criteria.

2.1.2 *The Indexing Problem*

The indexing problem in case based reasoning consists of the *matching problem* and the *retrieval problem*[6, pp. 19-20].
The matching problem is determining which features are important when selecting cases that match the current situation. These features may consist of surface features and derived features. Surface features are readily apparent in the description of the case. Derived features are obtained by performing calculations on the surface features. The problem of determining features can be acute, considering that we want to use case-based methods in areas in which theoretical knowledge is limited [4].

2.2 Reinforcement Learning

2.2.1 *Description*

In *reinforcement learning* an agent learns behavior through interaction with an environment. The environment provides a reinforcement signal which indicates the desirability of the current behavior of the system [5]. Reinforcement learning can be contrasted with supervised learning, in which the environment indicates to the agent the correct action to take in a situation [13]. Reinforcement learning has been termed *learning with a critic*, as opposed to supervised learning, or *learning with a teacher*.

Reinforcement learning is more difficult than supervised learning, since there is more information about correct outputs available in supervised learning. Reinforcement learning is also more applicable, since a supervised learning problem can be turned into a reinforcement learning problem by defining the reinforcement signal to be a scalar value derived from the difference between the current and the desired output. Reinforcement learning is also suitable in environments were many outputs may be equally valid. In this case the system can determine its own outputs rather than be required to learn some arbitrarily selected set of outputs [14]

An advantage of using reinforcement learning for parameter selection is that is not necessary for the user to know precisely which values the agent should select in order to obtain acceptable outputs. Rather, some measure of the the process outcome resulting from the agent's selection of parameters can be used as reinforcement signal.

In some environments there is more information available than just a reinforcement signal. In parameter selection, for instance, an expert may be available to advise the agent on the correct parameter settings to meet particular requirements. As mentioned above, reinforcement learning can still be used in such situations. But it will take longer than supervised learning since information will be lost in converting the parameter selection information into a scalar reinforcement value.

In a typical reinforcement learning model, at each step of interaction at time t, the agent is presented with a state x_t. In response, the agent produces an action a_t, which is passed to the environment. The environment responds with a reinforcement signal r_{t+1} along with a new state, x_{t+1} [5].

2.3 Related work

There has been much research in the development of similarity functions for classification research. In [4], cases are broken into component parts. These parts are retrieved to create plans to solve subgoals of new problems. The new plan has a set of constraints and actions designed to meet those constraints. The retrieval strategy is altered when the new plan fails, that is, when it has an inconsistent set of constraints.

Much research has also been done in the selection of which features to include when evaluating similarity. [12] and [1] both present surveys of approaches to this problem. [10] discusses a genetic technique called *random mutation hill-climbing* to select features. [3] uses hill-climbing methods to select attributes to use for classification using decision trees. [7] uses a probabilistic approach to feature selection. [11] uses a genetic approach.

[1] and [9] both describe methods for assigning weights to features depending on their importance in discriminating which class instances belong to.

[8] describes another technique for altering a similarity metric using reinforcement learning. This system adjusts two sets of weights using reinforcement learning. A *local* similarity function is used, which depends on the point in the input space from which the distance is taken. This allows conditions to be expressed such as "if feature A is greater than 70% then use only feature B and C to compute similarity". The use of multi-layer neural nets in the current design should also allow such conditions to be expressed.

3 Overview of System

The systems described in this paper use a case library containing the inputs and outputs of past processes. Given the desired outputs, the system searches for cases that have similar outputs. The acceptability is graded by the user and this value is fed back to the system to allow it to adapt the similarity metric it uses for future problems.

In the research described in this paper, a simulation of the above steps is used to test solutions to this problem. In the simulation, a known function is used to determine whether outputs suggested by the system are acceptable, and this value is fed back to the system for learning. Two functions are used to determine similarity in the simulations. The first is the inverse of the Euclidean distance. The second is an arbitrary function, with random values used for the similarity between parameter vectors.

4 General approach of solutions

The systems discussed in this paper use a case-based approach to solving the problem of selecting parameters for a desired outcome of a physical process. A case library is used to hold known cases of process outcomes. Each case consists a vector of the parameters used for the process and a vector of outcomes for that process. When presented with a vector of desired outcomes, the system examines each case in the case library. The vector of outcomes of each case is compared with the vector of desired outcomes using a similarity metric. A case is chosen with a probability proportional to the similarity metric. The values of the parameter vector stored with this case are then suggested to the user as parameter settings to use for the physical process.
When selecting a library case, the system takes into account the the amount of experimentation it has done when previously presented with the currently desired outcomes of the physical process. It will be more likely to choose cases that have not yet been chosen for this set of outcomes. That is, unless there is strong evidence that the system knows the best case to select for the current situation, it will experiment.
The formula used to calculate this probability is

$$(\alpha s_x^c)^{\beta \frac{h_x^c}{h_x+1}}$$

where β and α are parameters to control the effects of the terms, with $0 < \alpha < 1$, and

s_x^c is the strength of association between input x and case c, with $0 \leq s_x^c \leq 1$;

h_x^c is the number of times case c has been returned in response to case x; and

h_x is the number of times x has been input.

If two cases differ only in the similarity of the outcomes to the currently desired outcome, then the most similar case is more likely to be chosen. However, the exponent of the above formula ensures that the system will experiment by choosing those cases whose outcomes have not yet been compared to the currently desired outcomes.
The outcomes of the selected case are then presented to the user, along with the parameter values of the process that produced the outcomes. In the simulations described in this paper, the user is represented by a calculated function. The user compares the outcomes with the desired criteria, and returns a reinforcement value. The reinforcement is a real value within a known range (generally [0..1] or [-1..1]), and indicates how similar the outcomes of the selected case are to the desired outcomes.
Upon receiving the reinforcement from the user, the system adjusts its similarity metric according to the reinforcement. If the reinforcement is high, the similarity between the outcomes of the selected case and the currently desired outcomes is increased.
Two systems were implemented using this framework. They differ in how they implement the similarity function.

5 A linear weighting solution

In this solution, for each case in the library a vector of weights is kept with one component for each feature of the cases. The output of the similarity metric is the dot product of the weight vector and the input vector. For the selected case, the weight components are updated using the formula

$$w_i^{t+1} = w_i^t + \alpha r^{t+1} x_1^t \tag{1}$$

where

α is a parameter controlling learning increments, with $0 \leq \alpha \leq 1$,

r^t is the reward available at the start of time step t, and

x_i^r is a component of the input at time step t.

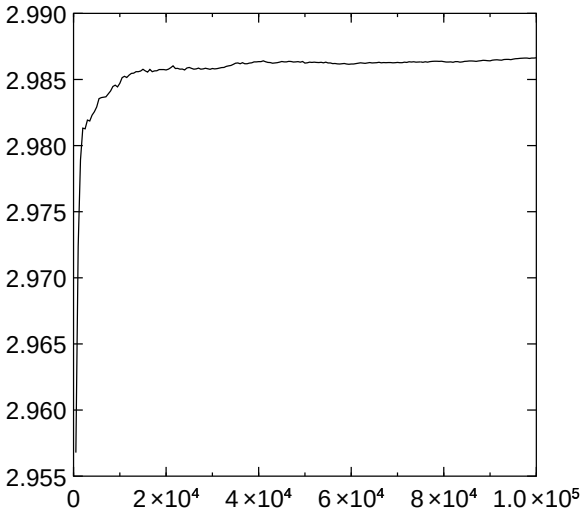

Figure 1: Reward per trial for linear weighting

The weight vector is then normalized by dividing through by the largest magnitude component so that all components are in the range [-1,1].
This formula 1 will cause the weights to be increased or decreased in proportion to the reward received an the component of the input vector associated with the reward. This means, for example, if the reward is large, and a particular component of the input was large, then the weight for that component is increased, making it more likely that this case will be selected in the future in response to this input.
While intuitively appealing, this approach has several drawbacks. Since the value α is positive, if we want a component to cause an increase in the output if we have a positive reinforcement, it will be necessary for the component of the input to be positive. Also, there is no way to cause the output of the algorithm to increase as one of the components is minimized.
Because of the limitations, this method must be modified in order to use it for parameter selection. In the experiments, the feature vectors representing the desired outcomes are pre-processed so that they include a component giving the value of the function that the user is trying to maximize. While not as desirable as a system that builds a similarity function from scratch, this method still may be valuable if a superset of possible evaluation functions is known in advance.
Figure 1 summarizes the results using this method. Each trial consists of using 3 cases as input. The reward is used is the Euclidean distance between the input and the case normalized to the range [0,1]. There are 10 cases in the library. The expected value of reward per trial is 1.53286. The maximum and optimal value is 3. The experiment shows that the linear weighting method is capable of selecting the correct feature to maximize.

6 A back-propagation solution

A more general approach is to use a multi-layer neural networks to implement the similarity function. Back-propagation is used to train the networks. The input layer of each network has a unit for each component of the input. There is a separate network for each case in the library. The output of each network is measure of the similarity of the current input and the library case associated with the network. This measure is then used to influence the probability of the case being chosen for output. After a case is output and a reinforcement signal is received, the neural network is updated by calculating a error value using

$$\epsilon = r - s$$

where

s is the strength returned for the selected case,

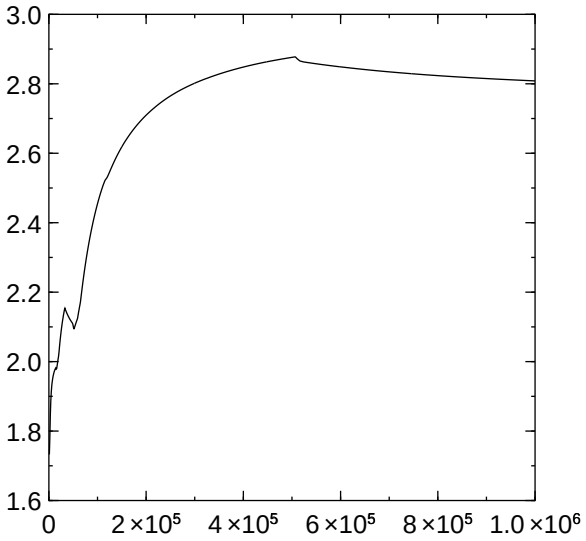

Figure 2: Reward per trial for back-propagation

r is the reinforcement received.

Thus, the network is gradually trained to return the reward expected when returning the associated case given the current requirements.
Figure 2 summarizes the results using this method. Each trial consists of using 3 cases as input. The reward is used is the Euclidean distance between the input and the case normalized to the range [0,1]. The components of the feature vectors are normalized to the range of [0,1]. The expected value of reward per trial is 1.53286 and the maximum and optimal value is 3. There are 10 cases in the library. The method reaches a value of about 2.875, but then begins to degrade. The network configuration used in this trial was a 8 unit input layer with a 16 unit middle layer and a one unit output layer.
A second run was done using a arbitrary function to determine similarity values. The reward used for each input/output pairing was randomly chosen in the interval [0,1]. Figure 3 summarizes the results using this method. Each trial consists of using 3 cases as input. As in the previous run, the reward is used is a randomly assigned value in the interval [0,1]. The components of the feature vectors are normalized to the range of [0,1]. The expected value of reward per trial is 1.70847 and the maximum and optimal value is 3. There are 10 cases in the library. The network configuration used in this trial was a 8 unit input layer with a 16 unit middle layer and a one unit output layer.

7 Conclusion

We have presented a method for combining reinforcement learning techniques with case-based reasoning. This allows case-based reasoning to be applied to applications for which there is no straightforward way to calculate the similarity between cases. This method was applied to the problem of determining which input parameters for a physical process will result in the outcomes closest to the desired outcomes. Two implementations of the model were presented. The first uses a linear weighting mechanism to construct a function for describing the similarity between process outcomes. This method was effective in determining which of several outcome features gave the best measure of similarity. The second method uses multi-layer neural nets trained with back-propagation. This method was effective in constructing similarity functions with no knowledge of the underlying application domain.

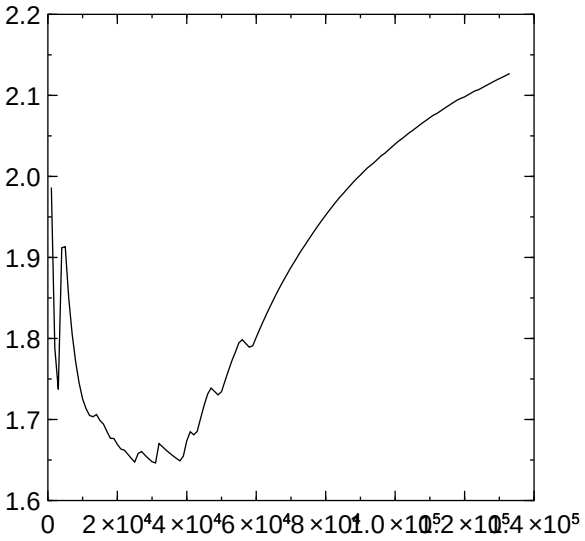

Figure 3: Reward per trial for back-propagation

References

[1] David W. Aha and Robert L. Goldstone. Learning attribute relevance in context in instance-based learning algorithms. In *Program of the Twelfth Annual Conference of the Cognitive Science Society*, pages 141–148. Lawrence Erlbaum Associates, Hillsdale, NJ, July 1990.

[2] James P. Callan, Tom E. Fawcett, and Edwina L. Rissland. Cabot: An adaptive approach to case-based search. In *Proceedings of the Twelfth International Conference on Artificial Intelligence*, pages 803–808, San Mateo, CA, 1991. International Joint Conference on Artificial Intelligence.

[3] Rich Caruana and Dayne Freitag. Greedy attribute selection. In *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, New Brunswick, NJ, 1994.

[4] L.H. Ihrig and S. Kambhampati. Storing and Indexing Plan Derivations through Explanation-based Analysis of Retrieval Failures. *Journal of Artificial Intelligence Research*, pages 161–198, 1997.

[5] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[6] Janet Kolodner. *Case-based reasoning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[7] Haun Liu and Rudy Setiono. Feature selection and classification - a probabilistic wrapper approach. In *Proceedings of the Ninth International Conference on Industrial and Engineering Applications of AI and ES*, 1996.

[8] Francesco Ricci and Paolo Avesan. Learning a local similarity metric for case-based reasoning. In *Proceedings of the First International Conference on Case-Based Reasoning*, pages 301–312. Springer-Verlag, Sesimbra, Portugal, 1995.

[9] M. Scherf and W Brauer. Feature selection by means of a feature weighting approach. Technical Report FK1-221-97, Institut für Informatik, Technische Universistät München, 1997.

[10] David B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh International Machine Learning Conference*, pages 293–301, New Brunswick, NJ, 1994. Morgan Kaufmann.

[11] Haleh Vafaie and Kenneth De Jong. Robust Feature Selection Algorithms. In *Proceedings of the 5th IEEE International Conference on Tools for Artificial Intelligence*, pages 356–363, 1993.

[12] D. W. Wettschereck, D. Aha and T. Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11:273–314, 1997.

[13] B. Widrow, N.K. Gupta, and S. Maitra. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 455–465, 1973.

[14] R. J. Williams. Reinforcement learning in connectionist networks: A mathematical analysis. technical report ics 8605. Technical report, Institute for Cognitive Science, University of California at San Diego, La Jolla, CA, 1986.