Feature Selection for CBR in Imitation of RoboCup Agents: A Comparative Study

Edgar Acosta, Babak Esfandiari, and Michael W. Floyd

Network Management and Artificial Intelligence Laboratory
Carleton University, Ottawa, Canada
http://www.nmai.ca/research-projects/agent-imitation

Abstract. We present a comparison of two methods for selecting the features employed in the representation of cases for a case-based reasoning approach to imitation of agents in the RoboCup simulation platform. Feature Selection methods are compared empirically in terms of the imitation performance, the size of the resulting feature set, and execution time. Differences and their implications are discussed.

1 Introduction

One central issue for every artificial intelligence problem is that of knowledge representation. Decisions regarding what to represent and how to do it often facilitate, and likewise constrain, the problem solving process and its success.

The most sensible way to deal with such sensitive decisions involves getting expert advice. However, when there is no expert knowledge available, feature selection methods can help to identify the pieces of information that are more relevant in solving the task [1].

Another aspect in which expert knowledge eases the solution of an artificial intelligence problem regards the processes underlying the task itself. When dealing with a new domain, or when the mechanisms leading to the problem's solution are not well known, case-based reasoning (CBR) provides a powerful learning methodology [2].

Imitation is one such task in which there is no formal model of the behaviour following a sequence of inputs. The RoboCup Simulation Imitation Agent Project (RCIAP)[3] employs both case-based reasoning and feature selection for learning to reproduce the behaviour of soccer playing agents based on the agent inputs.

We compare two feature selection methodologies in order to evaluate which methodology would better help to overcome some of the limitations of the current case recognition framework of the RCIAP.

2 Feature Selection

Feature (subset) selection is a machine learning sub-field that deals with selecting a set of relevant variables (features) from a large set of available features. The aim of feature selection is to reduce the number of features considered by the

learning task without deterioration to the learning performance. The employment of feature selection has a number of desirable advantages as a consequence of the reduction in dimensionality of the learning problem, such as faster learning and relatively smaller training sets.

Feature selection methods can be classified according to two parameters: the feature selection model (a.k.a. *feedback* [4]), and search strategies [1].

Three feature selection models can be identified. When feature selection is performed before the learning task in a pre-processing step, the method is called a **filter**. A **wrapper** model is one in which the search for a feature set is guided by a performance measure of the learned model. **Embedded** feature selection occurs when the construction of the set of features is integrated in the learning model (e.g. decision tree algorithms).

Different search strategies can be employed. The most common are **forward search** (starts with a small feature set and larger sets are explored), and **backward search** (starts with a large feature set and explores smaller ones). A simplification of these strategies consists of **sequential search**, in which one feature is included (or removed) at a time without backtracking. Other search methods include random and float search.

3 The RoboCup Simulation Imitation Agent Project

The purpose of the RCIAP is to build Robo Cup agents that imitate other (*target*) agents playing in the Robo Cup Simulation League, with as few domain-specific assumptions as possible.

3.1 The RoboCup Simulation League

The Robo Cup Simulation League is a platform for testing software agents playing on a soccer team. Each agent is a soccer player client that connects to a soccer simulation server. The simulation server keeps track of the game, sends messages to the clients and receives commands from them.

The server provides the agents with different kinds of information. There are two types of messages sent by the server: sensory information (the `see`, the `hear`, and the `sense body` messages), and game state information such as play modes (e.g. offside, goal kick, play on) and the match score. The `see` messages describe an incomplete view of the soccer field in terms of objects such as other players, the ball, field lines and flags. These objects also have properties such as distance and direction.

In response, agents send back one or more commands to the server, such as `dash`, `kick`, `turn`, or `say`. These commands also have properties such as power and angle of a kick.

The Case-Based Reasoning Approach: In RCIAP, imitation of agents is achieved using case-based reasoning. Observations are obtained from log files of the communication between an agent and the server. The captured data is transformed into a spatial representation (called a *scene*) that includes information

from the `see` messages (other types of inputs are ignored for now) together with the corresponding commands sent by the agent at each simulation cycle.

The obtained scenes constitute the knowledge of an imitative agent, which is employed in a case-based fashion when playing a soccer game. The agent transforms the current game situation into an scene representation, compares it to the stored scenes, finds the k-nearest scenes ($k = 1$), and then selects the command found on the *1*-nearest scene.

Challenges: Since a RoboCup simulation cycle takes 100ms, the imitative agent has to send a command back to the simulation server in less than 80ms (we impose a smaller than 100 ms limit to account for lag and message processing.) Thus the calculation of distance between the current simulation situation and stored cases has to be performed as fast as possible.

Distance calculation is affected by the number of features used to represent each case. The larger the number of features, the more complex the calculation procedure. Also, the number of cases required to discern between game situations increases factorially with the number of features employed. Therefore, it is convenient to consider as few features as possible.

Current Implementation: In the current RoboCup case-based agent [3], 160 visual input features and 3 output features were considered. The input features correspond to the relative coordinates of up to 80 visible objects (7 object types in bold face): the **Ball**, **flags** (up to 53 field landmarks,) players (up to 21, which are classified as **team mates**, **opponents**, and **not identifiable**,) the visible **goal** if any, and **field boundaries** (up to 4.) The output features correspond to the agent **command**, as well as the **power**, and aimed **direction** of the action.

Distance calculation (between the current input and a case input) is performed in the following way:

1. Seven lists of objects are generated, each ordered by nearest to farthest, and from left to right.
2. The objects from each list are matched in order. Non-matching objects are ignored. If only n objects can be matched, then let i be the set of matched objects from the current input, and c_m the corresponding set of matches in case m. i.e.

$$\begin{aligned} i &= \{\mathrm{obj}_1, \mathrm{obj}_2, \cdots, \mathrm{obj}_n\} \\ c_m &= \{match(\mathrm{obj}_1), match(\mathrm{obj}_2), \cdots, match(\mathrm{obj}_n)\} \end{aligned}$$

3. The euclidean distance between matched objects is obtained: $d\left(\mathrm{obj}_k, match(\mathrm{obj}_k)\right)$
4. All these distances are weighted and added up:

$$d(i, c_m) = \sum_{j=1}^{n} w_j \times d\left(\mathrm{obj}_j, match(\mathrm{obj}_j)\right)$$

Weights are determined offline in a pre-processing step. The weight calculation method used in the current implementation is a backward sequential search

binary wrapper algorithm [5, page 56]. Weights are 0 or 1 depending on the object type. The idea is to assign weight 1 only to types of objects that are highly relevant, and ignore object types with low relevance.

This approach is a feature selection algorithm that considers only 7 features, one per object type. It starts with the 7 object types (all weights are 1) and evaluates the CBR performance; this is the current best set of features. Starting with the current best features set, the algorithm considers all the subsets of features that obtain by removing only one feature (switching its weight to 0), and evaluates the CBR performance of each of these sets. If one of those subsets has the best CBR performance, and is at least 0.5% better than the current best, then it becomes the new current best set of features and the process is started again. Otherwise the algorithm returns the current best features set.

Limitations: One important limitation of this approach is that by bundling together all the objects of the same type into a single feature we are imposing an overgeneralizing and strong constraint to distance weights, which also affects the inductive bias of the k-nearest case search. By doing so, we end up using many features that may not be very relevant, and ignoring others that may be relevant.

For example, it may well be the case that the most relevant object is the player that is closer to the ball, and that the output command depends on whether that player is a team mate or an opponent.

If we performed feature selection over the original 160 features we would have better chances of capturing such distinctions. However, this feature selection algorithm is extremely slow, so it cannot scale to many more than 7 features. Time is of the essence because one of our future goals involves being able to build the case base and compute weights on line.

Improving Feature Selection: It would be useful to find a feature selection algorithm that is faster and can be scaled to at least 160 input features. We must, however, be systematic and attack each of these desiderata at a time. First we need to find a feature selection method that is faster and obtains similar CBR performance when we apply it to the same 7 features. Once we achieve that goal, we can try to use it with larger feature sets.

Zhong, et al. [6] report a rough set based filter approach for feature selection that is fast and effective in large databases. In the next section we propose a similar algorithm. Our hypothesis is that this algorithm will achieve the first goal, namely to perform feature selection faster without sacrificing CBR performance. Then, in Section 5 we test our hypothesis by comparing both algorithms in terms of CBR performance and feature selection performance.

Salamo and Golobardes [7] discuss two other (continuous) weighting methods for CBR based on rough sets. Other applications of rough set theory for CBR include case base reduction [8] and case representation [9].

4 Rough Set Theory Approach to Feature Selection

Rough set theory (RST) [10] deals with the representational properties of sets of finite discrete valued features.

In RST, a *decision system* is a tuple (U, C, D) where U is the universe of considered objects, and C and D are two disjoint collections of features. Both C and D induce equivalence classes in U, let us call their sets of equivalence classes $\mathcal{C}$ and $\mathcal{D}$ respectively. The idea in a decision system is to approximate classes in $\mathcal{D}$ in terms of the classes in $\mathcal{C}$. In other words, we try to characterize the different classes in $\mathcal{D}$ using only the feature set C.

This is relevant to machine learning if we think of C as the features used to describe observations, and of D as the features of those objects that we want to learn. Hence, we call them *condition features* and *decision features* respectively.

The *positive region* $POS_C(\mathcal{D})$ of $\mathcal{D}$ in terms of C is the set of objects in U that can be correctly classified as belonging to one class in $\mathcal{D}$ using only the information about features C.

4.1 Rough Set Theory methods for Feature Selection

Given a decision system $S = (U, C, D)$, the feature subset selection problem consists of finding a minimal family of features $C' \subseteq C$ such that $POS_{C'}(\mathcal{D}) = POS_C(\mathcal{D})$.

A solution C' to the feature subset selection problem is a D-reduct of C.
If f is a feature in C such that $POS_{C-f}(\mathcal{D}) \subsetneq POS_C(\mathcal{D})$, then f is D-indispensable in C.
The union of all D-indispensable features in C is the D-core of C, $CORE_D(C)$.

Some interesting properties of reducts are worth noticing:

multiplicity of solutions: There may be more than one D-reduct of C, i.e. the feature selection problem may have more than one solution.
core and reducts: Every D-reduct of C contains the $CORE_D(C)$.

These observations suggest a basic method for feature selection: we can start by finding the core, and then add features to the core until we find a subset of features with a positive region that matches that of the full set of condition features (see Algorithm 3). Finding the core is as simple as testing every feature in C for the indispensability property (see Algorithm 2). Each of these procedures depend on calculating the positive region of $\mathcal{D}$ (as the feature set evaluation metric) in terms of different subsets of condition features (see Algorithm 1).

Algorithm 3 returns the first solution it finds, thus, the order (`rank()` procedure) in which dispensable features are added to the core is one source of inductive bias. The other source of inductive bias comes from the feature set evaluation measure $|POS_{G \subseteq C}(\mathcal{D})|$ (`PR()` procedure.)

Algorithm 3 does not guarantee finding a D-reduct of C. Nevertheless, it finds a feature set A such that A contains a D-reduct of C, and $A \subseteq C$,i.e. no

Algorithm 1 Positive Region

Inputs: observations, conditionFeatures, decisionFeatures
Outputs: positiveRegion: number of discernible observations

```
PR(observations, conditionFeatures, decisionFeatures)
    conditionClasses = { }
    positiveRegion = 0
    for(each observation Obs in observations)
        cClass = determineConditionClass(Obs[Input],conditionFeatures)
        dClass = determineDecisionClass(Obs[Output],decisionFeatures)
        if(cClass does not exist in conditionClasses)
            elementCounter = 1
            outputClasses = set{dClass}
        else
            elementCounter = conditionClasses[cClass][0] + 1
            outputClasses = conditionClasses[cClass][1]
            outputClasses.set_add(dClass)
        conditionClasses[cClass]= {elementCounter,outputClasses}
    for(each condition class condClass in conditionClasses)
        elements = conditionClasses[condClass][0]
        outputs = conditionClasses[condClass][1]
        if(number of different outputs == 1)
            positiveRegion += elements
    return positiveRegion
```

information is lost by using this algorithm. In most cases Algorithm 3 will return a smaller set than C. Running the basic algorithm again on the first run solution may help to further reduce the set of condition features.

If n is the number of training cases, $c = |C|$, and m is the maximum number of values that a feature in C can take, then Algorithm 3 is of order $O(nmc^2)$.

4.2 Advantages of Rough Set Theory Methods

One consequence of RST is that the cardinality of the positive region $|POS_C(\mathcal{D})|$ is the maximum number of objects in U that can be correctly categorized with the information provided by any subset of the set of condition features. Thus, the positive region is an upper bound for the learning performance of any machine learning approach.

4.3 Proposed Feature Selection Method

Despite the fact that Algorithm 3 does not always find a reduct, we propose to compare it with the current weight calculation employed in RCIAP.
Hypothesis: Algorithm 3 implies a filter feature selection model, which has the potential for a faster execution time than the current wrapper approach. Thus, we expect an important reduction in execution time.

Algorithm 2 Core

Inputs: observations, conditionFeatures, decisionFeatures, maxPR
Outputs: coreFeatures

```
CORE(observations, conditionFeatures, decisionFeatures, maxPR)
    coreFeatures = { }
    for(each feature Feature in conditionFeatures)
        complement = conditionFeatures
        complement.remove(Feature)
        complementPR=PR(observations, complement, decisionFeatures)
        if(complementPR < maxPR)
            coreFeatures.add(feature)
    return coreFeatures
```

As for CBR performance, we do not expect statistically significant differences, or important differences otherwise. The same hypothesis holds for the number of selected features.

5 Feature Selection Methods Comparison

This section describes the comparisons performed between the current backward sequential search binary wrapper algorithm and a rough set based forward sequential search filter algorithm (Algorithm 3 with no ranking of dispensable features).

5.1 Materials

Target Agents: Data from 3 RoboCup agents was used. In order of complexity: *Krislet* a simple team that chases the ball and kicks it toward the opponents goal, *A1* based on Krislet (this agent will not follow the ball if it sees a team mate closer to it,) and *CMUnited 2000* RoboCup World Champions developed by Carnegie Mellon University.
Case Base: The case base consisted of $15,000$ cases per agent. Each case encoded the 160 input features, as well as the 3 output features described in Section 3. For feature selection, only 7 seven features (the object types) were used.
Independent variables: The experiment consisted of 6 experimental conditions with 2 independent variables: target agent (3 conditions), and feature selection algorithm (2 conditions).
Performance metrics:

Feature set size: The number of features in the solution set.
Execution time: This is the execution time of the feature selection procedure.
Accuracy: The rate of CBR outputs that matched the output in the testing case. This only takes into account the type of action.

Algorithm 3 Forward Sequential Feature Selection

Inputs: observations, conditionFs, decisionFs

Outputs: reduct

```
RSFS(observations, conditionFs, decisionFs)
    maxPR = PR(observations, conditionFs, decisionFs)
    coreFeatures = CORE(observations,conditionFs,decisionFs,maxPR)
    dispensableFeatures = conditionFs - coreFeatures
    despensableRanked = rank(dispensableFeatures)
    currentBest = coreFeatures
    bestPR = PR(observations, currentBest, decisionFs)
    for(each feature Disp in dispensableRanked)
        testSet = currentBest
        testSet.add(Disp)
        testPR = PR(observations, testSet, decisionFs)
        if(testPR > bestPR)
            currentBest = testSet
            bestPR = testPR
        if(bestPR == maxPR)
            return currentBest
    return currentBest
```

Recall per action type: For each action type, this is the rate of testing cases for which CBR selected the correct action.

Precision per action type: For each action type, this is the rate of CBR outputs that correctly selected this action type.

f-measure per action type: The f-measure combine recall and precision into a single metric

$$f_{\text{action}} = \frac{2 \times Recall_{\text{action}} \times Precision_{\text{action}}}{Recall_{\text{action}} + Precision_{\text{action}}}$$

Global f: The global f-measure is the average of the collected f-measures.

The first two metrics regard to the feature selection performance, and the remaining metrics regard to CBR performance with the resulting set of features. The experiment, and CBR, were performed with the tools described in [5, section 9.5.].

5.2 Methodology

30 trials per condition were performed. On each trial, $1,500$ random cases were used during feature selection, and $6,000$ random cases were used during CBR performance evaluation.

Statistical analysis: In order to determine significance of the observed differences between the two feature selection approaches, we performed a 1-factor repeated measures analysis of variance per metric collected for each agent condition.

5.3 Results

The next two tables show the averaged metrics for CBR performance and Feature Selection performance, as well as the result of the ANOVA performed for the effect of the feature selection algorithm on the collected metrics.

CBR Performance:

	Krislet		**A1**		**CMU**	
Algorithm	Global f	Accuracy	Global f	Accuracy	Global f	Accuracy
BS wrapper	0.84	0.98	0.81	0.89	0.70	0.74
RS filter	0.66	0.86	0.77	0.88	0.64	0.73
$F(1, 29)$	$F = 88.77$ $p < 0.01$ $\eta^2 = 0.75$	$F = 77.67$ $p < 0.01$ $\eta^2 = 0.73$	$F = 39.00$ $p < 0.01$ $\eta^2 = 0.57$	$F = 5.36$ $p < 0.05$ $\eta^2 = 0.16$	$F = 62.67$ $p < 0.01$ $\eta^2 = 0.68$	$F = 0.56$ $p = 0.46$ $\eta^2 = 0.02$

Feature Selection Performance: Execution times are in minutes.

	Krislet		**A1**		**CMU**	
Algorithm	time	set size	time	set size	time	set size
BS wrapper	66.29	1.13	77.59	2.90	56.57	3.93
RS filter	1.89	1.90	2.70	2.57	1.61	3.43
$F(1, 29)$	$F = 1030$ $p < 0.01$ $\eta^2 = 0.97$	$F = 69.41$ $p < 0.01$ $\eta^2 = 0.71$	$F = 759$ $p < 0.01$ $\eta^2 = 0.96$	$F = 4.26$ $p < 0.05$ $\eta^2 = 0.13$	$F = 572$ $p < 0.01$ $\eta^2 = 0.95$	$F = 4.58$ $p < 0.05$ $\eta^2 = 0.14$

6 Discussion

All the metrics, except CBR accuracy for the CMU agent, had statistically significant differences at the indicated significance levels (p value). As it was to expect, CBR performance degraded with an increase of complexity of the target agents, regardless of the feature selection method.

Although the current Backward Sequential Search binary wrapper algorithm produces superior CBR performance than the proposed Rough Set based Forward Sequential Search filter algorithm, the gap in CBR performance diminishes as the target agent complexity increases. The difference in CBR performance is likely due to the feature selection model: the wrapper approach finds a solution that best works with the particular learning algorithm.

The most impressive difference between the two feature selection approaches is observed in the execution time of feature selection, being the proposed algorithm almost 30 times faster than the current one. This difference is due to the cost of the feature set evaluation employed; whereas the current approach evaluates by observing the CBR performance that the feature set yields, the proposed algorithm computes the positive region of the feature set. Notice that any other wrapper approach (including most evolutionary approaches) would suffer from the same problem.

7 Conclusion

Although the proposed algorithm produced slightly, but significant, worse CBR performance than the current algorithm, it impressively improved the feature selection time. This makes the proposed algorithm a good candidate for performing feature selection over more than the current 7 features.

Further work should focus on trying to use this algorithm with more features, ideally the 160 employed in case distance calculation. Such improvement is likely to lead to better imitation performance. Also, the proposed algorithm can be improved both in speed, and in better approximating a D-reduct of the condition features.

References

1. Liu, H., Motoda, H.: Less is more. In: Computational Methods of Feature Selection. Chapman & Hall/CRC, Boca Raton (2008)
2. Pal, S.K., Shiu, S.C.K.: Introduction. In: Foundations of soft case-based reasoning. Wiley-Interscience, Hoboken, N.J. (2004)
3. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating RoboCup players. In: Proceedings of the Twenty-first International Florida Artificial Intelligence Research Society Conference, AAAI Press (2008) 251–256
4. Wettschereck, D., Aha, D.: Weighting features. In: Proceedings of the First International Conference on Case-Based Reasoning, Springer (1995) 347–358
5. Floyd, M.W.: Improving the performance of a RoboCup case-based imitation agent through preprocessing of the case base. Master's thesis, Carleton University, Ottawa, Ontario, Canada (December 2008)
6. Zhong, N., Dong, J.Z., Ohsuga, S.: Using rough sets with heuristics for feature selection. Journal of Intelligent Information Systems **16**(3) (August 2001)
7. Salamo, M., Golobardes, E.: Analysing rough sets weighting methods for case-based reasoning systems. Inteligencia Artificial. Revista Iberoamericana de IA (15) (2002) 10–18
8. Salamó, M., Golobardes, E.: Rough sets reduction techniques for case-based reasoning. In: Case-Based Reasoning Research and Development. Springer-Verlag (2001) 467–482
9. Wierzbicki, J.: Rough set approach to CBR. In Ziarko, W., Yao, Y., eds.: Rough Sets and Current Trends in Computing. Volume 2005 of Lecture Notes in Artificial Intelligence., Berlin, Germany, Springer-Verlag (2001) 503–510
10. Pawlak, Z.: Rough sets : Theoretical aspects of reasoning about data. Volume 9. Kluwer Academic Publishers, Dordrecht; Boston (1991)