Monotonicity in Ant Colony Classification Algorithms

James Brookhouse and Fernando E. B. Otero

School of Computing, University of Kent
Chatham Maritime, UK
{jb765,F.E.B.Otero}@kent.ac.uk

Abstract. Classification algorithms generally do not use existing domain knowledge during model construction. The creation of models that conflict with existing knowledge can reduce model acceptance, as users have to trust the models they use. Domain knowledge can be integrated into algorithms using semantic constraints to guide model construction. This paper proposes an extension to an existing ACO-based classification rule learner to create lists of monotonic classification rules. The proposed algorithm was compared to a majority classifier and the Ordinal Learning Model (OLM) monotonic learner. Our results show that the proposed algorithm successfully outperformed OLM's predictive accuracy while still producing monotonic models.

Keywords: ant colony optimization, semantic constraints, monotonic, data mining, classification rules, sequential covering

1 Introduction

Data mining is a research area focused on automating the search for useful patterns in data [6], where classification is one of the most studied tasks. A classification problem involves a set of examples, where each example is described by predictor attribute values and associated with a target class value. The goal of a classification algorithm is to find the best model that accurately represents the relationships between predictor and class attribute values, and therefore classification problems can be viewed as optimisation problems. Many algorithms concentrate on producing accurate models, however while accuracy is an important feature other properties of a model are also important, including a model's comprehensibility and its ability to preserve pre-existing domain knowledge. Both of these features can aid the acceptance of a model amongst its users who are normally experts in the domain being investigated.

Existing domain knowledge can be captured by semantic constraints, which can then be used to guide the construction of models. Algorithms that build models able to break these relationships can lead to increased model rejection by domain experts due to their counter intuitiveness [9]. One form of semantic constraints are monotonic constraints, which concern the sign of a relation between the predictor and target attributes. A problem that illustrates monotonic

properties is house rental prices as it is expected that as the size of a house increases so will its rental price—i.e., there is an increasing monotonic relation between size and rental price within the same location.

We investigate a strategy to add domain knowledge to the learning process of an Ant Colony Optimization (ACO) [4] classification rule learner to produce models that are both accurate and enforce monotonic constraints. We evaluate the impact of the proposed strategy on predictive accuracy and compare the results against an existing monotonic learner using publicly available data sets.

The rest of the paper is structured as follows. Firstly, Section 2 presents work from the literature that has been completed in related areas. Section 3 discusses the changes to *c*Ant-Miner$_{\text{PB}}$ allowing the enforcement of constraints. Next, Section 4 presents our computational results including a comparison with another monotonic learner, followed up by our conclusions and suggestions on possible future directions in Section 5.

2 Background

There are two main areas of related work, ACO-based classification rule learners and semantic constraints. First we will discuss existing sequential covering classification rule learners including the ACO-based Ant-Miner and its extensions. This will be followed by a summary of the literature surround semantic constraints, including monotonic constraints—the focus of this paper. Finally, we will discuss AntMiner+ in more detail, since it is an ACO-based classification rule learner that incorporates monotonic constraints.

2.1 Ant Colony Classification Algorithms

One of the most popular strategies for creating a list of classification rules from a given dataset is called *sequential covering* (or *separate-and-conquer*) [8]. It consists in transforming the problem of creating a list of rules into smaller problems of creating a single rule: (1) a single rule is created from the data (*conquer* step); (2) data instances that are covered by the rule are removed from the training data (*separate* step); these steps are repeated until the training data is empty or the number of uncovered data instances falls below a pre-defined threshold. Many rule induction algorithms follow the same separate-and-conquer strategy, their main difference is the way that individual rules are learn. In this context, ACO has been successfully applied to create classification rules.

The first ACO classification algorithm, called Ant-Miner, was proposed in [15]. Ant-Miner follows a sequential covering strategy, where individual rules are created by an ACO procedure. The main idea is to search for the best classification rule given the current training data at each iteration of the sequential covering. Ants traverse a construction graph selecting terms to create a rule in the form `IF` $\texttt{term}_1$ `AND ... AND` $\texttt{term}_n$ `THEN class`, where the `IF`-part represents the antecedent and the `THEN`-part is the class prediction. Each ant starts with an empty rule and iteratively selects terms to add to its partial rule based

on their values of the amount of pheromone τ and a problem-dependent heuristic information η, similarly to Ant System (AS) [3]. Following on Ant-Miner's success, many extensions have been proposed in the literature [13]: they involve different rule pruning and pheromone update procedures, new rule quality measures and heuristic information. There are two Ant-Miner extensions relevant to this work: AntMiner+ [12] and cAnt-Miner$_{\mathrm{PB}}$ [14].

AntMiner+ extends Ant-Miner in several aspects: (1) the complexity of the construction graph is reduced, in term of the number of edges connecting vertices, by defining it as a direct acyclic graph (DAG); (2) it makes a distinction between nominal attributes with categorical and ordered values, where ordinal attributes are used to create interval conditions; (3) the class value to be predicted and weight parameters used to control the influence of the pheromone and heuristic information are incorporated in the construction graph as vertices.

One potential drawback of using a sequential covering to create a list of rules is that there is no guarantee that the best list of rules is created. Ant-Miner (and the majority of its extensions) perform a greedy search for the list of best rules, using an ACO procedure to search for the best rule given a set of examples, and it is highly dependant on the order that rules are created. Therefore, they are limited to creating the *list of best rules*, which does not necessarily corresponds to the *best list of rules*. cAnt-Miner$_{\mathrm{PB}}$ is an ACO classification algorithm that employs an improved sequential covering strategy to search for the best list of classification rules [14]. While Ant-Miner uses an ACO procedure to create individual rules in a one-at-a-time (sequential covering) fashion, cAnt-Miner$_{\mathrm{PB}}$ employs an ACO procedure to create a complete list of rules. Therefore, it can search and optimise the quality of a complete list of rules instead of individual rules—i.e., it is not concerned by the quality of the individual rules as long as the quality of the entire list of rules is improving.

2.2 Semantic Constraints

When existing domain knowledge is available, semantic constraints can incorporate this knowledge into the construction of new models. For example, if you consider house rent the price can/will depend on the location and floor area. Table 1 shows a simple hypothetical rental dataset. One relationship in this data set is that houses in better locations (lower values of attribute *Location*) increases its rental price. This is the case for all possible pairs in the data set.

Model rejection by domain experts is a possibility if a model does not preserve existing patterns as it would seem counter intuitive. Hoover and Perez [9] state that the economic field scepticism towards data mining as a technique to search for models is due to the discovery of accidental correlations: *"Data mining is considered reprehensible largely because the world is full of accidental correlations, so that what a search turns up is thought to be more a reflection of what we want to find than what is true about the world."* [9, p. 197]. Semantic constraints allow model construction to be guided by providing information on real correlations present within the data. While there are a number of different semantic

Table 1. Simple house rental data set.

Target Attribute	Predictor Attributes	
Rental Value	Floor Area	Location
Medium	45	2
High	80	1
Low	33	3
Medium	65	2

constraint types, we explore the implementation of monotonic constraints in the discovery of classification rules.

2.3 Monotonicity

Monotonicity is found in many different fields including house/rental prices, medicine, finance and law. Looking at the first example of rental prices, it can be expected that as the location of a property becomes better (lower value of *Location*) its rental value will also increase—this can be seen in the example data shown in Table 1. The majority of data mining algorithm are not monotonically aware and do not enforce this relationship during model construction, yet still produce good models. However, if models violate these constraints they may not be accepted by experts as valid, and therefore, conforming to monotonicity constraints may help improve model acceptance [5,7].

Monotonicity can be defined formally in the following manner. Let $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_i$ be the instance space of i attributes, $\mathcal{Y}$ be the target space, and function $f : \mathcal{X} \to \mathcal{Y}$. It is also assumed that both the instance space and target space have an ordering. A function can then be considered monotone if:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} : \mathbf{x} \leq \mathbf{x}' \implies f(\mathbf{x}) \leq f(\mathbf{x}') \ , \tag{1}$$

where $\mathbf{x}$ and $\mathbf{x}'$ are two vectors in instance space, $\mathbf{x} = (x_1, x_2, \cdots, x_i)$ [16]. In other words, $f(\mathbf{x})$ is monotonic if and only if all the pairs of examples $\mathbf{x}$, $\mathbf{x}'$ are monotonic with respect to each other.

Monotonicity can be enforced in a number of different stages in the data mining process. The first is in the *pre-processing stage*, training data is manipulated so that the required attributes are monotonic with respect to the dependent attribute. Since this method is unable to enforce constraints in the model, it will be discussed no further. In the *model construction stage* the model construction algorithm creates monotonic models, possibly constraining the search. Finally, constraints can be enforced in a *post-processing stage* via the modification of constructed models to enforce monotonic constraints.

Constraints also appear in two different forms hard or soft. Hard constraints are enforced rigidly and will reject any new model or change to an existing model that would cause a violation. Good models can be rejected due to small

violations in their monotonicity when this method is used. The second method, soft constraints, balances the monotonicity of a model against the models quality, allowing small violations to exist if they sufficiently increase the quality.

Model Construction Soft constraints have been implemented in the model construction stage by Ben-David [1]. Ben-David assigns a non-monotonicity index to each tree produced. This index is the ratio between the number of non-monotonic leaf node pairs and the maximum number of pairs that could have been non-monotonic. First a non-monotonicity matrix m is constructed which has dimensions k (the number of branches in the tree). This matrix is used to find the number of violations in the current tree, given by:

$$W = \sum_{i=1}^{k} \sum_{j=1}^{k} m_{ij} \ , \text{ where } m_{ij} = \begin{cases} 1 & \text{, if } ij \text{ is non-monotonic} \\ 0 & \text{, otherwise} \end{cases} , \tag{2}$$

where i and j denote the current cell being referenced in the matrix. W can then be used to find a tree's non-monotonicity index, given by:

$$I_{a_1 \ldots a_v} = \frac{W_{a_1 \ldots a_v}}{k^2_{a_1 \ldots a_v} - k_{a_1 \ldots a_v}} \ , \tag{3}$$

where $a_1 ... a_v$ are the constrained attributes with v being the total number of constrained attributes, the $I_{a_1 \ldots a_v}$ index can be converted to an ambiguity score A and then incorporated with a tree accuracy score T, given by:

$$A_{a_1 \ldots a_v} = \begin{cases} 0 & \text{, if } I_{a_1 .. a_v} = 0 \\ -(log_2(I_{a_1 \ldots a_v}))^{-1} & \text{, otherwise} \end{cases} , \tag{4}$$

$$T_{a_1 \ldots a_v} = E_{a_1 \ldots a_v} + RA_{a_1 \ldots a_v} \ , \tag{5}$$

where R is the importance given to the monotonicity of trees produced and E is an error based measure, usually the accuracy. The accuracy of the models produced were not significantly degraded compared to the original algorithm, however the combined metric did produce fewer models that breached the monotonicity constraints [1].

Ben-David has also investigated monotonic ordinal classifiers, proposing the hypothesis that adding monotonicity constraints to learning algorithms will impair their accuracy against those that do not. Ordinal classifiers are classifiers that allow discrete categories to have an order, for example credit rating has an obvious order if the categories are poor, acceptable and good. There were two unexpected results. It was found that ordinal classifiers did not significantly improve their accuracy over non-ordinal classifiers. Secondly, the monotonic algorithms were not able to significantly outperform a simple majority-based classifier. It is theorised that these results were due to noisy data sets: the monotonic classifiers enforced hard constraints, in the presence of noisy data a softer approach may lead to better results [2]. The algorithms also enforce constraints on

all attributes, which may be unrealistic for real data sets as monotonically noisy attributes or those with no trend may masque the true monotonic attributes.

Qian et al. [17] have explored the possibility of fusing monotonic decision trees to improve the accuracy of the final model. This is achieved by reducing the original data set to create sets that maintain the monotonicity of the original. From these new reduced data sets, monotonic trees can be constructed. Each leaf node of a reduced tree then contain probabilities of the correctness of the prediction based on the training set. When a prediction is required, the probabilities at each tree's leaf nodes is averaged with the highest average being the class predicted by the model.

Post-Processing Feelders [7] has suggested that using non-monotonic criteria in tree construction is not beneficial as splits later in the construction process can transform a tree from a state of non-monotonicity to one that is. Therefore, Feelders has suggested several pruning methods to make the minimal number of changes to make a tree monotonic in a post-processing phase [7].

The first proposed pruner is the Most Non-monotone Parent (MNP) method, which aims to prune the node that gives the most number of monotone pairs. This method has the disadvantage of possibly creating more non-monotonic pairs than it removes leading to a net increase in non-monotonicity. The second method proposed is the best fix method, which prunes the node that gives the biggest reduction in non-monotonicity. While it solves the problem with the first pruner, it is more computationally expensive. The authors have also combined these pruning methods with existing complexity pruning methods and found that the monotonic trees produced no significant difference in performance compared to trees produced without monotonic pruning. However, it was observed that the trees produced were smaller, which aids the comprehensibility of the models produced further [7].

2.4 AntMiner+ with monotonicity constraints

As far as we are aware, the only implementation of an ACO that discovers monotonic classification rules was proposed by Martens et al. [11], who modified AntMiner+ to enforce hard and soft monotonic constraints. The basic idea is to limit the solution space by either removing nodes in the construction graph or manipulating the heuristic values of vertices. In the first approach, authors modified the construction graph by removing nodes, and subsequently closing off those areas of the search space, that could be used to create non-monotonic rules (hard constraints). This algorithm was applied to the binary classification problem of classifying customers as good or bad credit risks, where the algorithm could only create rules that predicted bad customers. For example, if there is a monotonic constraint on income, nodes corresponding to $income > x$ are removed leaving only those that express $income < x$, which will always produce monotonic rules when discovering the (negative) bad credit class. In the second approach, the heuristic value of a node that is monotonically related to the

predicted class is adjusted to incorporate this preference, although they did not include experiments verifying how effective this would be.

It was found that AntMiner+ with hard constraints consistently produced rule lists that contained less rules and less terms per rule, when compared to the original algorithm without impacting the accuracy of the model produced. The comprehensibility of the models produced would be increased by the reduced model size [11]. While their results were positive overall, their approach seem to be limited to binary classification problems: the algorithm creates rules for the minority (bad credit) class, while a default rule predicts the majority (good credit) class; removal of conditions is based on a particular class value to be predicted and it is not clear how the removal of nodes can be used to enforce constraints in multi-class problems. Additionally, it has the side effect of limiting the search space of solutions, not taking into account that monotonicity is a global property [22] and a partial non-monotone rule might become monotone after additional conditions.

3 Discovering Monotonic Classification Rules

In this section we will provide an overview of *c*Ant-Miner$_{\mathrm{PB}}$ and the modifications to the pruning strategies present in the proposed *c*Ant-Miner$_{\mathrm{PB+MC}}$ (Pittsburgh-based *c*Ant-Miner with monotonicity constraints).

3.1 *c*AntMiner$_{\mathrm{PB}}$ with monotonicity constraints

As we discussed in Section 2.1, *c*Ant-Miner$_{\mathrm{PB}}$ is an ACO classification algorithm that employs an improved sequential covering strategy to search for the best list of classification rules. In summary, *c*Ant-Miner$_{\mathrm{PB}}$ works as follows (Figure 1). Each ant starts with an empty list of rules and iteratively adds a new rule to this list (*for loop*). In order to create a rule, an ant adds one term at a time to the rule antecedent by choosing terms to be added to the current partial rule based on the amount of pheromone (τ) and a problem-dependent heuristic information (η). Once a rule is created, it undergoes a pruning procedure. Pruning aims at removing irrelevant terms that might be added to a rule due to the stochastic nature of the construction process: it starts by removing the last term that was added to the rule and the removal process is repeated until the rule quality decreases when the last term is removed or the rule has only one term left. Finally, the rule it is added to current list of rules and the training examples covered by the rule are removed.[1] An ant creates rules until the number of uncovered examples is below a pre-defined threshold (inner *while loop*).

At the end of an iteration, when all ants have created a list of rules, the best list of rules (determined by an error-based list quality function) is used to update pheromone values, providing a positive feedback on the terms present

[1] An example is covered by a rule when it satisfies all terms (attribute-value conditions) in the antecedent of the rule.

```
Input: training instances
Output: best discovered list of rules
 1. InitialisePheromones();
 2. list_gb ← {};
 3. t ← 0;
 4. while t < maximum iterations and not stagnation do
 5.     list_ib ← {};
 6.     for n ← 1 to colony_size do
 7.         instances ← all training instances;
 8.         list_n ← {};
 9.         while |instances| > maximum uncovered do
10.             ComputeHeuristicInformation(instances);
11.             rule ← CreateRule(instances);
12.             SoftPruner(rule, list_n);
13.             examples ← instances − Covered(rule, instances);
14.             list_n ← list_n + rule;
15.         end while
16.         if Quality(list_n) > Quality(list_ib) then
17.             list_ib ← list_n;
18.         end if
19.     end for
20.     UpdatePheromones(list_ib);
21.     if Quality(list_ib) > Quality(list_gb) then
22.         list_gb ← list_ib;
23.     end if
24.     t ← t + 1;
25. end while
26. HardPruner(list_gb);
27. return list_gb;
```

Fig. 1. High-level pseudocode of the *c*Ant-Miner$_{\mathrm{PB+MC}}$ algorithm. The main differences compared to *c*Ant-Miner$_{\mathrm{PB}}$ [14] are found on lines 12, 16 and 26.

in the rules—the higher the pheromone value of a term, the more likely it will be chosen to create a rule. This iterative process is repeated until a maximum number of iterations is reached or until the search stagnates (outer *while loop*).

One of the main differences in *c*Ant-Miner$_{\mathrm{PB}}$, when compared to other ACO classification algorithms, is that an ant creates a list of rules. Therefore, the ACO search is guided by and optimises the quality of a complete solution. Additionally, there is also the possibility of applying local search operators to the complete solution—e.g., a pruning procedure is an example of a local search operator. This is currently not explored in *c*Ant-Miner$_{\mathrm{PB}}$, since pruning is applied to individual rules and not to the entire list of rules.

*c*Ant-Miner$_{\mathrm{PB+MC}}$ is modified in three key places compared to the original *c*Ant-Miner$_{\mathrm{PB}}$. The first change is a modification to the pruning method (line 12 of Figure 1): this pruner is a soft pruner that balances monotonicity against accuracy. This modified quality is then used to update the pheromone levels

ready for the next iteration. The second modification is the addition of a hard prune that rigidly enforces the monotonic constraints, this occurs immediately before the rule list is returned (line 26). Both pruners are explained in more detailing in the following section. The final modification is to the list quality function (line 16), this quality now uses both accuracy and NMI combined with a weighting term when assigning a quality measure to the list and comparing it to the best so far. This is the same function that is used in the soft pruner and shown by equations 6 and 7.

3.2 Rule Pruning

There are two pruners used by cAnt-Miner$_{\mathrm{PB+MC}}$: soft pruner that may allow constraint violations and a hard pruning that guarantees constraints are satisfied. In ACO terms, a pruner is a local search operator.

Soft pruning A soft monotonic prune allows violations in the monotonic constraint if the consequent improvement in accuracy is large enough. The pruner operates on an individual rule and iteratively removes the last term until no improvement in the rule quality is observed. Applying a soft pruner during model creation allows the search to be guided towards monotonic models while still allowing exploration of the search space.

As monotonicity is a global property of the model, the rule being pruned is temporarily added to the current list of rules, its non-monotonicity index (NMI) can then be used as a metric to assess the rules monotonicity and is given by:

$$NMI = \frac{\sum_{i=1}^{k} \sum_{j=1}^{k} m_{ij}}{k^2 - k} \ , \tag{6}$$

where m_{ij} is 1 if the pair of rules $rule_i$ and $rule_j$ violate the constraint and 0 otherwise. k is the number of rules in the model. The NMI of a model is constrained between zero and one: it calculates the ratio of monotonic violating pairs over the total possible number of prediction pairs present in the model being tested, the lower a NMI is the better a model is considered. If this is the first rule in the partial model it will be automatically designated monotonic and be assigned a non-monotonicity Index of zero. The NMI is then incorporated into the quality metric by:

$$Q = (1 - \omega) \cdot Accuracy + \omega \cdot (1 - NMI) \ , \tag{7}$$

where Q is the quality of a model and ω is an adjustable weighting that sets the importance of monotonicity and accuracy to the overall rule quality. Note that Equation 7 can be used to calculate the quality of either a single rule (used during the soft pruner) or a complete list of rules (line 16 of Figure 1).

Hard Pruning The hard monotonic pruner enforces the monotonic constraints rigidly. It operates on a list of rules as follows: (1) the NMI of a list is first calculated (Equation 6); (2) if it is non zero, the last term of the final rule is removed or, if the rule contains no terms, the rule is removed; (3) the NMI is then recalculated for the modified list of rules. This is repeated until the NMI of the rule list is zero. Finally the default rule is added to the end of the list if it has been removed and the new monotonic rule list is returned.

4 Results

*c*Ant-Miner$_{\text{PB+MC}}$ has been compared to a majority classifier (ZeroR [18]), the original *c*Ant-Miner$_{\text{PB}}$ and a modified OLM [2]. The original OLM algorithm constrained all attributes, however our modified OLM constrains a single attribute to allow a fair comparison between the algorithms. The decision to only constrain a single attribute is more realistic to real world applications as it is unlikely that a monotonic relationship is present for every attribute. Forcing a relationship upon an algorithm is likely to negatively impact its performance.

In all experiments *c*Ant-Miner variations were configured with a colony size of 5 ants, 500 iterations, minimum cases covered by an individual rule of 10, uncovered instance ratio of 0.01, and constraint weighting (ω) of 0.5 (only used by *c*Ant-Miner$_{\text{PB+MC}}$). The four chosen algorithms were tested on five data sets taken from the UCI Machine Learning Repository [10]. Table 2 present the details of the chosen data sets, including a summary of the constraints used. All independent attributes had their NMI calculated to discover good monotonic relationships—the NMI results guided the choice of constrained attribute reported in the table.

Table 3 shows the predictive accuracy of all algorithms on the 5 data sets, with standard deviation shown in brackets. All results are the average of tenfold cross-validation, with the stochastic ACO-based algorithms running 5 times[2] on each fold to average out random differences.

The results show that *c*Ant-Miner$_{\text{PB+MC}}$ outperformed the majority classifier in every data set. OLM and the original *c*Ant-Miner$_{\text{PB}}$ implementation were beaten by *c*Ant-Miner$_{\text{PB+MC}}$ in four of the five data sets. The good performance of *c*Ant-Miner$_{\text{PB+MC}}$ compared to *c*Ant-Miner$_{\text{PB}}$ is very positive: it shows that using a pruning mechanism to enforce monotonic constraints does not affect the search process and the algorithm is able to create monotonic classification rules with good predictive accuracy

We further analysed the results of OLM and *c*Ant-Miner$_{\text{PB+MC}}$—both algorithms that enforce monotonic constraints—for statistical significance: *c*Ant-Miner$_{\text{PB+MC}}$ achieved statistically significantly better results than OLM in 3 out of 5 datasets, according to the Wilcoxon test with a significance level of 0.05. *c*AntMiner$_{\text{PB+MC}}$ enforces monotonic constraints on the entire list of rules, allowing global optimisation of monotonicity. OLM performs a local optimisation

[2] ACO-based algorithms therefore run a total of 50 times before the average is taken.

Table 2. The five UCI [10] data sets used in experiments including attribute and constraint information. The constraints information contain the attribute name, direction of constraint either ↑ (increasing) or ↓ (decreasing) and its corresponding NMI.

		Attributes		Constraint		
Name	Size	Nominal	Continuous	Constrained Attribute	Direction	NMI
Cancer	698	0	10	Uniformity of Cell Size	↑	0.0059
Car	1727	6	0	Safety	↑	0.0460
Haberman	305	0	3	Positive Axillary Nodes	↑	0.0861
MPG	397	0	7	Horsepower	↓	0.0566
Pima	767	0	8	Plasma Glucose Conc.	↑	0.0947

Table 3. Accuracy results for the four algorithms being tested, the accuracy is based on the average of 10 cross-validation runs with the standard deviation shown in brackets. The datasets where cAnt-Miner$_{\mathrm{PB+MC}}$'s performance is statistically significantly better than OLM (according to the Wilcoxon test with a significance level of 0.05) are marked with the symbol ▲; if no symbol is shown, no significant difference was observed. The best results are shown in boldface.

Data set	ZeroR	cAnt-Miner$_{\mathrm{PB}}$	OLM	cAnt-Miner$_{\mathrm{PB+MC}}$
Cancer	0.6552 [0.0156]	**0.9566 [0.0181]**	0.8355 [0.0149]	0.9554 [0.0178] ▲
Car	0.7002 [0.0201]	0.8929 [0.0151]	0.**9055 [0.0187]**	0.8954 [0.0154]
Haberman	0.7353 [0.0985]	0.7405 [0.0790]	0.6993 [0.0781]	**0.7552 [0.0664]** ▲
MPG	0.7286 [0.0542]	0.9200 [0.0293]	0.7663 [0.0367]	**0.9240 [0.0353]** ▲
Pima	0.6510 [0.0420]	0.7493 [0.0564]	0.7161 [0.0589]	**0.7599 [0.0640]**

as a rule cannot be added to the current list if it was to break the monotonicity of existing rules. This observation, together with the use of an ACO search strategy that aims at optimising both the accuracy and monotonicity of a model, are likely to account for the increased performance of cAnt-Miner$_{\mathrm{PB+MC}}$ over OLM.

5 Conclusions

This paper presented an extension to cAnt-Miner$_{\mathrm{PB}}$ that enforces monotonic constraints, called cAnt-Miner$_{\mathrm{PB+MC}}$. This is achieved by modifying the pruning strategies used during solution construction: soft constraints are used to modify the quality of rules and this their pheromone levels; hard constraints were then enforced by a global pruner operating on the entire list of rules. Monotonicity is a global property of a data set, therefore the creation of complete list of rules rather than individual rules allows cAnt-Miner$_{\mathrm{PB+MC}}$ to optimise the monotonicity of a model. cAnt-Miner$_{\mathrm{PB+MC}}$ has been shown to outperform a majority classifier and an existing monotonic algorithm, while not losing predictive accuracy when compared to the original implementation.

Currently the global pruner is naïve in its approach, as it simply removes the last term in a rule list. Further work is required to optimise the pruning strategy, one approach is to remove the term that improves the monotonicity of the list by the greatest amount.

References

1. Ben-David, A.: Monotonicity maintenancs in information-theoretic machine learning algorithms. Machine Learning 19, 29–43 (1995)
2. Ben-David, A., Sterling, L., Tran, T.: Adding monoticity to learning algorithms may impair their accuracy. Expert Systems with Applications 36, 6627–6634 (2009)
3. Dorigo, M., Maniezzo, V., Colorni, A.: Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B 26, 29–41 (1996)
4. Dorigo, M., Stutzle, T.: Ant Colony Optimization. A Bradford Book (2004)
5. Duivesteijn, W., Feelders, A.: Nearest neighbour classification with monotonicity constraints. In: Machine Learning and Knowledge Discovery in Databases, vol. 5211 5211, pp. 301–316. Springer Berlin Heidelberg (2008)
6. Fayyad, U., Piatetsky-Shapiro, G., Smith, P.: From data mining to knowledge discovery: an overview. In: Advances in Knowledge Discovery & Data Mining. pp. 1–34. MIT Press (1996)
7. Feelders, A., Pardoel, M.: Pruning for monotone classification trees. In: Advances in intelligent data analysis V, pp. 1–12. Springer (2003)
8. Fürnkranz, J.: Separate-and-conquer rule learning. Artificial Intelligence Review 13(1), 3–54 (1999)
9. Hoover, K., Perez, S.: Three attitudes towards data mining. Journal of Economic Methodology 7(2), 195–210 (2000)
10. Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`
11. Martens, D., Backer, M.D., Haesen, R., Baesens, B., Mues, C., Vanthienen, J.: Ant-based approach to the knowledge fusion problem. In: Ant Colony Optimization and Swarm Intelligence, pp. 84–95. Springer (2006)
12. Martens, D., Backer, M.D., Haesen, R., Vanthienen, J., Snoeck, M., Baesens, B.: Classification with ant colony optimization. IEEE Transactions on Evolutionary Computation 11(5), 651–665 (2007)
13. Martens, D., Baesens, B., Fawcett, T.: Editorial survey: swarm intelligence for data mining. Machine Learning 82(1), 1–42 (2011)
14. Otero, F., Freitas, A., Johnson, C.: A New Sequential Covering Strategy for Inducing Classification Rules With Ant Colony Algorithms. IEEE Transactions on Evolutionary Computation 17(1), 64–76 (2013)
15. Parpinelli, R., Lopes, H., Freitas, A.: Data mining with an ant colony optimization algorithm. IEEE Transactions on Evolutionary Computation 6(4), 321–332 (August 2002)
16. Potharst, R., Ben-David, A., van Wezel, M.: Two algorithms for generating structured and unstructured monotone ordinal data sets. Engineering Applications of Artificial Intelligence 22(4), 491–496 (2009)
17. Qian, Y., Xu, H., Liang, J., Liu, B., Wang, J.: Fusing monotonic decision trees. Knowledge and Data Engineering, IEEE Transactions on 27(10), 2717–2728 (2015)
18. Witten, H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, 2nd edn. (2005)