Citation for published version

DOI

Link to record in KAR
https://kar.kent.ac.uk/14459/

Document Version
UNSPECIFIED

Copyright & reuse
Content in the Kent Academic Repository is made available for research purposes. Unless otherwise stated all content is protected by copyright and in the absence of an open licence (eg Creative Commons), permissions for further reuse of content should be sought from the publisher, author or other copyright holder.

Versions of research
The version in the Kent Academic Repository may differ from the final published version. Users are advised to check http://kar.kent.ac.uk for the status of the paper. Users should always cite the published version of record.

Enquiries
For any further enquiries regarding the licence status of this document, please contact: researchsupport@kent.ac.uk

If you believe this document infringes copyright then please contact the KAR admin team with the take-down information provided at http://kar.kent.ac.uk/contact.html
A New Ant Colony Algorithm for Multi-Label Classification with Applications in Bioinformatics

Allen Chan and Alex A. Freitas
Computing Laboratory
University of Kent
Canterbury, CT2 7NZ, UK
achan.83@googlemail.com, A.A.Freitas@kent.ac.uk

ABSTRACT

The conventional classification task of data mining can be called single-label classification, since there is a single class attribute to be predicted. This paper addresses a more challenging version of the classification task, where there are two or more class attributes to be predicted. We propose a new ant colony algorithm for the multi-label classification task. The new algorithm, called MuLAM (Multi-Label Ant-Miner) is a major extension of Ant-Miner, the first ant colony algorithm for discovering classification rules. We report results comparing the performance of MuLAM with the performance of three other classification techniques, namely the very simple majority classifier, the original Ant-Miner algorithm and C5.0, a very popular rule induction algorithm. The experiments were performed using five bioinformatics datasets, involving the prediction of several kinds of protein function.

Categories and Subject Descriptors
I.2.6 [Artificial Intelligence]: Learning – concept learning, induction.

General Terms
Algorithms, Performance, Experimentation.

Keywords
Ant Colony Optimization, Data Mining, Bioinformatics.

1. INTRODUCTION

This work proposes a new ant colony algorithm tailored for a kind of classification task in data mining, called multi-label classification. In essence, this is a more challenging version of the conventional (single-label) classification task, as follows. In conventional classification the goal is to predict a single class for an example (record), based on the values of predictor attributes describing that example. By contrast, in multi-label classification there are two or more classes to be predicted for an example. A more detailed discussion of the differences between single-label and multi-label classification will be discussed in section 2. For now it should be noted that multi-label classification is an active and increasingly important research area, due to the growing interest in datasets which naturally have multiple classes to be predicted, particularly in the areas of text mining and bioinformatics [12], [17], [3].

The proposed ant colony algorithm is called MuLAM (Multi-Label Ant-Miner), and is a major extension of the Ant-Miner algorithm proposed in [13]. Ant-Miner addresses the conventional, single-label classification task. It discovers classification rules of the form:

IF (conditions) THEN (predicted class)

with the meaning that, if an example satisfies the conditions in the rule antecedent, that example is assigned the class predicted by the rule consequent. In the rules discovered by Ant-Miner, each consequent contains exactly one predicted class.

MuLAM extends this rule representation to allow more than one predicted classes in the rule consequent. This extension in the kind of knowledge discovered by the algorithm required a major re-design of several parts of the Ant-Miner algorithm, as will be discussed in section 4.

The remainder of this paper is organised as follows. Section 2 discusses single-label and multi-label classification. Section 3 presents a review of the original Ant-Miner algorithm. Section 4 describes the proposed ant colony algorithm for multi-label classification. Section 5 reports computational results evaluating the proposed algorithm. Section 6 concludes the paper and suggests future work.

2. SINGLE-LABEL VS. MULTI-LABEL CLASSIFICATION

Classification is one of the most investigated data mining tasks, with numerous commercial and industrial applications [20]. In essence, the classification task consists of discovering knowledge that can be used to predict the class of an example (record) whose class is unknown, based on the values of predictor attributes describing the example. This task generally involves splitting a data set into a training set and a test set. A classification algorithm is applied to all examples in the training set, where the class of each example is available to the algorithm. The algorithm analyses the relationship between the predictor attributes and the class for all training examples, and discovers a classification model for the data. Then the discovered model is applied to examples in the test set, where the class of each example is unknown to the system, in order to evaluate the predictive accuracy of the discovered model. It is crucial that the training and test sets contain disjoint sets of examples, i.e., the test set examples should never be used in the training set, in order to characterize a truly predictive scenario. We can then compute a measure of predictive accuracy on the test set. More precisely, for each example in the test set, the class predicted by the classification model is compared with the actual class of the example, in order to evaluate whether or not the prediction was correct. Then the standard definition of the predictive accuracy of a classification model is simply the number of examples in the test set correctly classified by that model divided by the total number of examples in the test set.

There are two versions of the classification task, according to the number of classes to be predicted for each example: a)
Single-Label Classification and b) Multi-Label Classification. Single-label classification refers to the standard task of classification, where there is only one class attribute (target attribute) to be predicted.

The basic principles in multi-label classification are similar to those in single-label classification; however, in multi-label classification there are two or more class attributes to be predicted. As a result, the consequent of a classification rule contains one or more predictions, each prediction involving a different class attribute.

There has been relatively little work in the area of multi-label classification, by comparison with the vast amount of work in standard single-label classification. In addition, most of the works in multi-label classification have been applied to text classification [10], [12], [17]. An exception is the work of [3], which has been applied to the area of bioinformatics.

Traditional classification algorithms are unable to cope with a multi-label dataset, since those algorithms predict a single class attribute. A simple workaround is to split the original dataset into near identical datasets, where each contains all predictor attributes and their values for each example, but each dataset produced in this way contains only one of the class attributes to be predicted. This results in requiring the classification algorithm to be trained on nearly the same dataset multiple times. More precisely, the algorithm has to be run once for predicting each of the class attributes. This is not a very good solution to the problem of multi-label classification [3], [18], for two main reasons. First, we would discover a set of rules for predicting each class attribute, but each of the class attributes would be treated individually, ignoring possible correlations between class attributes. Intuitively, an algorithm that discovers rules predicting more than one class attribute can capture some correlations between class attributes and discover a simpler rule set (with a smaller number of rules) than an algorithm that discovers only rules predicting a single class attribute. Second, the approach of running the classification algorithm once for each class attribute has the drawback of being computationally expensive.

One other approach that can be used to solve the multi-label classification problem consists of converting the existing class attributes into a single class attribute, where each value of this new class attribute represents a combination of the class attributes that were initially present in the data set. Using a simple example to illustrate this, consider a data set with three class attributes to be predicted, where each class attribute can have the value of either yes or no. Table 1 illustrates the possible combinations of class values in this dataset.

Table 1 shows that it is possible to convert multi-label problems into a single-label problem. But it is also evident that, by carrying out such a conversion, the number of values of the new single-class attribute will increase exponentially with the number of original class attributes. Hence, it becomes increasingly more difficult to predict a class value, as the number of examples associated with any given value of the new single class attribute decreases considerably, reducing the amount of information to effectively predict each class value.

In order to avoid these disadvantages associated with the conversion of a multi-label problem into one or more single-label problems, this paper directly addresses the multi-label classification task. That is, we propose a new multi-label classification algorithm (described in section 4) that was designed so that different class attributes can be potentially predicted using the same rule antecedent, which shows some correlations between the class attributes to be predicted.

### Table 1. Transforming a multi-label problem into a single-label problem

<table>
<thead>
<tr>
<th>Class Attr. 1</th>
<th>Class Attr. 2</th>
<th>Class Attr. 3</th>
<th>Single class Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>YYY</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>YYN</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>YNY</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>NYY</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>NNY</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>NNN</td>
</tr>
</tbody>
</table>

3. AN OVERVIEW OF ANT-MINER

At first glance, ants are seen as small unintelligent individuals, but on closer inspection, as a collective group (a swarm) they appear to be highly organised and yet require no supervision at all [1], [2]. The “intelligent” foraging behaviour of ant colonies has been studied in detail, as discussed in [5], [8]; and these studies have led to the development of the ACO (Ant Colony Optimisation) meta-heuristic, on which the Ant-Miner algorithm is based. Hence, before we review Ant-Miner, let us first briefly review this meta-heuristic.

The ACO meta-heuristic, as proposed by [6], is normally used to solve discrete optimization problems. In essence, each ant corresponds to a candidate solution to the target problem. The search space is conceptually represented as a graph, where nodes correspond to parts of a candidate solution and edges correspond to movements performed by ants in the search space. Hence, the path followed by an ant in that graph corresponds to the process of incrementally constructing a candidate solution. In other words, when an ant follows an edge leading to a given node \( i \), the part of the candidate solution represented in node \( i \) is added to the current candidate solution. Each ant keeps following a path in the graph incrementally constructing a candidate solution, until a complete solution is constructed.

This process is performed by a population of ants (an ant colony) for a number of iterations. During the construction of a candidate solution, an ant usually has to choose between two or more paths, i.e., it has to choose which edge it will follow (or which node it will visit) next. This choice depends on two factors, namely:

- The value of a problem-dependent heuristic function associated with each edge or node of the graph representing the search space.
- The amount of pheromone associated with each edge or node.

When an ant follows a path in the graph, it updates the amount of pheromone along that path. More precisely, the amount of pheromone deposited on the path followed by an ant is proportional to the quality of the candidate solution represented by that path. It is this pheromone updating mechanism that implements the concept of stigmergy [9], where ants modify the environment (amount of pheromone on edges or nodes of the graph) as an indirect means of communication, which allows them to cooperate to find good solutions to the target problem.

3.1 Ant-Miner

Ant-Miner was initially developed by Parpinelli and his colleagues [13], [14]. It was the first ACO algorithm for discovering classification rules and it has been shown to be
competitive against the CN2 [4] and C4.5 [16] algorithms for
classification. Ant-Miner generates solutions in the form of
classification rules. Conventional classification rules are in the
form of \text{IF} \langle\text{antecedent}\rangle \ \text{THEN} \ \langle\text{consequent}\rangle$, where the
\text{antecedent} contains an arbitrary number of \text{terms}, from zero to
to potentially the number of predictor attributes in the data being
mined. However, in practice one expects a rule to contain a
number of terms much smaller than the total number of
predictor attributes, since many attributes can be irrelevant to
predict the class of a given set of examples. Ant-Miner uses a
propositional logic representation, where each \text{term} takes the
form of a triplet \langle\text{attribute}, \text{operator}, \text{value}\rangle, where the
\text{attribute} is one of the predictor attributes in the data, the \text{value}
is one of the values that the \text{attribute} can take on, and the
\text{operator} is a relational operator. This relational operator could
in principle be $=, \neq, <, \leq, >, \geq$, but the original version of
Ant-Miner has the limitation that it can only process discrete
values – a common limitation in ACO algorithms. As a result,
Ant-Miner only uses the `$=$` operator. So each \text{term} takes on
the triplet \langle\text{attribute} = \text{value}\rangle. Also, each \text{rule} cannot contain
the same \text{attribute} twice, e.g. \langle\text{sex=m} \ \text{AND} \ \text{sex=f}\rangle,
otherwise the rule would contain a contradiction and so it would
never be satisfied by any example.

TrainingSet = \{all training examples\};
DiscoveredRuleList = \{\}; /* rule list is
initialized with an empty list */
WHILE (TrainingSet > MaxUncoveredExamples)
  \text{t} = 1; /* ant index */
  \text{j} = 1; /* convergence test index */
  Initialize all trails with the same amount of
  pheromone;
REPEAT
  \text{Ant}, starts with an empty rule and
  incrementally constructs a classification
  rule \text{R}_t \text{by adding one term at a time to the}
  current rule;
  Prune rule \text{R}_t;
  Update the pheromone of all trails by
  increasing pheromone in the trail followed
  by \text{Ant}; (proportional to the quality of \text{R}_t)
  and decreasing pheromone in the other trails
  (simulating pheromone evaporation);
  IF (\text{R}_t \text{is equal to} \text{R}_t - 1) /* update
  convergence test */
    THEN \text{j} = \text{j} + 1;
    ELSE \text{j} = 1;
  END IF
  \text{t} = \text{t} + 1;
UNTIL (\text{i} \geq \text{No_of_ants}) \text{OR}
  (\text{j} \geq \text{No_rules_converg})
  Choose the best rule \text{R}_{\text{best}} among all rules \text{R}_t
  constructed by all the ants;
  Add rule \text{R}_{\text{best}} to \text{DiscoveredRuleList};
  TrainingSet = TrainingSet - \{set of examples
correctly covered by \text{R}_{\text{best}}\};
END WHILE

The \text{consequent} contains the class attribute value to be predicted
by the rule. Ant-Miner however was designed initially to solve
a single-label classification task and so the consequent of the
rules generated by the original Ant-Miner will contain only one
class attribute value.

Let us now briefly review the main aspects of the rule discovery
process performed by Ant-Miner, as described by Pseudocode 1
[13]. For more details about the algorithm the reader is referred
to that reference.

Ant-Miner uses a sequential covering approach to discover a list
of classification rules, by discovering one rule at a time until all
or almost all the examples in the training set are covered by the
discovered rules. When the algorithm first starts, the training set
holds all the original training examples and the discovered rule
list is empty. Every iteration of the \text{WHILE} loop illustrated in
Pseudocode 1 creates a population of ants, each ant

\ienes the \text{antecedent} of the
\text{rule} which is selected using the same kind of roulette
wheel mechanism, and/or due to a local heuristic
function that only considers one attribute at a time, which has
the drawback of ignoring interactions between attributes.

Rule pruning consists of iteratively removing one term at a time
from the \text{rule} while this improves the \text{quality} of the \text{rule}. During
this stage, the \text{consequent} of the \text{rule} can change as the majority
class covered by the pruned rule can be different to that of the
original \text{rule}. This process repeats until there is only one term
left in the \text{rule}, or any term to be removed next will not improve
the \text{quality} of the \text{rule}.

After the \text{rule pruning stage}, \text{pheromone} levels of the \text{path}
taken by the \text{current ant} are increased. More precisely, the amount of
\text{pheromone} associated with each term in the \text{antecedent} of the
just-pruned is increased in proportion to the quality (predictive accuracy) of the rule. The other terms – i.e. the terms that are not present in the rule antecedent – have their pheromone reduced, to simulate pheromone evaporation in real ants. This is implemented by a simple normalization procedure: after increasing the pheromone of the terms used in the rule, the pheromone of each term (either used or not in the rule) is divided by the total sum of pheromones for all terms. Since the terms not used in the rule did not have their pheromone levels increased, their pheromone levels will be effectively reduced, by comparison with the terms used in the rule.

The REPEAT-UNTIL loop is repeated till at least one of the following terminating criteria is met:

• The number of constructed rules is equal or greater than the number of ants specified by the user.
• The rule constructed by Ant<sub>t</sub> is exactly the same as the rule constructed by the previous No_rules_conv<sub>err</sub> = 1 rules, where No_rules_conv<sub>err</sub> is a user-defined parameter.

4. THE PROPOSED MULTI-LABEL ANT-MINER

This section will describe the complete Multi-Label Ant-Miner (MuLAM) algorithm. Firstly, a high-level pseudocode is presented in Pseudocode 2, outlining the main functional behaviour of MuLAM.

At the start of the algorithm, MuLAM assigns all available training examples to the training set and it initialises the discovered rule list with an empty list. In MuLAM, each ant does not produce a single rule like in the original Ant-Miner. Rather, each ant discovers a candidate rule set – a very significant change. The reason for this is due to addressing the multi-label classification task, where there are multiple class attributes to be predicted. Each ant discovers at least one rule and at most a number of rules equal to the number of class attributes, i.e. a different rule for each class to be predicted. An ant will discover a single rule only in the case where that rule is considered good enough to predict all class attributes – according to a criterion to be defined later.

At the end of each outer WHILE loop iteration, examples that have all their class attributes correctly covered by any of the rules in the just discovered rule set are removed from the current training set. Hence, the training set gradually reduces in size. This brings us to the condition of this WHILE loop, which is executed as long as the number of examples left in the training set is greater than a user-defined parameter: MaxUncovExamples (maximum number of uncovered examples).

At the start of each iteration of the outer WHILE loop, the algorithm carries out pre-processing calculations which will be needed to calculate the probability of selecting a term to be added to a rule later. There are two kinds of calculation that are performed here, in the order they appear in Pseudocode 2. First, the algorithm calculates and stores the information gain [16] associated with each term. Note that the value of a term’s information gain does not change throughout the iteration of this outer WHILE loop. Secondly, a pheromone matrix is created for each class attribute. This is a generalisation of the original Ant-Miner where there is a single pheromone matrix because there is a single class attribute. Each of the pheromone matrices contains one cell for each term, representing the amount of pheromone associated with that term. Each pheromone matrix is initialized by assigning an amount of pheromone deposited directly proportional to the previously computed information gain of each term.

After the initialization of the pheromone matrices, the algorithm starts a REPEAT loop. Each iteration of this loop corresponds to a single ant constructing a candidate rule. The constructed rule can potentially be decomposed into a set of rules later in the algorithm, as will be explained later. The REPEAT loop stops when the ant with index <sub>t</sub> reaches a user-defined value, the parameter: MaxNoAnts, which is the maximum number of ants to be used for discovering a rule set in the current iteration of the outer WHILE loop.

Every ant in MuLAM starts off with an empty partial rule, i.e. a rule with no term in its antecedent. In addition, the rule set constructed by this ant, denoted RS<sub>t</sub>, is also initialized with the empty set. Next the WHILE statement inside this REPEAT loop decides if the current ant should proceed to select a term to be added to the partial rule, based on two conditions, both of which must be satisfied. The first condition makes sure that the ant can only select a new term to add to the partial rule if there are still unused attributes from the set of predictor attributes in the data. This condition is also used in Ant-Miner. The second condition ensures that there is still one or more classes that has not been predicted up to this point of the WHILE loop. This condition is used in MuLAM, but not in Ant-Miner. It represents an adaptation of MuLAM to the multi-label classification task. When both conditions evaluate true, the algorithm proceeds to the inside of this WHILE loop, where a new term is selected.

Each ant selects a term to potentially add to the current partial rule using a roulette wheel selection technique analogous to the roulette wheel selection method popularly used in evolutionary algorithms [7]. Each term occupies a slot of the roulette wheel with size proportional to the probability of selecting that particular term. The probability of selecting any given term is given by the product of the amount of pheromone associated with that term and the value of a heuristic measure for that term. Once a term has been selected, this term is only added to the current partial rule as long as it satisfies the condition of the IF statement; that is, as long as the inclusion of the selected term in the partial rule does not make the antecedent cover a number of examples smaller than the parameter MinExamplesPerRule (Minimum Number of Examples per Rule).

Once a term has been successfully added to the antecedent of the current partial rule, the algorithm then tries to make a prediction for each and every class attribute in the training set that has not been predicted up to this point of the algorithm.

Before the algorithm makes a prediction for this current partial rule, it initialises the rule consequent with the empty set. This rule consequent holds all class attribute values that are being predicted by the rule. The ant enters the FOR loop, where it processes each class attribute separately. So for every class attribute, the algorithm then decides under a certain pre-pruning criteria whether the current class attribute should be added to the rule consequent as a prediction.

The pre-pruning criteria used in MuLAM is based on Cramer’s V coefficient [11]. This criterion consists of applying pre-pruning when the value of Cramer’s V coefficient is greater than a certain threshold calculated based on the data, using the method described in [18]. If this criterion is satisfied, then the current class attribute is predicted by the rule. This means the algorithm adds to the rule consequent a term \(<C_i, V^>_i\) where \(C_i\) is the current (i-th) class attribute and \(V^>_i\) is the value of \(C_i\) having the largest frequency among all examples covered by the rule. This class attribute \(C_i\) is then marked as predicted for this ant. After the FOR loop, if the rule consequent is not empty, i.e. it contains one or more class attribute-value pairs, then the ant creates a complete rule using the current rule antecedent and
In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006), pp. 27-34

Pseudocode 2. A high-level description pseudocode of Multi-Label Ant-Miner (MuLAM)

As previously explained, inside the second WHILE loop, the condition in the first IF statement determines whether the current ant should add the newly selected term to its rule antecedent. If this condition fails, then the algorithm will not...
run the rest of the procedures within this second WHILE loop. Instead it exits the WHILE loop prematurely, and proceeds to the IF statement right after this WHILE loop. This IF statement tests if there are still unpredicted class attributes left for this ant to predict. If so, we need a method to complete predicting these class attributes. This is where the IF statement takes over and builds one rule using original Ant-Miner’s rule generation procedure, whereby the rule antecedent is constructed by adding one term at a time to the rule antecedent until the addition of a term causes the antecedent to cover less than a predefined number of examples (similar to the \textit{MinExamplesPerRule} parameter in MuLAM). The rationale for this step is that for the class attributes that are not predicted up to this point, it is better to build a new rule predicting the classes not predicted by the current rule than to attempt to correct the current rule, which is badly predicting these classes. This is because the current ant has already found a good rule predicting a subset of all class attributes, and by attempting to correct this rule to predict those currently unpredicted classes, the result would tend to be a bad rule predicting the majority of the classes. Once the antecedent of a rule is finished, in Ant-Miner the majority value of the class attributes is predicted. The majority class value is simply the class value with the largest frequency in the set of examples covered by the rule. In this IF statement, MuLAM generates each rule in a similar way as to Ant-Miner. The difference is that, instead of predicting just one class as in Ant-Miner, MuLAM will predict the majority value of each class attribute that has not been predicted up to this point, by creating one complete rule for those class attributes.

If a rule has been generated as a result of the IF statement mentioned above, this rule will undergo pruning as this rule can potentially be very large with respect to the number of terms in the rule antecedent as mentioned in [13]. The pruning technique used in MuLAM is an iterative procedure partly inherited from original Ant-Miner, whereby in each iteration the term whose removal best improves the rule quality is pruned out, and this process repeats till the rule quality can no longer be improved. In Ant-Miner, the class value predicted by the rule can potentially change. With Ant-Miner, as there was always only a single class attribute to be considered, this was not a problem. However, with MuLAM, if we allowed the procedure to alter the predicted class values for several class attributes, this process would become very computationally expensive. After all, every time a term is evaluated for its removal, the training set would need to be scanned for the possibility of the class attributes’ values changing and, if we considered all possible combinations of class attributes’ values, there would be a large number of combinations of values of unpredicted class attributes to be considered. In particular, this number would seriously reduce the scalability of the algorithm to problems with many class attributes. Hence, in MuLAM, to avoid this problem, all the predicted class attribute values in the consequent of the rule being pruned remain the same during the pruning procedure.

Once the current ant has finished generating one or more rules to predict all class attributes, pheromone trails are then updated simulating real world ants where they lay pheromone as they travel along their selected paths, as explained earlier in the description of Ant-Miner (Section 3.1).

Pheromone updating is carried out for each rule in the set of rules constructed by the current ant. For each rule, the pheromone matrix associated with each class attribute predicted by the rule is updated, by increasing the amount of pheromone of all matrix cells referring to the terms occurring in the antecedent of the rule. The \texttt{REPEAT-UNTIL} loop terminates when the number of ants reaches a user-defined parameter: \texttt{MaxNoAnts} (maximum number of ants). When this terminating condition is met, from the set of rule sets discovered by all the ants, the best rule set is chosen. Each of the rules in that best rule set is then added to the \texttt{DiscoveredRuleList}, which holds all the rules that will be used to classify the test data. To determine which rule set is the best out of all the rule sets constructed by all ants during the entire \texttt{REPEAT} loop, each rule set has its quality computed as the average of the quality measure of all rules in that set. (The formula for computing a rule quality is a natural extension of the formula used in Ant-Miner, viz. the product Sensitivity (Se) \times Specificity (Sp) [13]. The extension is that, instead of computing this product for a single class attribute, in MuLAM the rule quality is the arithmetic average of this product over all class attributes predicted by the rule.) The rule set with the best quality is chosen to be added to the \texttt{DiscoveredRuleList}. This rule set is then used to mark the training examples it correctly covers, i.e. examples matching both the antecedent and the consequent of one of the rules in the discovered rule set. Since there are multiple class attributes in the training set, one discovered rule may only predict a subset of class attributes and so effectively MuLAM does not physically remove the examples from the training set. Instead it uses a virtual flagging system whereby for each class attribute predicted by a rule, the examples that match both the rule antecedent and the predicted class value are flagged to be considered out of the training set when later iterations of the algorithm try to predict this class attribute. That is, any future calculations regarding the number of examples (and referring to this class attribute) will not include these covered examples. Once a reduced training set is produced, the outer WHILE loop will continue to run provided that the number of examples which are not covered by the rules discovered so far is greater than the previously-mentioned parameter \texttt{MaxUncovExamples}.

5. COMPUTATIONAL RESULTS

This section will briefly explain the data sets used in the experiments and then present the obtained results.

5.1 Biological Data Sets

The data sets used in our experiments originate from Uniprot [19], which is one of the largest bioinformatics databases holding information on sequenced proteins and their functions. Each record of the Uniprot database essentially contains information about a protein. We obtained, from Uniprot, 5 datasets, each with two class attributes to be predicted, as shown in Table 2. In order to create the predictor attributes for these datasets, out of the many fields describing a protein in Uniprot, we used a field which has a set of references to PROSITE patterns [15]. In other words, each protein’s record contains a reference to each PROSITE pattern (a biological motif) present in that protein. Prosite is actually a separate database which stores sequenced protein families and domains. Hence, our datasets were created by using cross references from Uniprot to the PROSITE database for each of the proteins included in our datasets. Each PROSITE pattern is used as a binary attribute. In other words, in each dataset, each example (protein) is described by a set of binary attributes, each attribute taking the value “yes” or “no”, indicating whether or not the corresponding PROSITE pattern is present in that protein. Therefore, the discovered rules take, for instance, the following form: “\text{IF } (prX = \text{yes}) \text{ AND } (prY = \text{no}) \ldots \text{ THEN (anti-oncogene = yes) AND (apoptosis = no}) \rangle”, where prX and prY are certain PROSITE patterns, whilst anti-oncogene and apoptosis are class attributes of dataset 1 in Table 2. The class attributes and the number of attributes and examples for each dataset are summarised in Table 2.
Table 2. Summary of five data sets used in experiment

<table>
<thead>
<tr>
<th>Data set</th>
<th>Class Attributes</th>
<th>No. of Attributes</th>
<th>No. of Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Anti-oncogene</td>
<td>153</td>
<td>540</td>
</tr>
<tr>
<td>2</td>
<td>Cell-cycle</td>
<td>156</td>
<td>1343</td>
</tr>
<tr>
<td>3</td>
<td>DNA-repair</td>
<td>102</td>
<td>1872</td>
</tr>
<tr>
<td>4</td>
<td>DNA-repair</td>
<td>101</td>
<td>1826</td>
</tr>
<tr>
<td>5</td>
<td>DNA-damage</td>
<td>34</td>
<td>622</td>
</tr>
</tbody>
</table>

5.2 Results

We applied Multi-Label Ant-Miner (MuLAM) to each of the data sets listed in Table 2, and compared these results with the results of three other classification techniques. First, as a very simple baseline, we used the trivial majority classifier technique to classify the examples in the test set. This technique simply assigns, to every test example (unseen during training), the class with the largest frequency in the set of training examples. Second, we used the original Ant-Miner algorithm. Third, we used Clementine’s implementation of the popular, industrial-strength C5.0 algorithm. Since Ant-Miner and C5.0 are single-label algorithms, they were run twice for each dataset. These two runs used the same set of predictor attributes, but each run aimed at discovering rules predicting the value of a different class attribute. This approach has the disadvantages discussed in section 2, but it is a fair way of comparing MuLAM with the above techniques. The comparison with original Ant-Miner is important since MuLAM is a major extension of Ant-Miner, and the comparison with C5.0 is important because this is a very popular classification-rule discovery algorithm.

Table 3. Number of ants used in Ant-Miner for each dataset

<table>
<thead>
<tr>
<th>Data Set</th>
<th>MuLAM (time in sec)</th>
<th>Ant-Miner (time in sec)</th>
<th>Max No of Ants in Ant-Miner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>228.7</td>
<td>210.4</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>627.2</td>
<td>644.7</td>
<td>450</td>
</tr>
<tr>
<td>3</td>
<td>308.1</td>
<td>310.0</td>
<td>250</td>
</tr>
<tr>
<td>4</td>
<td>266.9</td>
<td>265.0</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>67.8</td>
<td>61.2</td>
<td>3000</td>
</tr>
</tbody>
</table>

All experiments with each of the four techniques (MuLAM plus the other three techniques) were conducted running a 5-fold cross validation procedure [20] for each dataset. When running this procedure, exactly the same folds (partitions) of the data were used by each of the four techniques, in order to make their comparison as fair as possible.

C5.0 was used with its default parameters in all datasets. MuLAM was used with the following default parameters in all datasets: MaxUncovExamples = 10, MinExamplesPerRule = 10, MaxNoAnts = 100. These values of MaxUncovExamples and MinExamplesPerRule are actually the default values of these parameters in Ant-Miner too. Ant-Miner was used with its default values in all datasets, with the exception of its parameter MaxNoAnts, which was set to a different value for each dataset in order to perform more controlled experiments, for the following reason. We wanted to compare MuLAM and Ant-Miner by giving each algorithm roughly the same amount of computational time to solve the target classification problem. Otherwise the better result of an algorithm could be attributed just to it spending more time to solve the problem, rather than be due to its better effectiveness in discovering accurate rules. In order to give MuLAM and Ant-Miner roughly the same computational time in a controlled way, we first ran MuLAM and, for each dataset, we set the parameter MaxNoAnts of Ant-Miner to a value which makes an Ant-Miner run to take about the same amount of time as a MuLAM run. The parameter MaxNoAnts was chosen to be varied in these experiments because this is the parameter that most influences the computational time of Ant-Miner. Table 3 shows, for each dataset, the resulting MaxNoAnts value adjusted for Ant-Miner and the computational time taken for each algorithm, in seconds.

The predictive accuracy for each algorithm, for each class attribute in each dataset, is reported in Table 4. The numbers after the “±” symbol denote standard deviations. In the last three columns of Table 4, some cells are marked by (+), which means the corresponding accuracy is significantly better than MuLAM’s accuracy. (In principle a cell in the last three columns could alternatively be marked as (+), which would mean the corresponding accuracy is significantly worse than MuLAM’s accuracy for the same class attribute, by this result was not observed in Table 4.) A difference in accuracy was considered significant if the corresponding standard deviation intervals do not overlap. The majority classifier’s accuracy was significantly lower than MuLAM’s accuracy in 6 out of the 10 class attributes in Table 4. In the other 4 class attributes the differences in accuracies obtained by these two techniques was not significant. There was no significant difference between the

Table 4. Predictive accuracy (%) in the test set for each algorithm, using 5-fold cross validation

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Class Attributes</th>
<th>MuLAM</th>
<th>Majority classifier</th>
<th>Ant-Miner</th>
<th>C5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Anti-oncogene</td>
<td>79.5±3.56</td>
<td>77.41±0.13</td>
<td>72.5±18.61</td>
<td>77.41±0.51</td>
</tr>
<tr>
<td></td>
<td>Apoptosis</td>
<td>85.09±2.57</td>
<td>85.74±0.13</td>
<td>76.25±23.42</td>
<td>88.33±5.02</td>
</tr>
<tr>
<td>2</td>
<td>Cell-cycle</td>
<td>63.27±5.54</td>
<td>53.98±0.03 (-)</td>
<td>67.51±7.17</td>
<td>53.90±0.00 (-)</td>
</tr>
<tr>
<td></td>
<td>Cell-division</td>
<td>78.87±1.65</td>
<td>77.29±0.03 (-)</td>
<td>71.87±16.09</td>
<td>77.17±0.20</td>
</tr>
<tr>
<td>3</td>
<td>DNA repair</td>
<td>97.20±3.0</td>
<td>85.79±0.03 (+)</td>
<td>97.68±1.35</td>
<td>85.65±0.12 (+)</td>
</tr>
<tr>
<td></td>
<td>DNA damage</td>
<td>92.09±2.24</td>
<td>78.63±0.01 (-)</td>
<td>93.85±3.24</td>
<td>78.51±0.15 (-)</td>
</tr>
<tr>
<td>4</td>
<td>SOS response</td>
<td>9.21±2.19</td>
<td>87.96±0.00 (+)</td>
<td>99.58±0.00</td>
<td>87.76±0.13 (+)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>82.82±1.17</td>
<td>70.87±0.03 (-)</td>
<td>92.52±6.01</td>
<td>70.71±0.13 (-)</td>
</tr>
<tr>
<td>5</td>
<td>DNA damage</td>
<td>84.70±9.19</td>
<td>64.31±0.07 (+)</td>
<td>96.75±4.84</td>
<td>64.00±0.00 (+)</td>
</tr>
<tr>
<td></td>
<td>SOS response</td>
<td>85.02±4.60</td>
<td>85.53±0.02</td>
<td>91.24±14.22</td>
<td>85.12±0.44</td>
</tr>
</tbody>
</table>

In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2006), pp. 27-34
accuracies obtained by MuLAM and Ant-Miner in any of the 10 class attributes. Finally, C5.0’s accuracy was significantly lower than MuLAM’s accuracy in 6 class attributes, and there was no significant difference in the other 4 class attributes.

Table 5 Some rules found by MuLAM and Ant-Miner

<table>
<thead>
<tr>
<th>MuLAM’s Rule:</th>
<th>IF PS00321=0 THEN DNA-repair=1 DNA-damage=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant-Miner’s Rules:</td>
<td>IF PS00321=0 AND PS00162=1 THEN DNA-repair=1</td>
</tr>
<tr>
<td></td>
<td>IF PS00321=0 AND PS00618=0 THEN DNA-damage=0</td>
</tr>
</tbody>
</table>

Recall that, unlike Ant-Miner, MuLAM is a multi-label classifier and as such it will try to predict one or more class attributes with the same rule when possible. Table 5 shows examples of rules produced by MuLAM and Ant-Miner. The top section shows a rule discovered by MuLAM how and how the rule predicts two classes (DNA-repair = 1 and DNA-damage = 0). The bottom section shows two rules discovered by Ant-Miner, one of them predicting only the class DNA-repair = 1 and the other one predicting only the class DNA-damage = 0. Hence, it is this MuLAM’s accuracy and Ant-Miner’s accuracy in the same rule antecedent) two class attributes, which explicitly shows some correlations between different class attributes. Ant-Miner is of course unable to discover such correlations, since it is a single-label classification algorithm.

Recall that all results reported here used default parameters for all algorithms, in order to make the comparison among the algorithms as fair as possible. One direction for future work is to try to optimise the parameters of each algorithm to the datasets used in the experiments, to maximize the accuracy of the discovered rules. Another future work is to do experiments with more datasets and more class attributes per dataset.

6. CONCLUSION AND FUTURE WORK

The results of the experiments showed that, overall, MuLAM obtained predictive accuracies considerably better than the predictive accuracies obtained by the simple majority classifier and by C5.0. This clear superiority over the majority classifier was expected, given the extreme simplicity of that classifier, which actually ignores the values of all predictor attributes. The superiority over C5.0 was a positive result which was not expected, considering that C5.0 is an industry-strength algorithm resulting from several decades of research in decision tree induction, whereas MuLAM is a new algorithm. On the other hand, there was no significant difference between MuLAM’s accuracy and Ant-Miner’s accuracy in the experiments reported here. In any case, MuLAM at least has the advantage of discovering some rules that predict (using the same rule antecedent) two class attributes, which explicitly shows some correlations between different class attributes. Ant-Miner is of course unable to discover such correlations, since it is a single-label classification algorithm.

Recall that all results reported here used default parameters for all algorithms, in order to make the comparison among the algorithms as fair as possible. One direction for future work is to try to optimise the parameters of each algorithm to the datasets used in the experiments, to maximize the accuracy of the discovered rules. Another future work is to do experiments with more datasets and more class attributes per dataset.

7. REFERENCES