Noname manuscript No.
(will be inserted by the editor)

An Ant Colony Based Semi-Supervised Approach for Learning Classification Rules

Julio Albinati · Samuel E. L. Oliveira · Fernando E. B. Otero · Gisele L. Pappa

Received: date / Accepted: date

Abstract Semi-supervised learning methods create models from a few labeled instances and a great number of unlabeled instances. They appear as a good option in scenarios where there is a lot of unlabeled data and the process of labeling instances is expensive, such as those where most Web applications stand. This paper proposes a semi-supervised self-training algorithm called Ant-Labeler. Self-training algorithms take advantage of supervised learning algorithms to iteratively learn a model from the labeled instances and then use this model to classify unlabeled instances. The instances that receive labels with high confidence are moved from the unlabeled to the labeled set and this process is repeated until a stopping criteria is met, such as labeling all unlabeled instances. Ant-Labeler uses an ACO algorithm as the supervised learning method in the self-training procedure to generate interpretable rule-based models—used as an ensemble to ensure accurate predictions. The pheromone matrix is reused across different executions of the ACO algorithm to avoid rebuilding the models from scratch every time the labeled set is updated. Results showed that the proposed algorithm obtains better predictive accuracy than three state-of-the-art algorithms in roughly half of the datasets on which it was tested, and the smaller the number of labeled instances, the better the Ant-Labeler performance.

Keywords semi-supervised learning · self-training · ant colony optimization · classification rules

Julio Albinati and Samuel E. L. Oliveira have contributed equally to this work.

Julio Albinati (✉), Samuel E. L. Oliveira, Gisele L. Pappa
Federal University of Minas Gerais
Av. Antonio Carlos, 6627, Zip Code: 31.270-901
E-mail: {jalbinati, samuel.lima, glpappa}@dcc.ufmg.br

Fernando E. B. Otero
University of Kent
Chatham Maritime, Kent, ME4 4AG, UK
E-mail: F.E.B.Otero@kent.ac.uk

1 Introduction

Data availability, which was in the past a problem to test new algorithms and methods for knowledge discovery, is currently not an issue. Data coming from the Web 2.0, genomic and proteomics projects, governments, and other relevant sources, are waiting for data scientists to make sense of them. However, depending on what type of information we want to extract from these data, one valuable information is missing: labeled data (Ginestet, 2009, Li and Zhou, 2011, Triguero et al, 2015).

For a long period, researchers and practitioners tackled data analysis problems in a supervised or unsupervised manner, depending on the characteristics of the data available for learning or the resources available to label them. Supervised learning methods were used in cases where all instances available for training were labeled, while unsupervised learning methods took the opposite direction. In applications where data labeling was an expensive process, unsupervised methods were always considered the best option. However, supervised methods are proven to obtain better accuracy results than unsupervised ones (Zhu and Goldberg, 2009), and hence the ideal approach is to find a trade-off between labeling cost and accuracy. This is the rationale behind semi-supervised learning (SSL).

SSL methods create models from a few labeled instances and a great number of unlabeled instances (Chapelle et al, 2010). They appear as a good option in scenarios where there is a lot of unlabeled data and the process of labeling the data is expensive, such as those where most Web applications stand nowadays—e.g., there are large amounts of data being generated by social media websites. SSL can be used to solve clustering problems, where it is named constrained clustering (Davidson and Ravi, 2005), or semi-supervised classification task (SSC) (Chapelle et al, 2010), which is the problem we are interested in this paper.

Two different settings can be considered when dealing with SSC: transductive and inductive learning. In transductive learning approaches, the complete set of labeled and unlabeled instances are known in advance, and the classifier does not need to worry about model generalization—e.g., data disambiguation scenarios. In the inductive approach, in contrast, labeled and unlabeled instances are combined to build a model that will be used to predict the label of unseen data. The focus of this paper is on the latter.

Many different methods have already been proposed to solve SSC problems, including generative methods (Zhu and Goldberg, 2009), transductive inference with Support Vector Machines (Li and Zhou, 2011, Zhao et al, 2008) and probabilistic graphical models (Koutra et al, 2011). However, among the most popular methods are those named self-labeled (Blum and Mitchell, 1998, Triguero et al, 2015). Self-labeled methods take advantage of a supervised classifier to label instances of unknown class and include two well-known algorithms: co-training or self-training. Self-training iteratively learns a model M_k from the labeled instances L using any supervised learning algorithm and then uses M_k to classify instances in the set of unlabeled instances U. The instances that receive labels with high confidence are moved from U to L and the resulting L' is then used to train a new classification model M_{k+1}. This process is repeated until a stopping criterion, such as U becoming empty, is met.

As aforementioned, the supervised learning algorithm used by self-learning can be of any kind. This paper proposes a self-training algorithm, called Ant-Labeler,

which uses an ACO classification algorithm as a wrapper. ACO algorithms have been successfully applied in supervised classification, particularly in discovering classification rules (Martens et al, 2011, Otero et al, 2013). Ant-Labeler uses *c*Ant-Miner$_{\text{PB}}$ (Otero et al, 2013), an algorithm where each ant creates a complete list of classification rules, differently from the majority of ACO classification algorithms.

One of the motivations for using an ACO algorithm that generates rule lists as a self-learning wrapper, apart from being an accurate classification algorithm, is that the models generated are interpretable and can be used to understand how the labeling process evolves. Furthermore, these models are used in an ensemble, which is able to make more consistent decisions regarding which instances should be labeled. Finally, we can take advantage of its search mechanism to mitigate the potential drawback present in many self-training methods: the fact that each labeling round of the algorithm is usually independent of the previous, and models are built from scratch every time L is updated. In a rule-based ACO algorithm, this can be done in a straightforward manner, as the pheromone matrix represents the *state* of the search, which can be preserved from one iteration to the next one. We take advantage of this property of the ACO algorithm by preserving the pheromone matrix across different executions of the algorithm (i.e., labeling rounds), allowing each execution of the ACO to use the previous *state* of the search.

The remainder of this paper is organized as follows. Section 2 introduces the most popular algorithms for SSC and discusses semi-supervised approaches using ACO algorithms. Section 3 describes Ant-Labeler, the rule-based semi-supervised classification algorithm proposed in this paper. Section 4 presents the computational experiments comparing Ant-Labeler against other state-of-the-art SSC algorithms. Finally, Section 5 draws conclusions and discusses ideas for future work.

2 Related Work

This section introduces the basic notions of semi-supervised learning algorithms and reviews related work involving ACO algorithms. Semi-supervised methods work with both labeled and unlabeled data. They were initially studied by the natural language processing and text classification communities in order to automatically label data to improve learning (Ginestet, 2009, Wang et al, 2008).

One successful approach to tackle the semi-supervised learning problem is to use self-labeling methods, which take advantage of existing supervised learning algorithms. They use the results of the most confident predictions made by supervised methods to assign a class value to unlabeled instances. Two common self-labeling methods are the self-training and co-training algorithms. Both algorithms work as a wrapper, in the sense that they require another supervised learning algorithm to provide them confident predictions. The main difference between them is that co-training employs multi-view learning (Blum and Mitchell, 1998) and requires differentiated input data—i.e., data coming from independent data views, described by sufficient and redundant attribute subsets.

Self-training works with a single data view and is successfully used in many different contexts (Triguero et al, 2015). Self-training methods iteratively learn a model M_k from the labeled instances L and use M_k to classify unlabeled instances

in the set U. The instances that receive labels with high confidence are moved from U to L and the new L' is used to train a new classification model M_{k+1}. Each iteration of the labeling process is said to be a labeling round and the process goes on until a stopping criterion is met.

While the self-training approach is flexible and uses existing supervised learning algorithms, its performance is limited by the fact that the models used in each iteration are, most of the time, obtained independently. Hence, the underlining supervised algorithm has no knowledge of how it is being used and therefore cannot employ the models obtained in iterations $1, 2, ..., k$ in any way to obtain a new model M_{k+1}. One of the motivations of the algorithm proposed in this paper is to mitigate this limitation of most self-training methods by using a population of classifiers that evolve over time. At each step, models are built not only from the data but also from the models of previous labeling rounds by preserving the pheromone trails between different rounds.

To the best of our knowledge, there are two other algorithms following the self-training approach that also take advantage of information available from previous labeling rounds: Co-Forest and APSSC. Co-Forest (CO-trained Random FOREST) (Li and Zhou, 2007) uses a Random Forest to facilitate the process of determining the most confident instances to label and, in contrast with traditional co-training algorithms, does not require training data to be represented by sufficient and redundant attribute subsets. In Co-Forest, the unlabeled instances selected to be labeled by a given classifier are not removed from the set of unlabeled instances. Hence, they might be selected more than once to be labeled by different classifiers and receive distinct labels at different labeling iterations. APSSC (Aggregation Pheromone Density based Semi-Supervised Classification) (Halder et al, 2013), in contrast, works with the concept of temporary membership, where elements in the training set and their classes may change during the algorithm iterations, as detailed below.

Besides self-labeling approaches, many other methods for semi-supervised classification have been proposed. These methods can be divided into three main groups: generative models, low-density separation methods and graph-based methods (Chapelle et al, 2010). Generative models usually estimate the conditional density $p(x|y)$ of the data, where x is the attribute vector and y the class value (Zhu and Goldberg, 2009). A well-known example of this kind of method is the Expectation-Maximization algorithm, which finds maximum likelihood estimates of the distribution's parameters. Low-density separation methods, in contrast, include S3VM (Semi-supervised Support Vector Machines) or TSVM (Transductive SVM) (Joachims, 1999), which works by moving the class decision boundaries found by SVM away from the unlabeled points. Although first named using the term transductive, the algorithm produces a model which is expected to generalize to new data. Many extensions of TSVM were already proposed, including the works found in (Li and Zhou, 2011, Zhao et al, 2008). Finally, graph-based methods model the instances as a graph, where nodes represent labeled or unlabeled instances and edges reflect the similarity between instances. Many approaches can be followed to solve the SSC problem in this case. A popular one is to propagate information from labeled nodes through the graph to label all nodes (Koutra et al, 2011). Another approach is to address the SSC as a graph max-cut problem (Wang et al, 2013).

Although meta-heuristic methods have already been successfully applied to solve classification problems involving supervised learning (Freitas, 2002, Martens et al, 2011), their use in SSC is still understudied. We identified a few works that dealt with semi-supervised classification and meta-heuristics, including genetic programming and ant colony optimization based methods. The first, named KGP, uses genetic programming combined with self-learning algorithms for transductive learning (Arcanjo et al, 2011). It explores the global search of GP to ensure that classifiers deal with attributes dependencies, while specializing into different regions of the search space. KGP uses a committee of classifiers to predict labels for unlabeled instances, where their votes are weighted according to their confidence in predicting neighbour instances.

The second is based on Aggregation Pheromone System (APSs) and ACO-based algorithm for continuous domains. It was proposed by Halder et al (2010) and extended in (Halder et al, 2013), originating the aforementioned APSSC. In APSSC, each instance is an ant and its position is given by the attributes values. Ants can represent either labeled or unlabeled instances. Each labeled ant spreads pheromone of type c—where c is the ant's instance class value. The unlabeled ants can sense the accumulated pheromone of each class value c in their neighbourhood and they are assigned to the class value with the highest accumulated pheromone. The process is iterative and in iteration $t + 1$ the algorithm uses ants labeled in iteration t to spread pheromone. Another interesting fact is that the class value of a labeled ant can change over time, depending on pheromone concentrations. While we used ants to build classification rules, in their case ants represent instances in the training set, which spread pheromone to their neighbours according to a Gaussian function.

A third approach was proposed by Xu et al (2013), a transductive graph-based ACO approach for semi-supervised classification based on multi-colonies and random walks. The algorithm models each class as a different ant colony and unlabeled instances as food resources. The idea is that an ant colony competes with the others for food resources. This approach has no resemblance to the model proposed here.

The main differences between the algorithm proposed in this paper and those mentioned above are: (i) by using a rule-based method, the models generated are interpretable and can be used to understand how the labeling process evolves; (ii) the diversity of the models ensures an ensemble with high quality, which in turn is able to make more consistent decisions regarding which instances should be labeled; (iii) as in a few other methods previously proposed in the literature, labeling rounds are not independent since the pheromone is preserved from one round to the next.

3 An ACO-Based Approach for Semi-Supervised Learning

This section describes the proposed algorithm, called Ant-Labeler, and it is divided in four parts. We start by describing how the well-known semi-supervised self-learning approach is combined with an ACO (supervised) classification algorithm. We then present the details of the cAnt-Miner$_{\text{PB}}$ algorithm (Otero et al, 2013), which is the algorithm used to extract classification rules. The third part explores distinct ways to preserve diversity during different executions of the ACO

```
   Input : labeled set L, unlabeled set U
   output: ensemble model
 1 M = Initialize pheromones of ACO graph G;
 2 while U ≠ ∅ do
 3 |   RLS = k rule lists learned from L by cAnt-Miner_PB using M for n iterations;
 4 |   foreach instance i in U do
 5 |   |   Classify i using k rule lists in RLS;
 6 |   |   Use a voting procedure to determine final label of i;
 7 |   end
 8 |   subset ← subset of instances selected by the SelectConfidentLabels procedure;
 9 |   Move subset to L;
10 |   M = Update pheromone deposit in paths of top quality rule lists;
11 end
12 return final ensemble model;
```

Fig. 1 High-level pseudocode of the Ant-Labeler algorithm. The algorithm starts with a set of labeled instances L and a set of unlabeled instances U as input. Using a rule lists ensemble model created by an ACO classification algorithm, instances in U are transferred to L, following a self-learning approach. The process is repeated until there are no instances in U—i.e., all unlabeled instances are moved to the labeled set L.

classification algorithm, one aspect of extreme importance in the task being addressed. Finally, the last part discusses how the labeling of unlabeled instances occurs.

3.1 Ant-Labeler: a semi-supervised ACO algorithm

The Ant-Labeler algorithm can be described as an iterative procedure that receives a set of labeled instances L and a set of unlabeled instances U as input, and outputs a set of rule lists RLS. Using the labeled instances L, it generates the construction graph G and associated pheromone matrix M, which will be explored by the ACO classification algorithm to create the rule lists. These rule lists are used as an ensemble of classifiers to classify new unlabeled data. Figure 1 presents the high-level pseudocode of Ant-Labeler. At each labeling round (lines 2 to 11), k rule lists RLS are learned from L (line 3) and used to classify instances in U (lines 4 to 7). The instances in U that are labeled by RLS and satisfy a confidence threshold (line 8) are transferred to L (line 9).

This process is based on the self-learning approach, where instances are automatically moved from the set U to the set L as the learning process goes on. While there are unlabeled instances in U, an ACO classification algorithm is run for n iterations to produce a set of rule lists based on the instances of L (the labeled instances). Each rule list is able to classify all unlabeled instances in U and, because of that, some unlabeled instances may have multiple class values predicted. To deal with the cases where there is conflict (rule lists predicting different class values for the same unlabeled instance), we use a voting procedure to define a unique prediction for each unlabeled instance. This unique class value prediction is also associated with a confidence and number of votes (rule lists agreeing in the class value predicted). If these numbers show enough confidence in the prediction, we transfer the instances from the unlabeled set U to the labeled set L (details are given in Section 3.4). This last step increases the size of the labeled set L, which will be used in the next labeling round to generate new rule lists.

At the end of an iteration, the pheromone matrix is updated to reinforce the pheromone values associated to a set of rule lists (line 10), chosen based on a diversity strategy, so that the next execution of the ACO algorithm starts from the previous *state* (position in the solution space) of the algorithm, instead of starting the search from scratch.

3.2 Learning classification rules

As illustrated in Figure 1, Ant-Labeler uses an ACO classification algorithm to produce a set of rule lists. The set of rule lists is then used as an ensemble of classifiers to classify instances in the unlabeled set U. Ideally, the ACO classification algorithm should be able to produce a set of different rule lists. To this end, Ant-Labeler uses the cAnt-Miner$_{\text{PB}}$ classification algorithm (Otero et al, 2013). In cAnt-Miner$_{\text{PB}}$, each ant in the colony creates a complete rule list. While the whole colony represents a set of potentially different rule lists, only a single rule list is returned at the end. Recall that Ant-Labeler requires multiple executions of the cAnt-Miner$_{\text{PB}}$ classification algorithm and these executions should share the pheromone matrix, preserving the *state* of the search across different iterations of the Ant-Labeler. There is also a need to return a set of rule lists, instead of just one. Therefore, we modified the cAnt-Miner$_{\text{PB}}$ to receive the initial pheromone values as a parameter and to return the set of rule lists created in the last iteration of the algorithm.[1]

The high-level pseudocode of the modified cAnt-Miner$_{\text{PB}}$ procedure is presented in Figure 2. In summary, cAnt-Miner$_{\text{PB}}$ employs a different search strategy than most ACO classification algorithms. Rather than using an ACO procedure to create a single rule, cAnt-Miner$_{\text{PB}}$ uses an ACO procedure to create a complete list of rules instead: an ant starts with the full training set and an empty rule list (lines 6 to 7); then it creates a rule in the form *IF* $<term_1$ *AND* ... *AND* $term_n>$ *THEN* $<$*class value*$>$ by visiting the construction graph G and using its associated pheromone matrix to select terms for the antecedent of the rule. Rules are created by adding one term at a time to the rule antecedent by choosing terms to be added to the current partial rule based on their values of the amount of pheromone (τ) and a problem-dependent heuristic information (η). After a rule is created and pruned, the training instances correctly covered by the rule are removed and the rule is added to the current list of rules. These steps (lines 8 to 14) are repeated until the number of remaining instances in the training set is lower than a predefined threshold (*maximum uncovered*).

The construction graph uses a different representation for nominal and continuous attributes' terms (vertices): for each nominal attribute x_i and value v_{ij} (where x_i is the i-th nominal attribute and v_{ij} is the j-th value belonging to the domain of x_i), a vertex ($x_i = v_{ij}$) is added to the construction graph; for each continuous attribute y_i, a vertex (y_i) is added to the construction graph. Note that continuous attributes vertices do not represent a valid term, since they do not have a relational operator and value associated in the construction graph, in contrast to nominal attributes. The relational operator and a threshold value will be determined when an ant selects a continuous attribute vertex as the next term

[1] Refer to (Otero et al, 2013) for more details on the cAnt-Miner$_{\text{PB}}$ algorithm.

```
   Input  : labeled set L, pheromone matrix M
   Output: set of rule lists RLS
 1 pheromone ← M;
 2 t ← 0;
 3 while t < maximum iterations and not stagnation do
 4 |   RLS ← ∅;
 5 |   for n ← 1 to colony_size do
 6 |   |   instances ← all training instances;
 7 |   |   list_n ← ∅;
 8 |   |   while |instances| > maximum uncovered do
 9 |   |   |   ComputeHeuristicInformation(instances);
10 |   |   |   rule ← CreateRule(instances, pheromone);
11 |   |   |   Prune(rule);
12 |   |   |   instances ← instances − Covered(rule, instances);
13 |   |   |   list_n ← list_n + rule;
14 |   |   end
15 |   |   RLS ← RLS + list_n;
16 |   end
17 |   update_set ← diverse solutions from RLS;
18 |   UpdatePheromones(update_set, pheromone);
19 |   t ← t + 1;
20 end
21 return RLS;
```

Fig. 2 High-level pseudocode of the modified *c*Ant-Miner$_{\mathrm{PB}}$ procedure. The main modifications include: (i) initial pheromone values are given as a parameter (line 1); (ii) the current set of rule lists created in an iteration is saved (lines 5 and 17); (iii) the algorithm returns a set of rule lists created in the last iteration (line 28). Note that the pheromone update during the procedure does not modify the (global) pheromone matrix M—only the Ant-Labeler procedure updates M.

to be added to the rule. *c*Ant-Miner$_{\mathrm{PB}}$ incorporates an entropy-based dynamic discretisation procedure (Otero et al, 2008), which selects the pair (*operator*, *value*) representing the interval associated with the lowest entropy. Only the training cases currently covered by the partial rule are used in this selection, which makes the choice tailored to the current candidate rule being constructed, rather than chosen in a static preprocessing step. Since the discretisation procedure is deterministic, it selects the same (operator, value) pair given the same training cases, avoiding the need to store the pair in the construction graph.

At the end of an iteration (line 16), when all ants have created a rule list, a subset of the rule lists is selected based on a diversity strategy (see Section 3.3) and it is used to update pheromone values (lines 17 to 18). This provides a positive feedback on the terms present in the rules: the higher the pheromone value of a term, the more likely it will be chosen to create a rule. Note that the pheromone update during the *c*Ant-Miner$_{\mathrm{PB}}$ procedure does not modify the (global) pheromone matrix M—only the Ant-Labeler procedure updates M. This iterative process (lines 3 to 20) is repeated until a maximum number of iterations is reached or until the search stagnates. The set of rule lists (i.e., the set containing the rule list of each ant) created in the final iteration is returned as the discovered set of rule lists (line 21).

An interesting aspect of using *c*Ant-Miner$_{\mathrm{PB}}$ in the proposed algorithm is that, by re-using the pheromone matrix M across different executions, we do not start the search from scratch between iterations of the self-training (outer *while* loop

in Figure 1). In a high-level perspective, Ant-Labeler uses a modified *c*Ant-Miner$_{\text{PB}}$ procedure for n iterations to search for a set of rule lists using the labeled set L, labels instances from the unlabeled set U and augments L, and then re-starts *c*Ant-Miner$_{\text{PB}}$ for another t iterations using the extended set L and the pheromone matrix from the previous run. This process is repeated until the unlabeled set U is empty.

3.3 Preserving rule lists diversity

It is usually the case that a single rule list performs well in classifying a subset of instances, while others perform well in a different subset. In order to increase the confidence when labeling instances from the unlabeled set U, Ant-Labeler uses a multi-classifier approach, where different models vote for electing the class value of an instance. The rationale behind the voting process is the same as the one of an ensemble (Rokach, 2010), where different classifiers can work together to classify new instances. One of the underlying principles of ensembles is to have diversity. If all the classifiers return the same class value for all instances, any one of them is sufficient to perform the classification.

The *c*Ant-Miner$_{\text{PB}}$ algorithm as proposed in (Otero et al, 2013) for learning classification rules, however, does not consider the diversity of the generated rule lists. Hence, we adapt and incorporate to the algorithm two distinct strategies for diversity maintenance. The first strategy uses the notion of *fitness sharing* and it is an adaptation of the work proposed by Angus (2009). The second is based on *token competition* and it was originally proposed by Olmo et al (2010).

In the *fitness sharing* strategy, as its name suggests, the fitness of a candidate solution i is penalized according to the number of solutions in the search space that are similar to i. The idea is that there is only a limited amount of resources for different regions (niches) in the search space and that solutions sharing the same niche should also *share their fitness*. The size of a niche is defined by a user-defined radius r_{share}, where solutions within the radius are considered similar. In order to determine if two candidate solutions are within the same radius, the distance between them needs to be calculated.

In the proposed algorithm, each solution is represented by a sequence of paths in a graph—each rule represents a path in the construction graph. A simple metric, which considers the number of shared edges between the two paths normalized by their total size, is used to calculate the similarity between solutions i and j. The distance $d(i, j)$ between solutions i and j is defined as

$$d(i,j) = 1 - \frac{(edges(i) \cap edges(j))}{max(edges(i), edges(j))} \quad , \tag{1}$$

where $edges(i)$ and $edges(j)$ are the set of edges of candidates solutions i and j, respectively.

Given a set of m candidate solutions, the fitness sharing quality (Q') of a solution i corresponds to the quality of the solution $Q(i)$—the number of correct predictions (accuracy) in L—divided by a penalty coefficient $c(i)$, according to its similarity to all other solutions in m. This is given by

$$Q'(i) = \begin{cases} \frac{Q(i)}{c(i)} & , \text{ if } c(i) > 0 \\ Q & , \text{ otherwise} \end{cases} , \quad (2)$$

$$c(i) = \sum_{j=1}^{m} sh(i,j) \ , \quad (3)$$

$$sh(i,j) = \begin{cases} 1 - \left(\frac{d(i,j)}{r_{share}}\right)^p & , \text{ if } d(i,j) < r_{share} \\ 0 & , \text{ otherwise} \end{cases} , \quad (4)$$

where p is a user-defined parameter used to modify the shape of the sharing function and $d(i,j)$ is the distance between candidate solutions i and j.

The second strategy, based on *token competition*, works as follows. The rule lists created in an iteration of the ACO algorithm are ranked according to their quality. Then, we assign to the best rule list (the rule list with the highest rank) every instance it correctly classifies in the labeled training set L. In a next step, we remove these instances from L and repeat this process for the next rule list until there are no more instances in L. At the end of this procedure, the final subset of rule lists is the one that is able to correctly classify all training data. This subset is used to update the pheromone levels, instead of using only the best candidate solution (rule list) created in the iteration.

We have performed experiments using both *fitness sharing* and *token competition* diversity strategies, as presented in Section 4.

3.4 Labeling instances

In semi-supervised approaches, a common way to try to guarantee that instances are correctly classified is to use an ensemble of classifiers and define how their votes will count for a final classification. In the voting procedure defined in Ant-Labeler, each rule list votes for the class value of instances in the unlabeled set U and the procedure computes the majority vote among all rule lists.

Figure 3 describes the *SelectConfidentLabels* procedure, which receives as input the number of votes the class value assigned to an instance received (*votes*), the classification confidence (*confidence*), the unlabeled set U and the maximum number of instances that can be moved from U to L when the voting process is not unanimous ($instances_{max}$). The procedure returns a set of instances to be moved from U to L. Each unlabeled instance in U receives at most $|RLS|$ votes, where $|RLS|$ corresponds to the number of rule lists generated by Ant-Labeler and it is the value initially set to $votes_{min}$.

For each instance i in U, the procedure verifies if the number of votes the majority class value (i.e., the class value that received most votes) received is above a minimum number of votes ($votes_{min}$) and a minimum confidence ($confidence_{min}$) (line 5). The confidence is defined as the ratio among the sum of the confidences of the rules assigning the instance to the majority class value and the sum of the confidences of all rules—the $confidence_{min}$ is initially set to 1. When at least $votes_{min}$ rules agree with the assigned class value, instance i is moved to a temporary set *subset* (line 6).

```
   Input  : votes, confidence, unlabeled set U, instances_max
   Output: set of labeled instances
 1 confidence_min ← 1.0; votes_min ← |RLS|;
 2 while votes_min > 0 do
 3 |   subset ← ∅;
 4 |   forall the i in U do
 5 |   |   if (votes ≥ votes_min) and (confidence ≥ confidence_min) then
 6 |   |   |   Move i from U to subset;
 7 |   |   end
 8 |   end
 9 |   if (|subset| > instances_max) and (votes_min ≠ |RLS|) then
10 |   |   Move |subset| − instances_max least confident instances to U;
11 |   end
12 |   if (subset ≠ ∅) then
13 |   |   return subset;
14 |   end
15 |   if confidence_min > threshold then
16 |   |   confidence_min ← confidence_min − 0.1;
17 |   else
18 |   |   if votes_min > 1 then
19 |   |   |   votes_min ← votes_min − 1;
20 |   |   |   confidence_min ← 1.0;
21 |   |   end
22 |   end
23 end
24 return ∅;
```

Fig. 3 High-level pseudocode of the *SelectConfidentLabels* labeling procedure.

The rationale behind the procedure is to first move to L instances that received the same class unanimously—i.e., all rule lists predict the same class ($votes = |RLS|$)—and that are above a minimum voting confidence threshold. In this case, all instances in the subset are returned by the procedure (line 13). If there are no instances that received the class label unanimously ($|subset| = 0$), an iterative process starts and the conditions for labeling instances are gradually weakened. At each iteration, we reduce the voting confidence by 0.1 until it reaches a minimum acceptable *threshold* (lines 15 to 16). When the minimum voting confidence threshold is reached, the number of votes required for labeling decreases by 1 (line 19) and the minimum voting confidence is reset to 1 (line 20). This iterative process (lines 2 to 23) goes on until a subset of instances is labeled, in which case the subset is returned (line 13), or the minimum number of votes ($votes_{min}$) decreases to zero. In the latter case, an empty set is returned (line 24). Note that the parameter $instances_{max}$ determines how many instances can be labeled in a single round when classifiers are not unanimous in their decisions (lines 9 to 11). This parameter was created to slow down the labeling process, allowing the algorithm more time to adapt. If $instances_{max}$ is set to the number of instances in U, no upper bound is considered.

4 Experimental Results

The experiments with Ant-Labeler were performed in three phases. In the *parameter investigation phase* we compared three versions of Ant-Labeler in 8

datasets from the UCI Repository (Bache and Lichman, 2013). These three versions vary on the strategies implemented for maintaining solution diversity, which is essential to the performance of the proposed algorithm. We then moved to the *test phase*, where we compared the results of Ant-Labeler with four algorithms: S3VM, which is widely used in the literature due to its robust performance across different application domains (Kasabov and Pang, 2003, Tong and Chang, 2001); a version of self-learning with C4.5, which was shown in (Triguero et al, 2015) to be a method to achieve superior results than other wrapped classifiers; APSSC, an ant-colony based semi-supervised algorithm; and the original version of *c*Ant-Miner$_{\mathrm{PB}}$. In this second phase, we used additional 20 UCI datasets, different from those used in the parameter investigation phase.

For both phases, experiments were conducted using different proportions of labeled data, namely 10%, 40% and 70%, using a 5-fold cross validation process. Ant-Labeler was executed 10 times, as it is a stochastic algorithm, and the values reported correspond to the average over this 10 executions. Throughout this section, the results of the algorithms in both phases are compared using a two-step approach. First, we apply the Friedman's test with the null hypothesis $H_0 : \theta_1 = \theta_2 = ... = \theta_n$, where $\theta_1, \theta_2, ..., \theta_n$ are the mean accuracies from each algorithm being tested, and the alternative hypothesis that at least one mean accuracy is different from the others. The alternative hypothesis H_1 occurs if the p-value given by the Friedman's test is less than or equal to 0.05. If H_0 is rejected, we apply Wilcoxon's test with Bonferroni correction as a post-hoc procedure and make pairwise comparisons between the mean accuracies.

In a third phase, we investigated the behaviour of the algorithm when the number of labeled instances drops significantly, reaching 1% of labeled data. As in most real problems very few instances are available, this scenario is one of the most relevant to this work.

4.1 Parameter investigation: diversity maintenance methods

The 8 UCI datasets used in the parameter investigation phase of Ant-Labeler with different diversity maintenance methods are presented in Table 1. For each dataset, we show the dataset's name (first column), the number of instances (second column), number of attributes (third column) and classes (fourth column). In this phase we also tested the impact of preserving pheromone from one labeling round to another and fixed the values of the main ACO parameters. After preliminary experiments, the number of iterations was set to 20, the colony size to 50 and parameters $instances_{max}$ and $threshold$ (see Figure 3) were set to 5 and 0.75, respectively. For the fitness sharing strategy, r_{share} was set to 0.1 and p to 1 (see Eq. 4).

Table 2 shows the results of the average accuracy and standard error (*accuracy* [*standard error*]) obtained by Ant-Labeler using the parameters defined above and no special strategy for maintaining diversity. This table reports the results using 70%, 40% and 10% of labeled data (generated by withdrawing uniformly distributed samples from original data). For each dataset, there is a row labeled C (clear pheronome) and another labeled P (pheromone preservation)—in P, after each labeling round, the pheromone matrix is preserved and in C it is restarted (cleared).

Table 1 Datasets used in the parameter investigation phase.

Dataset	**# Instances**	**# Attributes**	**# Classes**
australian	690	14	2
balance-scale	624	4	3
glass	214	9	6
heart	270	13	2
horse-colic	368	22	2
iris	150	4	3
parkinsons	195	22	2
vertebral-column	310	6	2

Table 2 Predictive accuracy obtained in the parameter investigation phase by Ant-Labeler preserving pheromone levels after labeling instances (P) and resetting the pheromone levels after labeling instances (C), using no diversity strategy. The symbols ▲/▼ indicate that P result is statistically better/worse than C result.

		Percentage of labeled data		
Dataset		70%	40%	10%
australian	C	0.84 [0.02]	0.83 [0.03]	0.76 [0.07]
	P	0.84 [0.03]	0.84 [0.03]	0.81 [0.05]▲
balance-scale	C	0.85 [0.02]	0.81 [0.04]	0.76 [0.05]
	P	0.82 [0.04]▼	0.84 [0.03]▲	0.76 [0.06]
glass	C	0.82 [0.04]	0.80 [0.04]	0.71 [0.05]
	P	0.82 [0.06]	0.81 [0.04]	0.69 [0.09]
heart	C	0.76 [0.06]	0.72 [0.08]	0.70 [0.06]
	P	0.75 [0.04]	0.73 [0.04]	0.68 [0.07]
horse-colic	C	0.81 [0.03]	0.75 [0.05]	0.69 [0.07]
	P	0.81 [0.04]	0.75 [0.04]	0.70 [0.08]
iris	C	0.96 [0.02]	0.96 [0.03]	0.83 [0.09]
	P	0.96 [0.03]	0.97 [0.02]	0.94 [0.05]▲
parkinsons	C	0.86 [0.04]	0.83 [0.04]	0.79 [0.06]
	P	0.87 [0.07]	0.82 [0.07]	0.76 [0.10]▼
vertebral-column	C	0.81 [0.03]	0.82 [0.03]	0.71 [0.07]
	P	0.80 [0.03]	0.81 [0.04]	0.77 [0.05]▲
	P *vs.* C	0 ▲ / 1 ▼	1 ▲ / 0 ▼	3 ▲ / 1 ▼

We first compare the results of the pheromone preservation and then present the results using different diversity strategies. Our aim is to determine the optimal pheromone preservation and diversity strategies. In all tables in this section, ▼ indicates that the method represented by the row in the table achieved results significantly worse than the reference result, while ▲ means the method achieved results significantly better.

Looking at the results in Table 2, we observe that most of the results comparing the strategies of preserving or not the pheromone levels show no evidence of statistical difference. However, in 4 cases preserving the pheromone was statistically better than clearing it (cells marked with ▲) and worse in two cases (cells marked with ▼). As expected, as the number of labeled data decreases, so does the accuracy of Ant-Labeler. However, note that the accuracy does not fall drasti-

Table 3 Predictive accuracy obtained in the parameter investigation phase using 50 ants, 20 iterations and two diversity strategies: TK and FS. The symbols ▲/▼ indicate that the result of the method indicated in the row is statistically better/worse than the method using no diversity strategy (ND), presented in Table 2.

Dataset		Percentage of labeled data 70%	40%	10%
australian	TK	0.85 [0.03]▲	0.85 [0.03]▲	0.78 [0.05]
	FS	0.84 [0.05]	0.82 [0.04]	0.77 [0.09]
balance-scale	TK	0.87 [0.03]▲	0.82 [0.04]	0.72 [0.08]
	FS	0.85 [0.02]▲	0.83 [0.02]	0.74 [0.05]
glass	TK	0.82 [0.03]	0.78 [0.04]▼	0.69 [0.07]
	FS	0.81 [0.05]	0.78 [0.04]▼	0.69 [0.06]
heart	TK	0.74 [0.05]	0.75 [0.05]	0.70 [0.07]
	FS	0.74 [0.07]	0.72 [0.06]	0.70 [0.08]
horse-colic	TK	0.83 [0.03]▲	0.77 [0.06]	0.73 [0.11]
	FS	0.78 [0.05]▼	0.78 [0.06]▲	0.70 [0.06]
iris	TK	0.96 [0.04]	0.95 [0.04]▼	0.92 [0.06]
	FS	0.98 [0.01]	0.94 [0.04]▼	0.94 [0.05]
parkinsons	TK	0.83 [0.08]▼	0.82 [0.08]	0.74 [0.07]
	FS	0.82 [0.06]▼	0.83 [0.06]	0.76 [0.05]
vertebral-column	TK	0.82 [0.04]	0.78 [0.05]▼	0.75 [0.06]
	FS	0.82 [0.05]	0.78 [0.06]▼	0.77 [0.07]
	TK *vs.* ND	3 ▲ / 1 ▼	1 ▲ / 3 ▼	0 ▲ / 0 ▼
	FS *vs.* ND	1 ▲ / 2 ▼	1 ▲ / 3 ▼	0 ▲ / 0 ▼

cally. Consider, for example, the use of 10% of labeled data. The highest drop in accuracy occurred in the dataset *glass*, with the accuracy changing from 82% to 69%, while the number of labeled training instances was reduced from 149 to 21 instances. The highest gains of preserving the pheromone matrix occurred with 10% of labeled data, which is the most interesting scenario. Given these results, we decided to use the pheromone preservation strategy in the next experiments.

Next, we examine the results obtained when using the two diversity strategies: token competition and fitness sharing, from now on referred as TK and FS, respectively. Table 3 shows the results obtained by the two strategies, both preserving the pheromone matrix throughout labeling rounds, and compares with the results obtained by Ant-Labeler when no diversity strategy is used. Note that, overall, the TK method was better than no diversity in 4 cases and worse in other 4; FS, in contrast, was better than no diversity in 2 cases and worse in 5. The 4 cases where TK is worse than no diversity strategy are the same where FS also presents worse results than no diversity strategy. However, in the majority of cases, the tests do not show any evidence of statistical difference between the results. For 10% of labeled data, for example, the performance of the methods is the same. When comparing FS and TK, in turn, TK is better than FS in 3 cases, which are the same ones where it is better than no diversity strategy. Based on these results, we can say that using token competition or no diversity generate the same results, and that fitness sharing has a slight disadvantage.

However, the accuracy of the models is not the only aspect that matters. In semi-supervised methods using a self-training strategy—where the unlabeled data is labeled and then moved to the training set as the method runs—it is also inter-

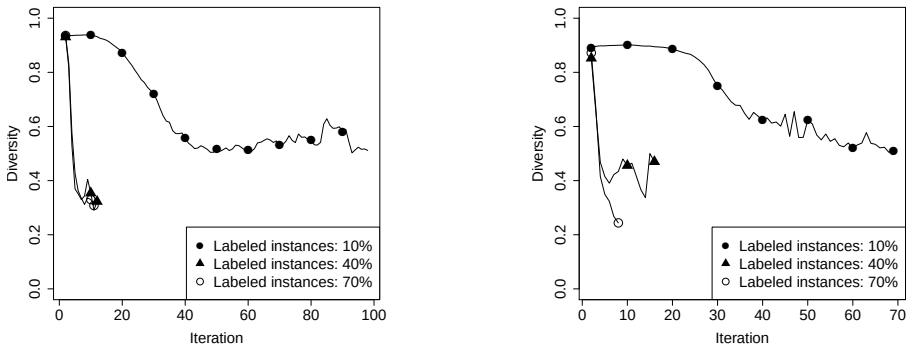
Table 4 Number of labeling rounds needed in the parameter investigation phase for different diversity strategies. The symbols ▲/▼ indicate that the result of the method indicated in the row is statistically better/worse than the method using fitness sharing (FS).

Dataset		Percentage of labeled data 70%	40%	10%
australian	ND	12.12 [2.69]▼	13.00 [3.61]▼	24.54 [14.65]
	TK	45.26 [8.33]▼	87.28 [18.86]▼	47.88 [21.18]
	FS	6.10 [0.96]	7.00 [0.82]	32.70 [22.86]
balance-scale	ND	16.06 [2.66]▼	24.24 [8.40]▼	17.64 [6.51]
	TK	30.56 [5.86]▼	75.96 [9.19]▼	88.82 [42.68]▼
	FS	6.44 [1.10]	9.36 [1.69]	15.42 [4.45]
glass	ND	16.98 [10.33]▼	35.74 [7.35]▼	45.78 [7.66]▼
	TK	12.62 [8.34]	41.04 [10.14]▼	45.02 [14.10]▼
	FS	3.74 [1.91]	6.62 [1.57]	17.32 [10.94]
heart	ND	11.80 [2.72]▼	16.74 [9.06]	50.36 [11.33]
	TK	31.40 [5.66]▼	36.86 [8.07]▼	50.50 [10.29]
	FS	4.92 [1.00]	7.16 [2.67]	39.68 [7.21]
horse-colic	ND	10.30 [2.16]	11.94 [3.23]	53.92 [7.41]
	TK	34.86 [5.52]▼	47.98 [16.46]▼	53.56 [9.70]
	FS	7.26 [1.67]	7.08 [1.73]	53.30 [15.93]
iris	ND	9.38 [1.78]	16.52 [2.15]▼	21.68 [3.86]▼
	TK	9.06 [2.54]	15.04 [3.35]	21.62 [4.53]▼
	FS	7.42 [2.01]	8.84 [4.41]	5.72 [1.75]
parkinsons	ND	17.70 [4.67]	38.92 [3.85]	40.50 [5.59]▼
	TK	20.40 [3.37]	41.06 [5.09]	49.98 [6.50]▼
	FS	20.56 [3.31]	38.00 [5.90]	18.80 [13.59]
vertebral-column	ND	17.52 [2.71]▼	32.80 [8.21]	60.24 [8.96]
	TK	24.86 [4.11]▼	44.84 [6.42]▼	60.38 [10.19]▼
	FS	7.68 [3.18]	18.82 [10.28]	43.28 [13.18]▼
ND *vs.* FS		0 ▲ / 5 ▼	0 ▲ / 4 ▼	0 ▲ / 3 ▼
TK *vs.* FS		0 ▲ / 5 ▼	0 ▲ / 6 ▼	0 ▲ / 4 ▼

esting to consider how many labeling rounds are necessary to move all instances from the unlabeled to the labeled set. Table 4 shows how many rounds were used by different versions of Ant-Labeler. The number of labeling rounds has a direct effect on the efficiency of the algorithm and a large value may cause overfitting.

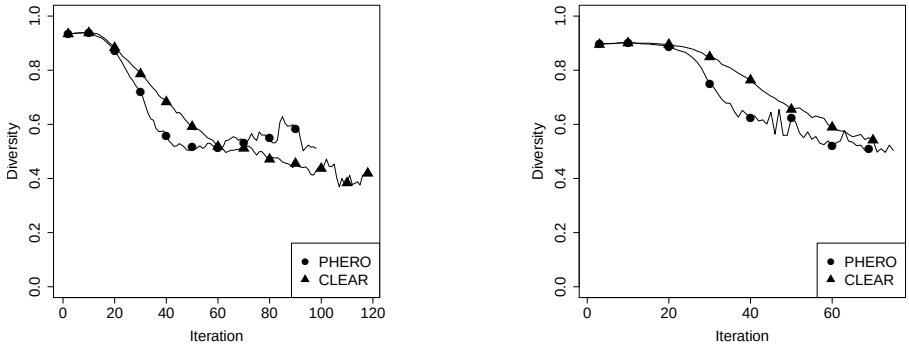
Observe that TK is the method that requires more labeling rounds, followed by ND (no diversity) and FS. As FS has the smallest number of rounds, the statistical test in this case compared FS against ND and TK. Note that, out of 24 cases (8 datasets $\times$ 3 data configurations), ND takes significantly more labeling rounds than FS in 12 and TK requires more rounds than FS in 15. Considering that FS inserts diversity in the colony at a smaller cost than TK, without degrading accuracy in the majority of datasets, we chose to use the fitness sharing as our diversity strategy for the remaining experiments.

Figure 4 illustrates diversity by using the number of different edges among all ants during the model creation process for 2 datasets, namely horse-colic and heart, when using different sizes of labeled training data. Each iteration represents a complete run of ACO, which is equivalent to one labeling round, and the graphs consider pheromone preservation within labeling rounds. As expected, diversity decreases as the number of labeling rounds increases and, consequently, after the



(a) horse-colic (b) heart

Fig. 4 Diversity evolution in terms of the difference in the number of edges (rule antecedents) among models in subsequent labeling rounds using different sizes of labeled training data.



(a) horse-colic (b) heart

Fig. 5 Diversity evolution in terms of the difference in the number of edges (rule antecedents) among models in subsequent labeling rounds when preserving or clearing the pheromone levels with a fitness sharing diversity strategy.

model has been tuned for some time, diversity drops and then stabilizes. Note that the fewer the number of initially labeled instances, the higher the diversity among the first labeling rounds. This may happen because, as the number of labeling rounds increases, so does the number of labeled instances used to generate the model.

Figure 5, in contrast, illustrates the number of different edges among the models generated at each subsequent labeling round when preserving or clearing the pheromone levels with a fitness sharing diversity strategy. Note that, in both cases, the number of different edges follows a similar pattern and decreases over time, reaching around 50% different edges towards the end of the process. However, as expected, the number of different edges decreases faster when the pheromone levels

Table 5 Datasets used in the test phase.

Dataset	# Instances	# Attributes	# Classes
anneal	945	38	6
breast-cancer	304	9	2
breast-tissue	124	9	6
bupa	360	6	2
car	1743	6	4
dermatology	409	34	6
ecoli	352	7	8
german	1029	20	2
hepatitis	183	19	2
house-votes	460	16	2
hungarian	316	13	5
ionosphere	394	34	2
mushroom	8124	22	2
soybean	351	35	19
tic-tac-toe	976	9	2
transfusion	761	4	2
vehicle	873	18	4
wdbc	608	30	2
wine	200	13	3
wpbc	200	32	2

are preserved. The results indicate that, by preserving the pheromone levels, we avoid that the model starts to be built from scratch at each new labeling round and, at the same time, it does not lead to overfitting.

4.2 Test Phase

Given the parameters chosen during the investigation phase (Section 4.1)—Ant-Labeler with fitness sharing and preserving the pheromone levels from one labeling round to the next—we then validated Ant-Labeler in another set of 20 datasets from the UCI Repository, described in Table 5. The results of predictive accuracy achieved by Ant-Labeler are compared with four algorithms: C4.5 with Self-Training (ST), described in Section 2; a Transductive (or Semi-Supervised) Support Vector Machine (S3VM) (Joachims, 1999), considered state-of-the art among semi-supervised methods; ant-based semi-supervised algorithm (APSSC, also described in Section 2); and the original version of *c*Ant-Miner$_{PB}$, which is a supervised algorithm and it will be trained with a small portion of a dataset. Note that, among these methods, ST is the most appropriate for this comparison, as it works based on the same principle of Ant-Labeler.

We used the version of S3VM implemented in SVMLight (Joachims, 1999) and ran the supervised version for 100% of labeled data and the transductive version for the three other cases. In both cases, the SVMLight implementation can handle only binary classification. In order to handle multiclass classification, we used the one-

vs-all approach described in (Hsu and Lin, 2002). All experiments were performed with the RBF kernel and, given the SVM sensitivity to parameters values, a grid-search procedure was performed considering the parameters $c = \{1, 2\}$ and $\gamma = \{1, 0.5, 0.25, 0.125, 0.0625\}$. The results reported here are those obtained with the best sets of parameters, which may vary from one dataset to the other.

APSSC was run using the implementation provided by KEEL (Alcalá-Fdez et al, 2009) with the following parameters: Gaussian spread $\sigma = 0.3$, evaporation rate $\rho = 0.7$, and minimum threshold (for an instance to belong to a class) equal to 0.75. The parameters of cAnt-Miner$_{\text{PB}}$ were those recommended in (Otero et al, 2013): 5 ants, 500 iterations, evaporation rate $\rho = 0.9$, and the maximum number of unconvered instances equal to 10.

The average predictive accuracy and the standard error achieved by the five algorithms for the 20 datasets are shown in Tables 6 and 7 (*accuracy* [*standard error*]). Apart from experiments with 70%, 40% and 10% of labeled data, we also show results with 100% of labeled data. This last experiment corresponds to the supervised approach and the results are interesting to serve as a baseline to the values of accuracy when all data is labeled.[2] We compare the results using the same approach followed in the parameter investigation phase: the Friedman's test followed by the Wilcoxon's test with Bonferroni correction as post-hoc procedure. The symbol following the row representing S3VM, ST, cAnt-Miner$_{\text{PB}}$ and APSSC indicates whether the respective algorithm is statistically better (▲) or worse (▼) than Ant-Labeler.

Table 8 summarizes the number of times Ant-Labeler achieved statistically significantly better or worse results than the algorithms tested. The column 100% is only illustrative, as it represents generating a model using the whole dataset (no unlabeled data) and its only purpose is to serve as a baseline for the results concerning the use of unlabeled data. When using 70% of labeled data, Ant-Labeler was superior to S3VM in 9 datasets and inferior in 4. For ST, Ant-Labeler was superior in 11 and inferior in 5. In comparison with cAnt-Miner$_{\text{PB}}$, Ant-Labeler was better in 10 datasets and worse in 5. In all other cases there was no evidence of statistical difference among the results. Similar numbers were obtained when using 40% and 10% of labeled data. However, as the number of labeled instances decreases, Ant-Labeler becomes more consistent, showing statistically significantly better results than S3VM, ST and cAnt-Miner$_{\text{PB}}$ in up to 12 out of 20 datasets and statistically significantly worse results in up to only 3. Concerning APSSC, for which we have results for only 40% and 10%, Ant-Labeler is statistically significantly better in 12 and 14 datasets, respectively, and statistically significantly worse than APSSC in only 3 datasets for 10% of labeled data; in all other cases there was no evidence of statistical difference among the results.

The best Ant-Labeler results were obtained for the experiments using 40% and 10% of labeled data, where Ant-Labeler was statistically significantly better than all algorithms in at least 10 out of 20 datasets and statistically significantly worse in no more than 3. Overall, out of the 280 comparisons summarized in Table 8—20 datasets and 14 different algorithms/proportions of labeled data pairs—Ant-Labeler was statistically significantly better than the other algorithms in

[2] No results for APSSC with 70% and 100% of labeled data are reported, as the KEEL implementation was not able to generate results for these data configurations.

Table 6 Average predictive accuracy in test datasets (*accuracy* [*standard error*]). The symbols ▲/▼ indicate that the result of the algorithm indicated in the row is statistically better/worse than Ant-Labeler.

		Percentage of labeled data			
		100%	70%	40%	10%
anneal	Ant-Labeler	0.88 [0.02]	0.99 [0.01]	0.97 [0.02]	0.90 [0.03]
	S3VM	0.87 [0.03]▼	0.84 [0.03]▼	0.80 [0.04]▼	0.72 [0.04]▼
	ST	0.92 [0.02]▲	0.90 [0.02]▼	0.86 [0.03]▼	0.80 [0.03]▼
	APSSC	–	–	0.26 [0.06]▼	0.31 [0.08]▼
	*c*Ant-Miner$_{PB}$	0.97 [0.01]▲	0.97 [0.00]▼	0.94 [0.01]▼	0.79 [0.01]▼
breast-cancer	Ant-Labeler	0.72 [0.07]	0.70 [0.05]	0.63 [0.05]	0.65 [0.06]
	S3VM	0.76 [0.06]	0.71 [0.06]	0.71 [0.07]▲	0.65 [0.08]
	ST	0.71 [0.05]	0.70 [0.05]	0.68 [0.04]▲	0.66 [0.09]
	APSSC	–	–	0.60 [0.07]	0.63 [0.09]
	*c*Ant-Miner$_{PB}$	0.74 [0.01]	0.73 [0.01]▲	0.69 [0.03]▲	0.72 [0.00]▲
breast-tissue	Ant-Labeler	0.86 [0.07]	0.88 [0.05]	0.85 [0.06]	0.78 [0.06]
	S3VM	0.67 [0.11]▼	0.58 [0.11]▼	0.48 [0.11]▼	0.41 [0.10]▼
	ST	0.67 [0.12]▼	0.63 [0.11]▼	0.60 [0.13]▼	0.37 [0.15]▼
	APSSC	–	–	0.60 [0.10]▼	0.46 [0.11]▼
	*c*Ant-Miner$_{PB}$	0.65 [0.03]▼	0.63 [0.01]▼	0.48 [0.00]▼	0.18 [0.02]▼
bupa	Ant-Labeler	0.68 [0.07]	0.64 [0.04]	0.61 [0.06]	0.52 [0.08]
	S3VM	0.59 [0.05]▼	0.62 [0.05]	0.59 [0.04]	0.54 [0.09]
	ST	0.64 [0.06]▼	0.64 [0.06]	0.62 [0.06]	0.57 [0.07]▲
	APSSC	–	–	0.57 [0.09]	0.53 [0.09]
	*c*Ant-Miner$_{PB}$	0.66 [0.02]	0.63 [0.01]	0.63 [0.01]	0.56 [0.00]
car	Ant-Labeler	0.82 [0.04]	0.92 [0.02]	0.91 [0.02]	0.86 [0.05]
	S3VM	0.95 [0.02]▲	0.93 [0.01]▲	0.90 [0.02]	0.83 [0.03]▼
	ST	0.92 [0.01]▲	0.89 [0.02]▼	0.86 [0.02]▼	0.78 [0.03]▼
	APSSC	–	–	0.50 [0.03]▼	0.48 [0.04]▼
	*c*Ant-Miner$_{PB}$	0.89 [0.00]▲	0.88 [0.00]▼	0.87 [0.00]▼	0.78 [0.00]▼
dermatology	Ant-Labeler	0.98 [0.01]	0.98 [0.01]	0.96 [0.01]	0.87 [0.03]
	S3VM	0.97 [0.02]	0.96 [0.02]▼	0.96 [0.03]▼	0.78 [0.10]▼
	ST	0.94 [0.03]▼	0.93 [0.04]▼	0.89 [0.06]▼	0.69 [0.09]▼
	APSSC	–	–	0.94 [0.02]▼	0.91 [0.05]▲
	*c*Ant-Miner$_{PB}$	0.91 [0.02]▼	0.88 [0.03]▼	0.81 [0.01]▼	0.61 [0.08]▼
ecoli	Ant-Labeler	0.94 [0.02]	0.94 [0.02]	0.93 [0.02]	0.86 [0.06]
	S3VM	0.86 [0.04]▼	0.85 [0.04]▼	0.84 [0.05]▼	0.78 [0.06]▼
	ST	0.82 [0.05]▼	0.81 [0.04]▼	0.79 [0.05]▼	0.68 [0.08]▼
	APSSC	–	–	0.75 [0.07]▼	0.70 [0.08]▼
	*c*Ant-Miner$_{PB}$	0.80 [0.01]▼	0.78 [0.01]▼	0.76 [0.01]▼	0.53 [0.00]▼
german	Ant-Labeler	0.71 [0.04]	0.73 [0.03]	0.71 [0.04]	0.70 [0.03]
	S3VM	0.70 [0.03]	0.70 [0.03]▼	0.70 [0.03]	0.70 [0.03]
	ST	0.71 [0.03]	0.70 [0.03]▼	0.69 [0.03]▼	0.66 [0.04]▼
	APSSC	–	–	0.69 [0.03]▼	0.65 [0.04]▼
	*c*Ant-Miner$_{PB}$	0.73 [0.00]▲	0.73 [0.01]	0.72 [0.01]	0.68 [0.01]
hepatitis	Ant-Labeler	0.81 [0.09]	0.79 [0.07]	0.82 [0.07]	0.78 [0.10]
	S3VM	0.79 [0.06]	0.79 [0.05]	0.79 [0.07]	0.79 [0.07]
	ST	0.77 [0.07]	0.79 [0.06]	0.79 [0.07]	0.77 [0.09]
	APSSC	–	–	0.81 [0.06]	0.77 [0.10]
	*c*Ant-Miner$_{PB}$	0.79 [0.04]	0.80 [0.04]	0.76 [0.05]▼	0.74 [0.00]
house-votes	Ant-Labeler	0.95 [0.03]	0.94 [0.04]	0.95 [0.04]	0.90 [0.07]
	S3VM	0.96 [0.02]	0.95 [0.02]	0.94 [0.03]	0.88 [0.05]
	ST	0.96 [0.03]▲	0.95 [0.02]	0.94 [0.03]	0.94 [0.04]
	APSSC	–	–	0.92 [0.03]	0.91 [0.03]▼
	*c*Ant-Miner$_{PB}$	0.95 [0.00]	0.94 [0.01]	0.94 [0.00]	0.92 [0.01]

Table 7 Average predictive accuracy in test datasets (*accuracy* [*standard error*]). The symbols ▲/▼ indicate that the result of the algorithm indicated in the row is statistically better/worse than Ant-Labeler.

		Percentage of labeled data			
		100%	70%	40%	10%
hungarian	Ant-Labeler	0.80 [0.02]	0.81 [0.04]	0.81 [0.04]	0.77 [0.07]
	S3VM	0.65 [0.07]▼	0.65 [0.06]▼	0.63 [0.04]▼	0.64 [0.05]▼
	ST	0.66 [0.05]▼	0.64 [0.05]▼	0.63 [0.06]▼	0.60 [0.07]▼
	APSSC	–	–	0.47 [0.13]▼	0.47 [0.14]▼
	*c*Ant-Miner$_{PB}$	0.65 [0.01]▼	0.65 [0.02]▼	0.64 [0.00]▼	0.63 [0.00]▼
ionosphere	Ant-Labeler	0.93 [0.02]	0.92 [0.03]	0.88 [0.04]	0.80 [0.08]
	S3VM	0.95 [0.02]▲	0.94 [0.03]▲	0.94 [0.03]▲	0.86 [0.06]▲
	ST	0.90 [0.03]▼	0.88 [0.04]▼	0.87 [0.04]	0.80 [0.08]
	APSSC	–	–	0.85 [0.05]	0.78 [0.09]
	*c*Ant-Miner$_{PB}$	0.91 [0.01]▼	0.87 [0.03]▼	0.84 [0.04]▼	0.66 [0.03]▼
mushroom	Ant-Labeler	0.99 [0.01]	1.00 [0.00]	1.00 [0.00]	1.00 [0.00]
	S3VM	1.00 [0.00]▲	1.00 [0.00]	0.99 [0.00]▼	0.99 [0.01]▼
	ST	1.00 [0.00]▲	1.00 [0.00]	1.00 [0.00]	0.99 [0.01]▼
	APSSC	–	–	0.99 [0.00]▼	0.99 [0.00]▼
	*c*Ant-Miner$_{PB}$	1.00 [0.00]▲	1.00 [0.00]	1.00 [0.00]	0.99 [0.00]▼
soybean	Ant-Labeler	0.97 [0.01]	0.96 [0.02]	0.95 [0.01]	0.90 [0.01]
	S3VM	0.92 [0.03]▼	0.85 [0.05]▼	0.80 [0.05]▼	0.52 [0.09]▼
	ST	0.83 [0.06]▼	0.80 [0.07]▼	0.68 [0.08]▼	0.38 [0.09]▼
	APSSC	–	–	0.81 [0.06]▼	0.53 [0.08]▼
	*c*Ant-Miner$_{PB}$	0.75 [0.05]▼	0.64 [0.02]▼	0.49 [0.03]▼	0.29 [0.02]▼
tic-tac-toe	Ant-Labeler	0.72 [0.04]	0.77 [0.04]	0.74 [0.04]	0.64 [0.04]
	S3VM	0.99 [0.01]▲	0.97 [0.02]▲	0.73 [0.03]	0.77 [0.04]▲
	ST	0.94 [0.02]▲	0.92 [0.03]▲	0.88 [0.04]▲	0.70 [0.05]▲
	APSSC	–	–	0.74 [0.03]	0.62 [0.04]▼
	*c*Ant-Miner$_{PB}$	0.80 [0.02]▲	0.80 [0.03]▲	0.81 [0.01]▲	0.67 [0.01]▲
transfusion	Ant-Labeler	0.76 [0.05]	0.75 [0.04]	0.74 [0.05]	0.74 [0.03]
	S3VM	0.73 [0.04]▼	0.74 [0.03]	0.73 [0.03]	0.72 [0.04]
	ST	0.78 [0.03]	0.77 [0.03]	0.76 [0.03]	0.75 [0.04]
	APSSC	–	–	0.65 [0.04]▼	0.65 [0.09]▼
	*c*Ant-Miner$_{PB}$	0.78 [0.00]	0.78 [0.00]▲	0.77 [0.00]▲	0.74 [0.00]
vehicle	Ant-Labeler	0.79 [0.02]	0.83 [0.02]	0.80 [0.02]	0.77 [0.03]
	S3VM	0.73 [0.03]▼	0.72 [0.03]▼	0.69 [0.04]▼	0.59 [0.06]▼
	ST	0.72 [0.03]▼	0.71 [0.03]▼	0.68 [0.03]▼	0.57 [0.06]▼
	APSSC	–	–	0.62 [0.04]▼	0.54 [0.05]▼
	*c*Ant-Miner$_{PB}$	0.68 [0.00]▼	0.67 [0.01]▼	0.64 [0.01]▼	0.49 [0.05]▼
wdbc	Ant-Labeler	0.95 [0.02]	0.94 [0.02]	0.93 [0.02]	0.89 [0.02]
	S3VM	0.63 [0.05]▼	0.63 [0.04]▼	0.63 [0.04]▼	0.60 [0.09]▼
	ST	0.94 [0.02]▼	0.93 [0.02]	0.92 [0.03]	0.90 [0.03]
	APSSC	–	–	0.94 [0.02]	0.93 [0.02]▲
	*c*Ant-Miner$_{PB}$	0.94 [0.01]▼	0.94 [0.01]	0.91 [0.01]▼	0.88 [0.00]
wine	Ant-Labeler	0.97 [0.02]	0.95 [0.03]	0.95 [0.04]	0.87 [0.06]
	S3VM	0.97 [0.01]	0.97 [0.02]▲	0.94 [0.03]	0.89 [0.09]
	ST	0.93 [0.04]▼	0.90 [0.05]▼	0.87 [0.06]▼	0.72 [0.13]▼
	APSSC	–	–	0.96 [0.03]	0.94 [0.05]▲
	*c*Ant-Miner$_{PB}$	0.94 [0.02]▼	0.87 [0.02]▼	0.85 [0.03]▼	0.37 [0.00]▼
wpbd	Ant-Labeler	0.76 [0.06]	0.69 [0.07]	0.71 [0.06]	0.73 [0.06]
	S3VM	0.76 [0.06]	0.76 [0.06]▲	0.76 [0.06]▲	0.75 [0.08]
	ST	0.71 [0.08]▼	0.69 [0.07]	0.68 [0.08]	0.65 [0.11]▼
	APSSC	–	–	0.57 [0.11]▼	0.56 [0.08]▼
	*c*Ant-Miner$_{PB}$	0.68 [0.04]▼	0.70 [0.04]	0.66 [0.04]▼	0.76 [0.00]

Table 8 Comparison of Ant-Labeler against the other four algorithms on the 20 UCI datasets. The symbol ▲ indicates that Ant-Labeler was statistically better than the method on the row, while ▼ indicates that Ant-Labeler was statistically worse than the method on the row.

		Percentage of labeled data			
		100%	70%	40%	10%
Ant-Labeler *vs.*	S3VM	9 ▲ / 4 ▼	9 ▲ / 5 ▼	10 ▲ / 2 ▼	11 ▲ / 2 ▼
	ST	11 ▲ / 5 ▼	11 ▲ / 2 ▼	10 ▲ / 2 ▼	12 ▲ / 2 ▼
	APSSC	–	–	12 ▲ / 0 ▼	14 ▲ / 3 ▼
	*c*Ant-Miner$_{\text{PB}}$	10 ▲ / 5 ▼	10 ▲ / 2 ▼	12 ▲ / 3 ▼	11 ▲ / 2 ▼

152 cases (54% of the experiments) and statistically significantly worse in 39 cases (14% of the experiments).

It is interesting to note that Ant-Labeler was statistically significantly better than *c*Ant-Miner$_{\text{PB}}$ in 10 cases when training was performed with 100% of instances, and statistically significantly worse in 5 cases. The fact that the algorithm we used as a wrapper is outperformed by the proposed algorithm when we consider a supervised task (i.e., training with all labeled data) might be strange at first. However, note that Ant-Labeler does work with a diversity scheme and uses an ensemble of rule lists to make predictions. These two features are probably responsible for the improvements observed in the results.

Regarding the behaviour of Ant-Labeler when the number of labeled instances decreases, the fewer labeled instances the lower the accuracy, as expected. However, we observed that in a some datasets—including *german*, *house* and *transfusion*—there were no significant differences among the results using different percentages of labeled data. In *anneal*, a different phenomenon was observed. The accuracies achieved for 10%, 40% and 70% of labeled data increased before decreasing again when reaching 100%. There are two potential explanations for this behaviour. When using 100% of labeled instances, Ant-Labeler runs for 20 iterations and stops—i.e., only one labeling round is performed. For smaller labeled sets, Ant-Labeler runs for many more iterations. Hence, the algorithm has more time to generate an optimal (near optimal) model. The increase in accuracy inline with the increase in the size of the labeled data is likely due to the fact that instances labeled and moved to the labeled set at each labeling round introduced small errors to the final labeling process, therefore, the smaller the unlabeled set, the smaller the potential error introduced in the predictions.

4.3 Lower ratio of labeled data

The results in the previous section have showed that Ant-Labeler outperforms state-of-the-art semi-supervised algorithms in a set of 20 UCI datasets with different ratios of labeled and unlabeled data. This section presents results considering even lower number of labeled data, which correspond to 1% and 5% of the datasets. For this analysis three new datasets, with a higher number of instances than those described in Table 5, were selected from UCI: *nursery* (with 12,960 instances, 8 attributes and 6 classes), *magic* (with 19,020 instances, 10 attributes and 2 classes) and *EEG* (with 14,490 instances, 14 attributes and 2 classes). In order to compare the results obtained in these datasets with those achieved in smaller datasets, we

selected three small datasets used in the experiments in the previous section to be added to this experiment: *anneal*, *transfusion* and *wdbc*. These datasets were selected because they provide a reasonable (although very small) number of labeled instances when considering very low ratios of labeled data (9, 7 and 6 for 1% labeled instances, respectively).

We focus the analysis on semi-supervised algorithms. The results are compared with S3VM and APSCC; however, we replaced the C4.5 with Self-Training by a Tri-Training (Zhou and Li, 2005) version of C4.5 to evaluate a different semi-supervised strategy. Tri-training is based on three learners, and works as follows. Initially, three learners h_1, h_2 and h_3 are trained using the set of labeled data, and then, in pairs, learners are used to label unlabeled data. When h_2 and h_3 agree on the label of an unlabeled instance, the instance is added to a temporary labeled set associated with h_1. Three temporary labeled sets are generated considering the agreement of pairs of classifiers on unlabeled instances. A set of conditions regarding classification error and noise on unlabeled instances are checked, and when these conditions allow, the learners are retrained using these temporary sets plus the original labeled set. This process goes on until h_1, h_2 and h_3 are invariant.

The parameters used for Ant-Labeler, S3VM and APSCC are the same reported in the previous section. For Tri-Training, we used the implementation provided by KEEL (Alcalá-Fdez et al, 2009) with C4.5 and its default parameters.

The average predictive accuracy and the standard error (*accuracy* [*standard error*]) achieved by each algorithm is presented in Table 9—they were calculated over a 10-fold cross-validation. Again, the statistical significance analysis is performed using the Friedman's test followed by the Wilcoxon's test with Bonferroni correction as post-hoc procedure. The symbol following an algorithm's value indicates whether the respective algorithm is statistically significantly better (▲) or worse (▼) than Ant-Labeler.

Table 10 summarizes the number of times Ant-Labeler achieved statistically significantly better or worse results than the algorithms tested. Overall, Ant-Labeler results are positive: it achieved similar or better results than the other semi-supervised algorithms used in the experiments at both 1% and 5% in the majority of the experiments. An exception is the *magic* dataset, where at 1%, Ant-Labeler results are statistically significantly worse than all remaining algorithms; at 5%, S3VM is statistically significantly better than Ant-Labeler. There are two other cases where Ant-Labeler obtained statistically significantly worse results. The first is the *wdbc* dataset at 1%, where the results are worse than those of APSCC, followed by Tri-Training in *nursery* with 5%. Ant-Labeler results are particularly positive in *nursery*, *EEG*, *anneal* and *transfusion*. In the *nursery* dataset, Ant-Labeler results are statistically significantly better than APSCC in both the 1% and 5% cases; in the *EEG*, Ant-Labeler results are statistically significantly better than Tri-Training in both the 1% and 5% cases; in the *anneal* dataset, Ant-Labeler results are statistically significantly better than APSCC at 1% and it is statistically significantly better than all remaining algorithms at 5%; finally, in the *transfusion* dataset, Ant-Labeler results are statistically significantly better than APSCC in both the 1% and 5% cases. Once more, these results show that Ant-Labeler is effective with different ratios of labeled and unlabeled datasets, being competitive to or better than other state-of-the-art semi-supervised algorithms.

Table 9 Predictive accuracy in extreme scenarios of Ant-Labeler, S3VM, APSCC and Tri-Training. The symbols ▲/▼ indicate that the result of the algorithm indicated in the row is statistically better/worse than Ant-Labeler.

		Percentage of labeled data	
		5%	1%
nursery	Ant-Labeler	0.81 [0.03]	0.79 [0.04]
	S3VM	0.81 [0.02]	0.77 [0.04]
	APSCC	0.60 [0.02]▼	0.40 [0.05]▼
	Tri-Training	0.89 [0.01]▲	0.80 [0.09]
magic	Ant-Labeler	0.73 [0.01]	0.70 [0.03]
	S3VM	0.82 [0.01]▲	0.81 [0.01]▲
	APSCC	0.73 [0.02]	0.73 [0.01]▲
	Tri-Training	0.75 [0.05]	0.77 [0.01]▲
EEG	Ant-Labeler	0.57 [0.01]	0.57 [0.02]
	S3VM	0.57 [0.03]	0.54 [0.03]
	APSCC	0.55 [0.04]	0.55 [0.03]
	Tri-Training	0.51 [0.05]▼	0.50 [0.05]▼
anneal	Ant-Labeler	0.88 [0.04]	0.72 [0.07]
	S3VM	0.76 [0.01]▼	0.76 [0.01]
	APSCC	0.27 [0.11]▼	0.39 [0.14]▼
	Tri-Training	0.73 [0.22]▼	0.75 [0.04]
transfusion	Ant-Labeler	0.76 [0.04]	0.69 [0.09]
	S3VM	0.71 [0.06]	0.66 [0.11]
	APSCC	0.61 [0.11]▼	0.57 [0.12]▼
	Tri-Training	0.73 [0.06]	0.61 [0.18]
wdbc	Ant-Labeler	0.90 [0.06]	0.68 [0.10]
	S3VM	0.63 [0.00]▼	0.63 [0.00]
	APSCC	0.92 [0.05]	0.91 [0.05]▲
	Tri-Training	0.88 [0.06]	0.67 [0.10]

Table 10 Comparison of Ant-Labeler against S3VM, APSCC and TriTraining in extreme scenarios. The symbol ▲ indicates that Ant-Labeler was statistically better than the method on the row, while ▼ indicates that Ant-Labeler was statistically worse than the method on the row.

		Percentage of labeled data	
		5%	1%
Ant-Labeler *vs.*	S3VM	2 ▲ / 1 ▼	0 ▲ / 1 ▼
	APSSC	3 ▲ / 0 ▼	3 ▲ / 2 ▼
	Tri-Training	2 ▲ / 1 ▼	1 ▲ / 1 ▼

4.4 Time complexity analysis

Ant-Labeler is an iterative procedure, where each iteration is divided into three parts: (i) a call to cAnt-Miner$_{\text{PB}}$, (ii) the classification of all unlabeled instances through a voting procedure; and (iii) the labeling of the most confident instances. The time complexity of cAnt-Miner$_{\text{PB}}$, previously studied in Otero et al (2013), is $O(t * c * a^2 * d_l^2)$, where t is the number of iterations of the ACO procedure, c is the number of ants, a is the number of attributes and d_l is the number of labeled instances in the dataset. The complexity associated with classifying d_u unlabeled instances—where d_u is the number of unlabeled instances in the dataset—is $O(d_u * c * d_l * a)$, given that for each unlabeled instance we have to verify the label and

accuracy provided by all c rule lists and each rule list is composed of at most d_l rules using at most a terms. The labeling procedure is an iterative procedure, where the constraints are relaxed by decrementing the number of votes from c to 1 and the confidence from 1 to 0.75 by a constant step, in the worst case scenario. Since d_u instances need to be checked at each iteration, the labeling procedure complexity is $O(c * d_u)$. Therefore, the time complexity of an iteration of Ant-Labeler is dominated by cAnt-Miner$_{\text{PB}}$ complexity and the number of iterations of Ant-Labeler is upper-bounded by $O(\frac{d_u}{m})$. Since instances are labeled at each iteration, d_u decreases as d_l increases and $d = d_u + d_l$ (therefore $d \geq d_u$ and $d \geq d_l$), we conclude that the final time complexity of Ant-Labeler is $O(\frac{d_u}{m} * t * c * a^2 * d^2)$. Note that this analysis uses two pessimistic assumptions: it considers that rules have a terms and that the number of rules in a list is equal to d; in practice these values are smaller (Otero et al, 2013).

Regarding the actual execution time, Tri-Training is the fastest overall, followed by Self-Training with C4.5 and APSSC. The execution time of these algorithms is within the same order of magnitude for different dataset sizes. S3VM and Ant-Labeler, in contrast, are slower and within the same order of magnitude—for smaller datasets, Ant-Labeler is faster than S3VM; the bigger the dataset, the more similar the performances of Ant-Labeler and S3VM. The computational time is highly dependent on the number of necessary labeling rounds, which varies significantly from one dataset to another. For example, in smaller datasets (less than 1,000 instances, e.g., tic-tac-toe and anneal) considering 1% of labeled data, Tri-Training took less than 1 second, Ant-Labeler 7 seconds and S3VM 24 seconds; in the larger datasets (more than 20,000 instances, e.g., nursery and magic) considering 1% of labeled instances, Tri-Training took 3 seconds, APSSC 76 seconds, S3VM 4.3 hours and Ant-Labeler 11.3 hours.[3] It is important to emphasize that the reported times are for the training phase. After the models have been generated, the testing time for all algorithms is essentially the same (in the order of seconds). The additional effort spent training the Ant-Labeler is paid off by the higher accuracy obtained during the test phase, as our results illustrate. In addition, ACO algorithms can be easily parallelised since each ant builds and evaluates a candidate solution independently of all the other ants. Therefore, a large speed up could be achieved by running a parallel version of cAnt-Miner$_{\text{PB}}$ in applications where the computational time of Ant-Labeler is considered a significant issue.

5 Conclusions and Future Work

This paper proposed Ant-Labeler, a self-training algorithm that uses as a wrapper the cAnt-Miner$_{\text{PB}}$ algorithm (Otero et al, 2013), produces rule lists and preserves pheromone levels from one labeling round to another. The original version of cAnt-Miner$_{\text{PB}}$ was modified to include two different types of mechanisms to encourage diversity, namely fitness sharing and token competition, as the results of multiple classifiers were considered when moving instances from the unlabeled to the labeled set.

[3] The running times were observed on a Xeon 2.4 GHz machine with 3.5 GB of RAM.

Experiments were performed in three phases. In the parameter investigation phase, eight datasets were used to set parameters and compare different diversity strategies. Although token competition was superior to fitness sharing in three out of eight datasets, it required a much larger number of labeling rounds. Hence, we opted for using the fitness sharing strategy.

In the test phase, we used an additional 20 datasets and compared the results of Ant-Labeler against well-known semi-supervised algorithms, namely the C4.5 algorithm with Self-Training (ST), Transductive (or Semi-Supervised) Support Vector Machines (S3VM), APSSC (Aggregation Pheromone Density based Semi-Supervised Classification) and the original supervised version of *c*Ant-Miner$_{\mathrm{PB}}$. Our results are very promising, showing that Ant-Labeler outperforms ST, S3VM, APSSC and *c*Ant-Miner$_{\mathrm{PB}}$ with statistical significant differences. The comparison against these algorithms is interesting as (i) ST follows the same self-training strategy; (ii) S3VM is considered a state-of-the-art in semi-supervised learning; (iii) APSSC is based on the idea of ant colony algorithms; and (iv) *c*Ant-Miner$_{\mathrm{PB}}$ is the algorithms used as the core of self-training in the proposed approach. The results also showed that the smaller the number of labeled instances, the better Ant-Labeler performance compared to the other algorithms.

Finally, a third set of experiments showed the performance of the algorithm with a reduced number of labeled instances, corresponding to 1% and 5% of the dataset, and considering three additional datasets with a larger number of instances. Again, Ant-Labeler showed to be competitive with the state-of-the-art algorithms.

As future work, we intend to test Ant-Labeler using real-world data with even smaller proportions of labeled data, a common scenario in many applications. Implementing a multi-objective version of the algorithm would also be interesting, as it would allow to take into account the number of instances labeled and the accuracy of the predictions made by the rule lists. Another interesting direction will be to consider further adaptations of *c*Ant-Miner$_{\mathrm{PB}}$ to work with other self-learning like approaches, such as Tri-Training, given the promising results obtained. We also intend to investigate how the algorithm could be adapted to work in graph-based scenarios, since ACO has a natural way of modeling problems as graphs.

Acknowledgements The authors would like to thank the anonymous reviewers and the associate editor for their valuable comments and suggestions. This work was partially supported by the following Brazilian Research Support Agencies: CNPq, FAPEMIG, and CAPES.

References

Alcalá-Fdez J, Sánchez L, García S, del Jesus M, Ventura S, Garrell J, Otero J, Romero C, Bacardit J, Rivas V, Fernández J, Herrera F (2009) KEEL: a software tool to assess evolutionary algorithms for data mining problems. Soft Computing 13(3):307–318

Angus D (2009) Niching for ant colony optimisation. In: Lewis A, Mostaghim S, Randall M (eds) Biologically-Inspired Optimisation Methods, Studies in Computational Intelligence vol 210, Springer, pp 165–188

Arcanjo FL, Pappa GL, Bicalho PV, Meira W Jr, da Silva AS (2011) Semi-supervised genetic programming for classification. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO 2011), ACM, pp 1259–1266

Bache K, Lichman M (2013) UCI machine learning repository. URL http://archive.ics.uci.edu/ml

Blum A, Mitchell T (1998) Combining labeled and unlabeled data with co-training. In: Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT '98), ACM, pp 92–100

Chapelle O, Schölkopf B, Zien A (eds) (2010) Semi-Supervised Learning. MIT Press, 528 pages

Davidson I, Ravi S (2005) Clustering with constraints: Feasibility issues and the k-means algorithm. In: Proceedings of the 2005 SIAM International Conference on Data Mining (SDM05), SIAM, pp 201–211

Freitas AA (2002) Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer, 264 pages

Ginestet C (2009) Semisupervised learning for computational linguistics. Journal of the Royal Statistical Society: Series A (Statistics in Society) 172(3):694–694

Halder A, Ghosh S, Ghosh A (2010) Ant based semi-supervised classification. In: Swarm Intelligence: 7th International Conference (ANTS 2010), Springer, LNCS, vol 6234, pp 376–383

Halder A, Ghosh S, Ghosh A (2013) Aggregation pheromone metaphor for semi-supervised classification. Pattern Recognition 46(8):2239–2248

Hsu C, Lin C (2002) A comparison of methods for multiclass support vector machines. IEEE Transactions on Neural Networks 13(2):415–425

Joachims T (1999) Transductive inference for text classification using support vector machines. In: Proceedings of the 16th International Conference on Machine Learning (ICML '99), Morgan Kaufmann, pp 200–209

Kasabov N, Pang S (2003) Transductive support vector machines and applications in bioinformatics for promoter recognition. In: Proceedings of the 2003 International Conference on Neural Networks and Signal Processing (ICNNSP), IEEE, pp 1–6

Koutra D, Ke TY, Kang U, Chau DH, Pao HKK, Faloutsos C (2011) Unifying guilt-by-association approaches: Theorems and fast algorithms. In: Machine Learning and Knowledge Discovery in Databases: European Conference (ECML PKDD 2011), Springer, LNCS, vol 6912, pp 245–260

Li M, Zhou ZH (2007) Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 37(6):1088–1098

Li YF, Zhou ZH (2011) Towards making unlabeled data never hurt. In: Proceedings of the 28th International Conference on Machine Learning (ICML '11), ACM, pp 1081–1088

Martens D, Baesens B, Fawcett T (2011) Editorial survey: swarm intelligence for data mining. Machine Learning 82(1):1–42

Olmo JL, Luna JM, Romero JR, Ventura S (2010) An automatic programming aco-based algorithm for classification rule mining. In: Trends in Practical Applications of Agents and Multiagent Systems: 8th International Conference on Practical Applications of Agents and Multiagent Systems, Advances in Intelligent and Soft Computing vol 71, Springer, pp 649–656

Otero F, Freitas A, Johnson C (2008) *c*Ant-Miner: An Ant Colony Classification Algorithm to Cope with Continuous Attributes. In: Ant Colony Optimization and Swarm Intelligence: 6th International Conference (ANTS 2008), Springer, LNCS, vol 5217, pp 48–59

Otero F, Freitas A, Johnson C (2013) A New Sequential Covering Strategy for Inducing Classification Rules With Ant Colony Algorithms. IEEE Transactions on Evolutionary Computation 17(1):64–76

Rokach L (2010) Ensemble-based classifiers. Artificial Intelligence Review 33(1):1–39

Tong S, Chang E (2001) Support vector machine active learning for image retrieval. In: Proceedings of the 9th ACM International Conference on Multimedia (MULTIMEDIA '01), ACM, pp 107–118

Triguero I, Garçia S, Herrera F (2015) Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. Knowledge and Information Systems 42(2):245–284

Wang J, Zhao Y, Wu X, Hua XS (2008) Transductive multi-label learning for video concept detection. In: Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval (MIR '08), ACM, pp 298–304

Wang J, Jebara T, Chang SF (2013) Semi-supervised learning using greedy max-cut. Journal of Machine Learning Research 14(1):771–800

Xu X, Lu L, He P, Ma Y, Chen Q, Chen L (2013) Semi-supervised classification with multiple ants maximal spanning tree. In: Proceedings of IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), IEEE, pp 315–320

Zhao B, Wang F, Zhang C (2008) CutS3VM: A Fast Semi-Supervised SVM Algorithm. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), ACM, pp 830–838

Zhou ZH, Li M (2005) Tri-training: Exploiting unlabeled data using three classifiers. IEEE Transactions on Knowledge and Data Engineering 17(11):1529–1541

Zhu X, Goldberg AB (2009) Introduction to Semi-supervised Learning. Morgan & Claypool Publishers, 130 pages