Expert Systems with Applications 41 (2014) 2688–2702

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Applying Ant Colony Optimization to configuring stacking ensembles for data mining

YiJun Chen, Man-Leung Wong *, Haibing Li

Department of Computing and Decision Sciences, Lingnan University, Tuen Mun, Hong Kong

ARTICLE INFO

Keywords:
ACO
Ensemble
Stacking
Metaheuristics
Data mining
Direct marketing

ABSTRACT

An ensemble is a collective decision-making system which applies a strategy to combine the predictions of learned classifiers to generate its prediction of new instances. Early research has proved that ensemble classifiers in most cases can be more accurate than any single component classifier both empirically and theoretically. Though many ensemble approaches are proposed, it is still not an easy task to find a suitable ensemble configuration for a specific dataset. In some early works, the ensemble is selected manually according to the experience of the specialists. Metaheuristic methods can be alternative solutions to find configurations. Ant Colony Optimization (ACO) is one popular approach among metaheuristics. In this work, we propose a new ensemble construction method which applies ACO to the stacking ensemble construction process to generate domain-specific configurations. A number of experiments are performed to compare the proposed approach with some well-known ensemble methods on 18 benchmark data mining datasets. The approach is also applied to learning ensembles for a real-world cost-sensitive data mining problem. The experiment results show that the new approach can generate better stacking ensembles.

1. Introduction

Over years of development, it has become more and more difficult to improve significantly the performance of a single classifier. Recently, there has been growing research interest in the method to combine different classifiers together to achieve better performance. The combining method is referred to as Ensemble. In early research, ensembles were proved empirically and theoretically to perform more accurately than any single component classifier in most cases. If an ensemble is generated by a set of classifiers which are trained from the same learning algorithm, this ensemble is a homogeneous ensemble. If an ensemble is generated by a set of classifiers, which are trained from different learning algorithms, this ensemble is a heterogeneous ensemble (Dietterich, 2000). For example, Bagging (Breiman, 1996) and Boosting (Schapire, 1990) are homogeneous ensembles, while stacking (Wolpert, 1992) is a heterogeneous ensemble.

To generate an ensemble to achieve expected results, two important things should be considered carefully. The first is to introduce enough diversity into the components of an ensemble. The second is to choose a suitable combining method to combine the diverse outputs to a single output (Polikar, 2006). The diversity is the foundation of an ensemble. However, as the diversity increases, the marginal effect decreases after a certain threshold. The memories and computing cost increase significantly while the performance does not improve steadily. For early Bagging and Boosting methods, the diversity is achieved by using the resample strategy. The classifiers included in Bagging are trained with the data subsets, which are randomly sampled from the original dataset. A majority voting scheme is applied as the combining method to make a collective decision. Boosting uses a weighted resample strategy. The weights of all instances are initialized equally. If an instance is misclassified, its weight will be increased. Thus it will be more likely to select the misclassified instances into the next training subset. The diversity generating process stops when the errors are too small. The combining scheme of Boosting is a weighted majority voting. Compared to Bagging and Boosting, stacking does not manipulate the training dataset directly. Instead, an ensemble of classifiers is generated based on two levels. In the base level, multiple classifiers are trained with different learning algorithms. The diversity is introduced because different learning algorithms make different errors in the same dataset. A meta-classifier is applied to generate the final prediction. The meta-classifier is trained with a learning algorithm using a meta-dataset which combines the outputs of base-level classifiers and the real class label.

One problem of stacking is how to obtain an "appropriate" configuration of the base-level classifiers and meta-classifier for each

* Corresponding author. Tel.: +852 26168093.
E-mail addresses: yijunchen@ln.edu.hk (Y. Chen), mlwong@ln.edu.hk (M.-L. Wong), haibingli@ln.edu.hk (H. Li).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.10.063

domain-specific dataset. The number of base-level classifiers and the kinds of learning algorithms are closely related to the diversity. The kind of meta-classifier is also important to the fusion of the base-level classifiers. However, such configuration is still "Black Art" (Wolpert, 1992). Some researchers have proposed different methods to determine the configuration of stacking. Ting and Witten solved two issues about the type of meta-classifier and the kinds of its input attributes (Ting & Witten, 1999). Džeroski and Ženko introduced Multi-Response Model Trees as the meta-classifier (Džeroski & Ženko, 2002). Zheng and Padmanabhan (2007) and Zhu (2010) proposed their Data Envelopment Analysis (DEA) approaches respectively. Ledezma et al. and Ordóñez et al. proposed approaches which search the ensemble configurations using Genetic Algorithms (GAs) (Ledezma, Aler, & Borrajo, 2002; Ordóñez, Ledezma, & Sanchis, 2008).

In this work, we propose an approach using Ant Colony Optimization (ACO) to optimize the stacking configuration. ACO is a meta-heuristic algorithm which is inspired by the foraging behaviour in real ant colonies. Some approaches were proposed recently to apply ACO in data mining. Parpinelli et al. proposed Ant Miner to extract classification rules (Parpinelli, Lopes, & Freitas, 2002). Some approaches apply ACO in feature subset selection tasks (Al-Ani, 2006; Zhang, Chen, & He, 2010).

The rest of this paper is organized as follows. In Section 2, the background of this work, including the related ensemble approaches and the Ant Colony Optimization method, is introduced. In Section 3, the details of our approach are presented. In Section 4, a number of conducted experiments are described to compare our approach with other ensemble methods. Further, the experiment results are presented and discussed in this section. In Section 5, our approach is applied to solve a real-world data mining problem. In the last section, a conclusion is given.

2. Background

2.1. Ensembles

2.1.1. Bagging

Bagging, short for bootstrap aggregating, is considered one of the earliest ensemble scheme (Breiman, 1996). Bagging is intuitive but powerful, especially when the data size is limited. Bagging generates a series of training subsets by random sampling with replacement from the original training set. Then the different classifiers are trained by the same classification algorithm with different training subsets. When a certain number of classifiers are generated, these individuals are combined by the majority voting scheme. Given a testing instance, different outputs will be given from the trained classifiers, and the majority will be considered as the final decision.

A Random Forest is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest (Breiman, 2001). Random Forest can be considered a special type of Bagging.

2.1.2. Boosting

In 1990, Schapire's weak learning framework was proposed (Schapire, 1990). An elegant algorithm, Boosting, which boosts any given weak learners to a strong learner was also provided in this work.

Boosting also applies re-sampling of training data set and majority voting. However, Boosting does not treat all the instances equally, but focuses on the more informative instances which are important to the classification decision. The algorithm generates three classifiers using the same weak learner. The first learner C_1 is trained with a random subset of the training set. The second learner C_2 is trained with a more informative dataset by iteratively flipping a fair coin to decide which instances to add. If a head comes up, some samples are selected from the training set and presented to C_1 until an instance is misclassified by C_1. This instance is added to the training set of C_2. If a tail comes up, a similar process is conducted whereas the first correctly classified instance is selected. The third learner C_3 is trained with the instances which are differently classified by C_1 and C_2 by filtering the whole training set. Finally, a three-way majority voting scheme is used to combine the three classifiers.

AdaBoost is a popular variation of the original Boosting scheme (Freund & Schapire, 1997). AdaBoost maintains a weighted distribution of instances, trains a series of classifiers of the same weak learner with different instances drawn according to the distribution and finally combines the weak learners through a weighted majority voting scheme to generate the final decision. At the beginning of the process, all the instances are initialized with the same weight. For each training iteration, a training subset is drawn from the instances distribution D_t. Then the classification error of this weak learner is calculated and used in changing the weight updating parameter α_t to manipulate the sample distribution to enlarge the probabilities of the currently misclassified instances to be used in the next training iteration. After the weight updating and normalization, the new instances distribution D_{t+1} is generated. α_t is also used as the weight of the weak learner in the weighted majority voting procedure. Some variations of AdaBoost, such as AdaBoost.M1 and AdaBoost.R, have been proposed (Freund & Schapire, 1996, 1997).

2.1.3. Stacking

In the previous ensemble schemes, the individual weak learners are the same. On the other hand, stacking has a two-level structure: level-0 (base-level) classifiers and a level-1 (meta) classifier (Wolpert, 1992). The base-level classifiers are trained with the training set and generate their predictions. Then the meta-classifier is trained with the meta-data to map the outputs of the base-level classifiers to the actual class label. The meta-data could be $((y_i^1, y_i^2, \ldots, y_i^m), y_i)$, where y_i^m means the prediction given by the m^{th} base-level classifier on the i^{th}instances, and y_i is the actual class label. During the process of classifying a new instance, the trained base-level classifiers will give their individual predictions, and the predictions will be considered as the input of the meta-classifier to generate the final decision.

GA-Ensemble was proposed by Ordóñez et al. as an extension of their previous approach (Ordóñez et al., 2008). GA-Ensemble applies a genetic algorithm in searching the configurations according to different datasets without *a priori* assumptions. At the beginning, a set of candidate base-level classifiers is trained to generate a pool of base-level classifiers thus to improve the efficiency without losing accuracy. The candidate set must be encoded in a chromosome, which represents a potential configuration. Binary encoding is used to accompany the canonical GA, where a 0 in the gene means that the classifier of this gene will not be used in the configuration and a 1 means the classifier will be used. The last gene in a chromosome represents two different stacking combining schemes: multi-response model tree or majority voting. This GA search process will iterate for several generations. For each generation, the classification accuracies on validation sets are used as the fitness values to evaluate the chromosomes. Some elite chromosomes will be kept for the next generation and some poor ones will be eliminated. Mutation and crossover operations will be applied to some chromosomes to generate new chromosomes. After all generations are finished, the best chromosome will be chosen as the final configuration.

Todorovski and Džeroski proposed a meta level approach called *Meta Decision Tree* (MDT) in a learning stacking ensemble (Todorovski et al., 2000). The tree is named MLC4.5, indicating a modification from the C4.5 DT. The meta data set for the MDT is composed of the properties which reflect the confidence of the base classifiers instead of the probability distribution or the simple class label. Such properties are the entropy, the maximum probability and the fraction of training samples. The tree uses the class labels in the leaf nodes only. The leaves of MDT specify which base classifiers should be used instead of predicting the class label directly.

Zhu proposed the DEA-Stacking approach which applies *data envelopment analysis* (DEA) to find optimal stackings (Zhu, 2010). DEA is a linear programming methodology to measure the efficiency of multiple *decision-making unit*s (DMUs) when the production process presents a structure of multiple inputs and outputs (Ramanathan, 2003). DEA-Stacking considers the classifiers as the DMUs in DEA. In this approach, the inputs and outputs of a DMU are extracted from the confusion matrix of the model. At the first stage, the classifiers are trained and evaluated. The DEA models take the number of *false positive* and *false negative* as the inputs and the number of *true positive* and *true negative* as the outputs of the DMUs to find out the efficient one(s) to be the base classifier(s). Several classifiers with an efficiency of 1 will be selected as the base classifiers in stacking. At the second stage, the meta classifier is also selected by the DEA models. The stackings with each learning algorithm in the set combining the selected base classifier(s) is treated as the DMUs to find the most efficient as the final configuration.

2.2. Ant Colony Optimizations (ACO)

The idea of ACO is inspired by the collective behaviour of real ant colonies, which enables the ants to find the shortest path from their nest to the food source (Dorigo & Stützle, 2004). Each ant has limited intelligence to find the best or shortest path; however it can use indirect communication to communicate with other ants. When an ant is walking, it deposits a chemical material called a *pheromone* on the ground. The ants can smell the pheromone and use it to find their way. The ants choose their path to walk in a probabilistic manner, so that the paths with stronger pheromone concentrations will be chosen with larger probabilities. If the pheromone is absent, the ants will randomly choose a path to walk. After a period, the shorter path is chosen more frequently, which means more ants walk this way and the pheromone accumulates faster. The accumulation of pheromone attracts more ants to choose this path. Double bridge experiments have proven this behaviour system (Goss, Aron, Deneubourg, & Pasteels, 1989). If a path is not chosen by the ants, the pheromone will evaporate. The accumulation of pheromone is positive feedback to encourage the ants to choose the shortest path. However, some ants may select paths with less pheromone, but this situation is very important for the ants to get rid of the local shortest path to find another way to achieve the global shortest path. If the new path is shorter than the current path, the pheromone will accumulate and attract more ants to walk this way. Then the optimal path will be changed to this one. In conclusion, although the ability of ants is limited, the optimal shortest path is likely to be achieved by the collective behaviour of ants through this indirect communication. Some works have proved the convergence of ACO with rigid mathematical reasoning (Gutjahr, 2002).

2.3. Application of ACO in data mining

The ACO approach is widely used in many aspects of data mining. In data mining tasks, feature subset selection is an important step to reduce the redundant features and therefore build more precise and efficient models. Al-Ani presented a feature searching procedure based on ACO which utilizes both local importance of features and overall performance of feature subsets (Al-Ani, 2006). This approach is applied to speech segment and texture classification problems and outperforms the GA-based approaches. Sivagaminathan and Ramakrishnan proposed an approach which is a hybrid method based on ACO and Artificial Neural Networks (ANNs) to address feature selection. The ANNs are employed as the classification models, which produce the error corresponding to each subset (selected by ants) in order to find the optimal solution set, whereas the ACO is used for evaluating the process to determine the final subset. A heuristic value calculation is applied in the approach to reduce the set of available features (Sivagaminathan & Ramakrishnan, 2007).

In relation to rule-based classification problems, Parpinelli et al. proposed an algorithm called Ant-Miner (Ant Colony-based Data Miner) to extract classification rules from a dataset (Parpinelli et al., 2002). Each ant in the colony represents a classification rule such as $IF \langle term_1 \rangle AND \langle term_2 \rangle AND \cdots \langle term_n \rangle THEN \langle class \rangle$, where $term_i$ is generated in the preliminary test and represents the trails in the ground, where the ants live. For each iteration, the pheromone of the trail, which is adopted by the "ants" will increase. At the end of each iteration, the best "ant" is added to a list which contains all the classification rules discovered by Ant-Miner. The authors claimed that Ant-Miner could discover more rules and perform better than C4.5, a well-known approach for the same task.

Liu et al. demonstrated a variant of Ant-Miner called Ant-Miner3, which has better performance than Ant-Miner applied in the study by Parpinelli et al. Two main improvements are included in this study. First, Ant-Miner3 has a flexible stochastic component to balance the exploitation and exploration process. By using this mechanism, the generated models are more accurate. Second, a different pheromone update rule is designed in order to cause future ants to make better decisions. There are some drawbacks of this approach. It requires the setting of a number of parameters to achieve desired performance and it has not been evaluated on a real-world data mining problem (Liu, Abbas, & McKay, 2003). Wang and Feng proposed an improved ant colony algorithm for mining classification rules called ACO-Miner (Wang & Feng, 2005). Compared to the classical Ant-Miner, ACO-Miner is able to produce simple state transition rules and self-adaptive pheromone updating rules. In addition, ACO-Miner applies a new heuristic function to avoid convergence to a single constructed rule too quickly; thus it can generate better predictive rules in several benchmark data sets. However, it has the same drawbacks of Ant-Miner3.

By using ACO in data mining classification problems, rule pruner is a technique that removes uncorrelated variables in the antecedent part of a classification rule. Chan and Freitas proposed a new hybrid rule pruner for Ant-Miner (Chan & Freitas, 2006). The objective of the new rule pruner is to shorten the classification rules in order to provide more compact knowledge to support decision-making. Ant-Miner with the new rule pruner has lower accuracy than the one with the original rule pruner on several data sets. However, the comprehensibility of the generated rules is significantly improved.

ACO has also been applied to learn knowledge represented in other representations. Campos et al. used ACO to learn Bayesian Networks (Campos, Fernández-Luna, Gámez, & Puerta, 2002). A Bayesian Network (BN) is a probabilistic graphical model comprising nodes and directed edges in the form of directed acyclic graphs. In the study, ACO is used to guide a scoring-based search process, as ACO allows the searching to exploit heuristic knowledge with simple but efficient forms of cooperation between independent

agents (ants). Pinto et al. proposed two ACO-based approaches to learn the structure of a BN (Pinto, Nägele, Dejori, Runkler, & Sousa, 2009).

Although stacking is a well-known heterogeneous ensemble technique, it is still a difficult problem to configure an optimal stacking for a specific dataset. From several applications of ACO in data mining problems, ACO performs well but has not been employed for handling the stacking configuration problem. Thus we develop an integrated approach called ACO-Stacking for the problem. Moreover, the previous stacking ensemble learning techniques consider only the global performance of the stacking ensemble while ignoring the local performance information of individual base-level classifiers. In this research, different kinds of local information are studied to improve the performance of ACO-Stacking. Furthermore, we evaluate and compare the performance of ACO-Stacking with many existing data mining methods on a number of benchmark and real-world problems. We show that ACO-Stacking is a promising approach for handling problems in data mining.

3. ACO-Stacking approach

Considering the outstanding performance of ACO in different applications, we extend the application of ACO in stacking configuration optimization. In an ACO-Stacking construction task, a set of base-level classifier candidates and a set of meta-classifier candidates are given as well as the training sets, the validation sets and the testing set. The base-level classifiers in the set are taken as the "paths" to be selected by the ants. For each iteration, an ant tries to select a path in its route to achieve better performance. Each ant is assigned a certain meta-classifier to combine with the selected "paths" into the "path" package of the ant. A stacking model is configured with the base-level classifiers ("paths" of the ant) and the meta-classifier. This stacking is then trained with the training set(s) and validated with the validation set(s). If the new "path" package is better than the existing one, it will replace the existing package. Otherwise, the existing "path" package of this ant does not change. At the end, the configuration (the "path" package) of the best ant will be the final configuration of the approach. Finally this configuration is tested by using the test set. The above process is given in Fig. 1. In the following subsection, the algorithm framework of ACO-Stacking is discussed.

3.1. ACO-Stacking algorithm framework

Before discussing the algorithm framework, some notations that will be used in the algorithm description are given as follows:

- C is the pool of base-level classifier candidates. It contains m classifiers generated from the learning algorithms, $C = \{c_1, \cdots, c_m\}$.
- k artificial ants in the colony, each ant carries a meta combining method and represents a stacking configuration.
- μ_i: the pheromone associated with the c_i in C.
- η_i: the local information of c_i, which is a metric to evaluate the ability of c_i.
- S_j: the stacking configuration constructed by the j^{th} ant, $j \leqslant k$.
- α_S: the evaluation criterion of the stacking S. Here the classification accuracy of S is used as α_S.
- τ: the evaporation rate and $\tau \in [0, 1]$.
- L: the maximum iteration number.

At the beginning of ACO-Stacking, a set C containing base-level classifier candidates is given. Some pre-tests are conducted to gather the local information of the base-level classifiers. Here, the term "local information" is used to represent the metric to evaluate the individual classification performances of the base-level classifiers. Moreover, the pheromone μ_i of each base-level classifier c_i is initialized to a small positive number for the probability selection process. The pheromone will increase or decrease during the ACO searching process. Each ant in the colony is assigned a learning algorithm as its meta combining scheme to generate the meta-classifier. Thus an ant represents a stacking configuration. The number of ants is usually set to be multiples of the meta combining schemes. After all the settings and configurations are prepared, the main process of the ACO heuristic begins. Like other ACO approaches, ACO-Stacking will execute several iterations. In the first iteration, each ant is given a base-level classifier randomly and the accuracy α_{S_i} of this configuration is calculated from an independent validation set. In the following iterations, when the j^{th} ant begins its configuration searching, it selects a classifier 'c' from the pool C which does not exist in its current configuration S_j using roulette wheel selection. The probabilities of classifiers are normalized and mapped to the fractions of the roulette. The larger the fraction in the roulette, the larger the possibility that this classifier will be selected. The probability p_i of the classifier c_i to be selected by the j^{th} ant is given by Eq. (1).

$$p_i = \begin{cases} \frac{q_i}{\sum_{t=1, c_t \notin S_j}^{m} q_t} & \text{if } c_i \notin S_j, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

where q_i refers to the metric of the i^{th} classifier to be mapped in the roulette. The q_i could be generated by using the pheromone of the i^{th} classifier only or the product of its pheromone and its local information. Suppose that c_i is selected then a new configuration S_j' of this ant is generated where $S_j' = S_j \cup c_i$. Then S_j' is tested by the same validation set. If the performance of S_j' is better than S_j, it will replace S_j and then the ant continues to find another base-level

1. **Input:**
   - Datasets: Training Sets, Validation Sets
   - Learning algorithms for base-level classifiers and meta-classifiers
2. **Generating stacking by ACO-Stacking Framework:**
   - Applying ACO to search stacking configurations
   - Training and validating the stacking
   - Output the best Ant as the final configuration
3. **Testing:**
   - Applying the final configuration on the Testing set

Fig. 1. General process of ACO-Stacking.

classifier to add to the new S_j according to the same strategy to generate a new stacking. If S_j' cannot improve the accuracy of S_j, this ant keeps its current stacking configuration and stops its search in the iteration. Then the next ant in the colony starts its searching, until all the k ants finish their search. During the ants' searching process, once a classifier c_i is chosen to be added to any S_j to generate a better configuration S_j', the pheromone of c_i will accumulate, thus enhancing the probability of this classifier being selected by the other ants. The improvement of accuracy from S_j to S_j' is used to update the pheromone of c_i. The update rule is given in Eq. (2).

$$\mu_i' = \mu_i * (1-\tau) + CC * \mu_i * \frac{\alpha_{S_j'} - \alpha_{S_j}}{\alpha_{S_j}} \tag{2}$$

where CC refers to a constant number. The evaporation rate τ and CC are introduced to adjust the emphasis of historical knowledge and the current knowledge. The greater τ is, the less historical information will be used. The greater CC is, the more important current knowledge is considered.

During the ACO metaheuristic, the pheromone of the strong candidates will accumulate and the pheromone of the poor ones will vanish. After all iterations finish, the best configuration S_{best} among all k ants will be chosen as the final stacking configuration.

3.2. Local information

In the previous subsection, local information is mentioned as the metric to evaluate the abilities of the base-level classifiers. In this subsection, we focus on the discussion of the adoption of local information. Firstly, consider the situation where an approach does not implement local information of the classifiers. In such a case, only the pheromone can affect the probabilities of selecting base-level classifiers. In the previous discussion, the pheromone represents how the classifier improves the global performance. However, in the early iterations of the approach, the selection of the base-level classifiers is quite random. Some "weak" classifiers may be selected in the early iterations and acquire pheromone accumulation. Therefore the "weak" classifiers will get larger values of pheromone and are more likely to be selected in the following iterations than some "strong" ones which have no pheromone accumulation. Such situations cause increased execution time to generate a promising configuration, as some "weak" classifiers are selected and discarded again and again. To solve this problem, selecting some "strong" classifiers in the early iterations is quite important. Local information of the classifiers is therefore used to identify the "strong" and "weak" classifiers. Local information is also called heuristic information and local importance (Pinto et al., 2009; Al-Ani, 2006).

The accuracy is the global performance evaluation of the stackings constructed by ACO-Stacking. With the aim of optimizing the data fusion of ensembles which can generate better decision boundaries from the different base-level classifiers, one intuitive option is to use the base-level classifiers which already have good decision boundaries. An illustration is given in Fig. 2 of two boundaries which separate two kinds of data objects. In the simple example, each decision boundary makes mistakes when separating the two categories of objects. The dotted line mistakes two triangles as the circles while the solid line mistakes one triangle as the circle. To adopt different parts of the lines will either improve or undermine the separation.

A pre-test of each c_i on the whole training set is conducted to gather measures of the local information. The measure Precision (Pr) could be suitable as the local information used to fuse the decision boundaries of different classifiers. Precision is the measure used to evaluate the percentage of correctly classified positive instances in the instances which are classified as positive by a classifier. We set the class which takes up the largest percentage in the dataset as the positive class in the measure of precision. The higher precision indicates fewer mistakes in the boundary of this classifier. In other words, this classifier is "stronger".

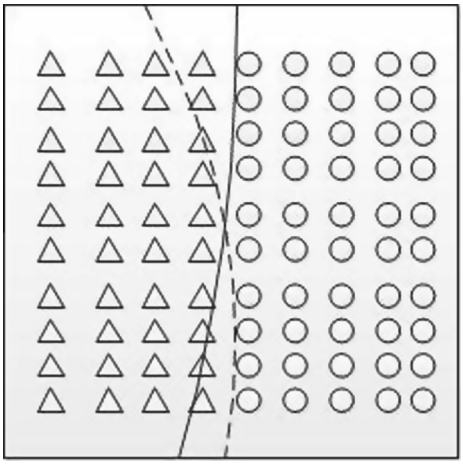

Fig. 2. An illustration of decision boundary.

Although the use of Precision as the local information improves the performance of the approach, there are some limitations. Sometimes the classifiers with greater precision may have similar decision boundaries for certain difficult problems. Thus including these classifiers only overlaps their boundaries and cannot improve performance significantly. Some classifiers may have smaller precision values, but their decision boundaries are quite different from those classifiers with high precision values. In such cases, selecting these classifiers may improve the overall performance. We materialized the differences in decision boundaries into the correlative differences of the predictions given by different classifiers on the training set. Some previous approaches inspired us to develop the measure of the correlative differences of classifiers (Merz, 1999; Lu, Wu, Zhu, & Bongard, 2010). Merz considered the usage of correspondence analysis in combining classifiers (Merz, 1999). Lu et al. proposed an ensemble pruning approach via individual diversity contribution ordering (Lu et al., 2010).

Given the pre-test set, each classifier runs a ten-fold cross validation. Both the training set and testing set are the same for each classifier in the same fold which ensures that all the classifiers are treated equally. When the pre-test is finished, all the predictions of the classifiers on the same instance in the set are collected. The difference: $D_{i,j}$ between C_i and C_j is the number of instances when they make different predictions. The difference matrix of the classifiers is:

$$\begin{vmatrix} 0 & D_{1,2} & \cdots & D_{1,n} \\ D_{2,1} & 0 & \cdots & D_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ D_{n,1} & \cdots & D_{n-1,n} & 0 \end{vmatrix}$$

In the matrix, $D_{i,j} = D_{j,i}$ and the larger $D_{i,j}$, the larger differences between C_i and C_j.

The local information η_i of the i^{th} classifier is calculated from the items in matrix by Eq. (3):

$$\eta_i = \frac{\sum_{t=1, c_t \in S_j}^{m} D_{i,t}}{k} \tag{3}$$

where k equals the number of classifiers in the current configuration S_j. According to the equation, the larger the average difference of the candidate classifier c_i from the classifiers in S_j, the greater the

difference between the decision boundary of c_i and the decision boundaries of the current stacking S_j. Thus if this c_i is selected, the data fusion may be improved.

Up to this point, the local information used in our approach and its importance in improving the performance of ensembles have been discussed. However, the time taken to apply local information still requires consideration. In order to generate promising ensembles, it is necessary to introduce enough diversity into the components of an ensemble (Polikar, 2006). Depending on local information to select base-level classifiers will overemphasize the "strong" classifiers and may reduce diversity in the stacking. Thus, ACO-Stacking uses the pheromone alone in the roulette selection function (1) in the first half of the iterations and then uses the product of pheromone and local information in the selection function in the following iterations.

3.3. Different versions of ACO-Stacking

We have implemented three different versions of ACO-Stacking, which are called ACO-S1, ACO-S2, and ACO-S3. They are descibed in the following subsections.

3.3.1. ACO-S1

In this version, the meta learning algorithm is set to the C4.5 Decision Tree (DT) (Quinlan, 1993) so there is only one meta-classifier. Moreover, local information is not implemented in this version to guide the searching process. Thus the update rule is given in Eq. (1) with $q_i = \mu_i$. The approach is more stochastic than the other versions in exploring many possible combinations of base-level classifiers with the same meta-classifier. Thus, more iterations are needed to find the optimal solution. Since all ants use the same meta-classifier, only the combinations of base-level classifiers can affect the performance of the ants. The pseudo code of ACO-S1 is presented in Fig. 3.

3.3.2. ACO-S2

ACO-S2 has three main features. Firstly, the meta-classifiers of the ants can be built by assigning a learning algorithm from a set. Each learning algorithm is treated equally and is assigned to the ants by a uniform distribution. By using more learning algorithms to learn meta-classifiers, the approach can adapt to the characteristics of the datasets in different domains.

Secondly, a pool of base-level classifiers is generated to accelerate the execution speed. The metaheuristic methods usually suffer from a long execution time. In the stacking training process, the base-level classifiers should be trained and the outputs are used to generate the meta training set for the training of the meta-classifiers. If many stackings will be generated and trained, the same base-level classifiers may be trained several times using the same training sets, which is very costly. To improve the efficiency of the approach, the pool of classifiers proposed in GA-Ensemble is generated *a priori* in our approach (Ordóñez et al., 2008). Consider the stacking training process, where the training set is split into ten partitions. One partition is separated to be the validation partition and the other nine partitions are used to train the classifiers until all the partitions are validated. The outputs of each validation partition of this learning algorithm are joined together. For each base-level learning algorithm, this process is conducted. Next, all the prediction results of the base-level classifiers on each training instance are stored in a pool. To generate a stacking ensemble, only the meta-classifier needs to be trained. The meta training set is the conjunction of the predictions of the selected base-level classifiers in the pool.

Thirdly, local information is introduced. Before ACO-S2 starts to search for the configurations, the pool of base-level classifiers is generated. Then a series of pre-tests is conducted to find the suitable metric to act as the local information. In ACO-S2, the precisions of the base-level classifiers are used as the local information. For each classifier c_i in the pool, its local information η_i is initialized and the pheromone μ_i is initialized with a small positive value. Once the local information of the classifier is set, it cannot be changed during the searching process. Thus the probability p_i of selecting the classifier c_i is changed to Eq. (4).

$$p_i = \begin{cases} \frac{\mu_i * \eta_i}{\sum_{t=1, c_t \notin S_j}^{m} \mu_t * \eta_t} & \text{if } c_i \notin S_j, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

where η_i is the precision of c_i. The pseudo code of ACO-S2 is presented in Fig. 4.

3.3.3. ACO-S3

In this version, we use the correlative differences of different base-level classifiers on the training set as the local information. The other components of ACO-S3 are the same as those of ACO-S2.

1. For i from 1 to m, initialize the pheromone μ_i of C_i in C; initial L to 0
2. While the maximal iteration L does not reach
   (a) For j from 1 to k, the j^{th} ant begins its searching
   - Initialize its configuration $S_j = \emptyset$
   - Initialize the current best configuration $S' = \emptyset$
   - Set the flag of adding a new classifier to *true*
   - While the flag equals to *true*
     - Using roulette wheel technique to select a $c_i \notin S_j$ to generate a new configuration: S'_j and $S'_j = S_j \cup \{c'\}$
     - IF current best configuration $S_j = \emptyset$, THEN Set $S' = S'_j, S_j = S'_j$
     - ELSE
       * Apply S'_j to train an ensemble on the training set
       * Evaluate the accuracy of the ensemble on an independent data set
       * Compare the accuracy of S'_j to that of S'
         · IF S'_j is superior, THEN update the pheromone μ_i of c_i and,$S' = S'_j, S_j = S'_j$
         · ELSE, set the flag of adding a new classifier to $false$

   (b) Evaporation occurs when an iteration finishes.
   (c) $L = L + 1$.
3. Using the same searching process of an ant to generate the final Stacking configuration

Fig. 3. The algorithm of ACO-S1.

1. Generate the pool of base-level classifiers
2. Initialize settings: τ, η, μ, L and k
3. While the maximal iteration L does not reach
   (a) For j from 1 to k, the j^{th} ant begins its searching
   - Initialize the S_j by given a meta combining method and randomly select a c from C
   - Calculate α_{S_j}
   - Set the flag $search\_next = true$
   - While $search\_next = true$
     Select a c from C according to the pheromone distribution and local information
     If no c can be selected
       set $search\_next = false$
     Else
     – Add c to generate new configuration S'_j
     – Calculate $\alpha_{S'_j}$
       If $\alpha_{S'_j} > \alpha_{S_j}$
         $S_j = S'_j$
         Update the pheromone of c
       Else
         $search\_next = false$

   (b) Evaporation works after an iteration ends
   (c) $L++$
4. Output the best configuration S_{best} in the final iteration as the final configuration of ACO-Stacking
5. Test S_{best} in the independent testing set

Fig. 4. The algorithm of ACO-S2.

3.4. Differences between ACO-Stacking and GA-based approaches

In Section 2, we briefly introduced GA-Ensemble, a GA-based stacking configuration search approach. Though ACO-Stacking and GA-Ensemble are all hybrids of metaheuristics with stacking ensembles, there are some differences between them. During the ACO searching process, the ants use the pheromone as an indirect communication method, while during the GA searching process, the chromosomes cannot communicate with each other. The crossover points and the mutation points are selected randomly, so some well-performed stackings may generate poor offspring. The searching process in GA-Ensemble is therefore more stochastic than that in ACO-Stacking.

To escape from sticking in local minima, the weak ants in ACO-Stacking will not be eliminated but simply stop searching in this iteration. In GA-Ensemble, the last *n* cull chromosomes will be eliminated and the top *m* elite chromosomes will be kept for the next generation. The mutation and crossover operations on the elite chromosomes are used to escape from local minima. However, there are no strategies to stop the same weak stackings from being generated again in the following generations, which will be expensive because these weak stackings have to be evaluated again.

ACO-Stacking is more flexible than GA-Ensemble in meta-classifiers selection. GA-Ensemble can only select either a multiple-response model tree or a majority voting scheme as the meta-classifiers, while ACO-Stacking can select the meta-classifiers from a set of learning algorithms. If the number of base-level classifier candidates in ACO-Stacking is the same as the number of genes representing classifier candidates in GA-Ensemble, the search space of ACO-Stacking is larger than that of GA-Ensemble. Furthermore, if the best meta-classifier for a certain dataset is neither the majority voting scheme nor a model tree, GA-Ensemble is unable to find it.

4. Experiments and results

To compare the performance of ACO-Stacking approaches and the other well-known ensemble approaches, experiments are conducted in the Waikato Environment for Knowledge Analysis - WEKA (Hall et al., 2009). This environment implements some well-known ensemble methods and different machine learning algorithms to generate classifiers.

To make the experiment results more robust, a ten-fold cross validation scheme is used for each data set during the experiments. A dataset is randomly split into 10 mutually exclusive and exhaustive folds. Each time, one fold is selected as the test set and the other nine folds are combined together as the training set. The learning approaches use the training set to train the models and use the test set to evaluate the models. The average of evaluation results is given.

Eighteen data mining datasets in different domains from the UCI machine learning repository (Frank & Asuncion, 2010) are used to compare different approaches. The names and some properties of these datasets are summarized in Table 1. During the experiment, all the datasets are kept the same as those in the repository, without any preprocessing or feature selection.

4.1. Learning algorithms and experiment settings

In order to obtain optimal configurations of stacking, ten different learning algorithms in WEKA are used as the base-level classifier candidates. The ten algorithms can be categorized into different kinds of methods, thus making them as diverse as possible when generating classifiers.

- Naïve Bayes (NB) (John & Langley, 1995) learns classifiers by the naive probabilistic estimator based on the Bayes' theorem.
- Logistic Regresion (LeCessie & VanHouwelingen, 1992) builds a multinomial Logistic Regression model to make predictions.
- IB1 (Aha, Kibler, & Albert, 1991) learns the instance-based nearest neighbour classifier using normalized Euclidean distance.
- IBk is similar to IB1, which uses k-nearest neighbour instead of one nearest neighbour. Here, $k = 5$ is used.
- KStar (Cleary & Trigg, 1995). KStar is an instance-based classifier. The class label of a test instance is decided by entropy-based functions.

Table 1
Dataset description.

Dataset	Attributes	Instances	Classes
Balance-Scale	5	625	3
Breast-w	11	699	2
Chess	37	3196	2
Colic	27	368	2
Credit-A	15	690	2
Credit-G	21	1000	2
Glass	10	214	7
Heart-C	14	303	2
Heart-statlog	14	270	2
Hepatitis	20	155	2
Ionosphere	35	351	2
Iris	5	150	3
Labor	17	57	2
Lymphography	19	148	4
Sonar	61	208	2
Vehicle	19	846	4
Vote	17	435	2
Wine	14	178	3

- OneR (Holte, 1993). The classifier uses the minimum-error attribute for prediction.
- PART (Frank & Witten, 1998) builds a partial C4.5 decision tree in each iteration and turns the "best" leaf into a classification rule using the separate-and-conquer strategy.
- ZeroR. It uses 0-R classifiers for prediction.
- Decision Stump (Iba & Langley, 1992) generates a one-level decision tree classifier.
- C4.5 Decision Tree (DT) (Quinlan, 1993) generates a decision tree classifier.

These algorithms are also used as the meta-classifier candidates for ACO-Stacking. The parameters of ACO, including the number of ants in the colony, the maximal iteration, the evaporation rate and the constant *CC*, are listed in Table 2.

4.2. Compared approaches

In the experiments, the stackings found by ACO-Stacking are compared with the following ensemble approaches.

- AdaBoost with C4.5 DT as its base-level learning algorithm;
- Bagging with C4.5 DT as its base-level learning algorithm and $F = 0.67$;
- Random Forest (Breiman, 2001);
- StackingC with Naïve Bayes, IBk, and C4.5 DT as its base-level learning algorithms and a Multi-Response Model Tree as its meta learning algorithm (Seewald, 2002; Džeroski & Ženko, 2002);
- GA-Ensemble that uses the same base-level classifiers as ACO-Stacking. The meta combining method is determined by GA-Ensemble; either a Multi-Response Model Tree or a majority voting scheme can be selected. The parameters of GA-Ensemble are listed in Table 3.

Table 2
ACO parameters.

Parameter	Value
Colony size	30
Iterations	10
Evaporation rate	0.1
CC	10

Table 3
GA parameters.

Parameter	Value
Population size	30
Generations	10
Elite rate	0.1
Cull rate	0.1
Cross operation	Uniform
Mutation rate	0.1
Crossover rate	0.5

4.3. Results and analysis

Table 4 summarizes the results of the average accuracies of the approaches from the 18 datasets. In some datasets, such as Ionosphere, Iris and Vote, the performance of all the approaches is not significantly different from each other. In the simple datasets, all the approaches are promising. However, in some datasets, such as Balance-Scale and Sonar, the accuracies of stacking-based approaches (StackingC, GA-Ensemble, ACO-S1, ACO-S2, and ACO-S3) are better than the non-stacking-based approaches; furthermore, the metaheuristic stacking-based approaches are better than the non-metaheuristic stacking approaches. For example, in the Balance-Scale dataset, the accuracies of Bagging, AdaBoost and Random Forest are smaller than 80%, while the best result, 98.88%, is achieved by ACO-S1.

In the following empirical and statistical tests, we focus on the comparison between ACO-S3 and the other approaches. The comparisons of the different versions of ACO-Stacking are also given.

4.3.1. Empirical analysis

The empirical *w/t/l* test results are given in the last row of Table 4, where *w* means that ACO-S3 outperforms the corresponding approach, *t* means that their performances are the same and *l* means that ACO-S3 is not as good as the corresponding approach. Compared with Bagging, Random Forest and GA-Ensemble, ACO-S3 respectively wins in 12, 13, and 11 of the 18 datasets. It ties in one, two, and two datasets respectively. On the other hand, ACO-S3 loses in five, three, and five datasets respectively. Compared with StackingC, ACO-S3 wins in 10 datasets, ties in one dataset and loses in seven datasets. Compared with AdaBoost, ACO-S3 wins in 13 datasets, ties in one dataset, and loses in four datasets.

Relative Improvement (RAI) is also conducted to evaluate the approaches. RAI is calculated by using Eq. (5).

$$p = \sum \frac{\alpha_i - \alpha'_i}{\alpha'_i} \tag{5}$$

where α_i refers to the accuracy of ACO-S3 in the *i*th data set and $\alpha\prime_i$ refers to the accuracy of the approach being compared with. According to the RAI test in Table 5, ACO-S3 gains relative improvement of 97.05% with Bagging, 70.46% with AdaBoost, 71.56% with Random Forest, 58.04% with StackingC and 20.98% with GA-Ensemble. From the two empirical tests, ACO-S3 outperforms Bagging, AdaBoost, Random Forest, StackingC and GA-Ensemble.

4.3.2. Statistical analysis

To demonstrate the statistical significance of the experiments, pairwise *T*-tests are conducted. The performances of the other approaches and those of the ACO-S3 are compared to find statistical significance. The results of the *T*-test are also shown in Table 4. The *T*-test results show that ACO-S3 significantly outperforms Bagging in seven of the 18 datasets at the 5% level and in two of them at the 10% level. ACO-S3 is significantly better than Random Forest in four datasets at the 5% level and in three datasets at the 10% level. It is significantly superior to GA-Ensemble in three datasets at the 5% level and in two datasets at the 10% level. Moreover, ACO-S3 is

Table 4
The classification accuracies of the ensembles.

Dataset	Bagging	AdaBoost	Random Forest	StackingC	GA-Ensemble	ACO-S1	ACO-S2	ACO-S3
Balance-Scale	71.68 [a]	76.48 [a]	76.96[a]	86.08[a]	98.72	**98.88**	98.56	98.72
Breast-W	95.14[a]	96.42	95.99	**97.28**[d]	96.14[b]	97.00	95.14[a]	96.99
Chess	99.44	**99.50**	98.91[a]	99.44	99.19	99.34	99.14[a]	99.34
Colic	67.93[a]	70.92[a]	71.47[b]	64.13[a]	75.00	**82.88**[c]	76.90	78.26
Credit-A	86.38	84.35	84.35	**86.81**	85.65	84.35[a]	82.32[a]	85.94
Credit-G	74.0	69.6[a]	74.1	74.7	73.7[b]	74.8[b]	75.0	**76.1**
Glass	73.83	**79.44**[c]	73.36	69.16[a]	77.10	72.43	76.17	75.23
Heart-C	78.88	76.90	79.21	**84.16**[c]	77.89	81.19	74.59[a]	78.22
Heart-statlog	80.0[a]	80.37	78.15[a]	**84.16**	80.0[a]	81.85	75.93[a]	82.96
Hepatitis	83.23 [b]	85.81	80.65[a]	81.94	84.52	83.23	**87.74**	86.45
Ionosphere	**93.45**	93.16	**93.45**	90.88	92.88	92.02	89.17	92.31
Iris	95.33	93.33[b]	95.33	95.33	95.33	94.67	**96.0**	95.33
Labor	84.21[b]	89.47	87.72	89.47	85.96[a]	91.29	87.72	**92.9825**
Lymphography	79.05	81.08	81.08	83.11	82.43	82.43	**85.81**[c]	81.08
Sonar	74.52[a]	77.88	80.77	81.73	86.06	81.73	**87.98**[c]	83.65
Vehicle	76.60[a]	76.24[a]	77.07[b]	74.11[a]	75.53[a]	75.2941	74.23[a]	**79.91**
Vote	96.32	95.86	95.86	**96.78**	95.17	95.63	94.25	95.17
Wine	94.94[a]	96.63[a]	97.19[b]	96.07[a]	98.31	97.75[b]	98.31	**98.88**
w/t/l	12/1/5	13/1/4	13/2/3	10/1/7	11/2/5	11/2/5	12/1/5	–

[a] Using paired *T*-test, the average accuracy is significantly worse than that of ACO-S3 at 0.05 level.
[b] Using paired *T*-test, the average accuracy is significantly worse than that of ACO-S3 at 0.1 level.
[c] Using paired *T*-test, the average accuracy is significantly better than that of ACO-S3 at 0.05.
[d] Using paired *T*-test, the average accuracy is significantly better than that of ACO-S3 at 0.1 level.

Table 5
RAI test result.

	Bagging	AdaBoost	Random Forest	StackingC	GA-Ensemble	ACO-S1	ACO-S2	ACO-S3
RAI	97.05%	70.46%	71.56%	58.04%	20.98%	13.78%	29.53%	–

not significantly inferior to the above three approaches in any datasets in the experiments. Compared with AdaBoost, ACO-S3 is significantly superior in five datasets at the 5% level and in one dataset at the 10% level, while it is significantly inferior in one dataset at the 5% level. Compared with StackingC, ACO-S3 is significantly superior in five datasets at the 5% level and significantly inferior in two datasets.

The non-parametric Friedman test (Friedman, 1937) is conducted to compare the performance of different approches over multiple datasets (Demšar, 2006; García & Herrera, 2008). The average rankings of these approaches can be found in the second row of Table 6. The Friedman test obtains a *p*-value of 0.08041. In other words, we can reject the *null* hypothesis that all approaches have equivalent performance at the 10% level of significance. The Holm's procedure (Holm, 1979) is used to find the adjusted *p*-value when comparing various approaches with ACO-S3. These values are listed in the last row of Table 6. Compared with Bagging, Random Forest and AdaBoost, ACO-S3 performs significantly better than them at the 5% level. It can be observed that ACO-S3 outperforms ACO-S2 and GA-Ensemble significantly at the 10% level. The same conclusion can be obtained when Hochberg's procedure and Hommel's procedure have been used (García & Herrera, 2008).

In general, ACO-S3 is superior to many other approaches in different tests. Therefore, we can conclude that the performance of ACO-S3 is promising.

4.3.3. Comparisons of different versions of ACO-Stacking

The same tests (*w/t/l*, RAI, *T*-test, Friedman test, Holm's procedure) are used to compare the performance of different versions of ACO-Stacking. The results of *w/t/l* and RAI tests between the three versions are summarized in Table 7 and the *T*-test results between ACO-S1 and ACO-S2 are given in Table 8. The Friedman test generates a *p*-value of 0.09173. Thus we can reject the *null* hypothesis that the three different versions of ACO-Stacking have equivalent performance at the 10% level of significance.

Comparing ACO-S1 and ACO-S2, in the *w/t/l* test as well as the *p*-values in *T*-test (Table 8), ACO-S1 wins in 11 of the datasets and loses in seven datasets. ACO-S1 is significantly superior to ACO-S2 in six datasets at the 5% level and in three datasets at the 10% level. ACO-S2 significantly outperforms ACO-S1 in only one dataset at the 5% level and in two datasets at the 10% level. According to the RAI test, the result is −13.10%, which means ACO-S2 cannot show improvement over ACO-S1. The adjusted *p*-value obtained by using Holm's procedure is 0.40466. Thus ACO-S1 and ACO-S2 are not significantly different.

In the *w/t/l* test in Table 7, ACO-S3 wins in 12 of the datasets, ties in one dataset and loses in the remaining five datasets compared with ACO-S2. Furthermore, ACO-S3 outperforms ACO-S2 in six datasets at the 5% level and is inferior in Lymphography and Sonar at the 5% level (Table 4). According to the RAI test in Table 7, ACO-S3 gains relative improvement of 29.53% with ACO-S2. The

Table 6
Average rankings and adjusted *p*-values.

	Bagging	AdaBoost	Random Forest	StackingC	GA-Ensemble	ACO-S1	ACO-S2	ACO-S3
Average rankings	5.2778	5.0	5.1389	4.3611	4.6389	3.9722	4.7222	2.8889
Adjusted *p*-value	0.0241	0.0486	0.0351	0.1427	0.0989	0.1846	0.0989	–

Table 7
Results of *w/t/l* tests and RAI tests.

	Test	Result
ACO-S2 vs. ACO-S1	*w/t/l* test	7/0/11
	RAI test	−13.10%
ACO-S3 vs. ACO-S1	*w/t/l* test	11/2/5
	RAI test	13.78%
ACO-S3 vs. ACO-S2	*w/t/l* test	12/1/5
	RAI test	29.63%

Table 8
T-test results: comparing ACO-S2 with ACO-S1.

Dataset	*p*-value
Balance-Scale	0.0839330
Breast-W	0.00668037
Chess	0.0786766
Colic	0.03740137
Credit-A	0.01089679
Credit-G	0.44267226
Glass	0.0979825
Heart-C	0.00153940
Heart-statlog	0.02943033
Hepatitis	0.12527892
Ionosphere	0.22912368
Iris	0.0645235
Labor	0.0748713
Lymphography	0.19531785
Sonar	0.00471518
Vehicle	0.04860191
Vote	0.16595619
Wine	0.47400674

adjusted *p*-value obtained by using Holm's procedure is 0.06052. Thus ACO-S3 significantly outperforms ACO-S2 at the 10% level.

In the *w/t/l* test in Table 7,, ACO-S3 wins in 11 of the datasets, ties in two datasets and loses in five datasets compared with ACO-S1. In the *T*-test, ACO-S3 is significantly superior to ACO-S1 in one dataset at the 5% level and in two datasets at the 10% level, but inferior to ACO-S1 in one dataset at the 5% level (Table 4). According to the RAI test, the relative improvement is 13.78%. The adjusted *p*-value obtained by using Holm's procedure is 0.18242. Thus ACO-S1 and ACO-S3 are not significantly different.

We are also interested in the number of base-level classifiers used in the stackings found by different versions of ACO-Stacking. The average numbers of base-level classifiers in different versions of ACO-Stacking are given in Table 9. For ACO-S1, its average number of base-level classifiers is much more than those in ACO-S2 and ACO-S3. This interesting phenomenon could be explained by the differences of the versions. ACO-S1 focuses on the search for the combinations of base-level classifiers with the same meta-classifier. Given a sufficient number of iterations, ACO-S1, which is more stochastic without local information, can discover good stackings with more base-level classifiers. The other versions use local information to guide the searching process and use different meta-classifiers to extend the searching space. The local information in ACO-S2 helps the construction of the combination of base-level classifiers to focus on the "powerful" candidates so that some less strong, but potentially useful candidates, are ignored. Thus the average number of base-level classifiers in ACO-S2 is smaller. The major difference between ACO-S2 and ACO-S3 is that ACO-S3 uses the correlative differences as the local information. The correlative differences focus on searching the base-level classifiers which are not similar to the existing ones in the stackings. This local information does not ignore the base-level classifiers with "average" performance. The optimized local information improves the performance while bringing a small increase in the average number of base-level classifiers. From the above analysis, ACO-S3 could be the best of the three versions.

Table 9
Average numbers of base-level classifiers in stackings.

Approaches	Number of base-level classifiers
ACO-S1	4.9375
ACO-S2	3.125
ACO-S3	3.3333

5. A real-world cost-sensitive data mining application

In this section, ACO-Stacking is used to handle a real-world data mining application in direct marketing. Direct marketing is a type of marketing that reaches its potential customers without traditional advertising, such as TV, newspapers or radio, and instead communicates directly with the consumer with advertising such as direct mail, catalogues and email advertisements. Direct marketing companies often maintain massive databases of their customers' information, including (but not limited to) their contacts, their previous purchasing records, their responses to previous marketing campaigns and so on.

Not every customer in the databases is interested in the products or services of the company, so some customers will never purchase. Other customers will only purchase occasionally and spend small amounts of money. Only a few customers are highly loyal to the company and purchase frequently. The former two kinds of customers account for a much larger proportion of the databases than the loyal ones (e.g., 95% to 5%). In other words, the direct marketing databases are highly unbalanced.

Furthermore, buyers contribute different profits when they respond to a marketing campaign. Some buyers are identified as most likely to respond and make a purchase, so the company may send some gifts with the catalogue. However, although they respond, they may only place a small order; thus the company can only earn a small amount of profit. On the other hand, some buyers seldom respond to a campaign but will place a big order if they respond. So in the direct marketing problem, the profit varies significantly among customers. Thus, this problem is cost-sensitive.

Because of budget constraints and required return of marketing investment, the company cannot contact all customers in the database. Therefore, it is essential to identify the customers who are more responsive to marketing activities and more profitable for the company. For a marketing campaign, typically only the names in the top two deciles or the 80th percentile (i.e. those with the highest probabilities of responding) will receive the promotion materials from the company (Zahavi & Levin, 1997).

Direct marketing companies therefore build varieties of predictive models from the databases to narrow their target customer groups, thus realizing a desirable return within the budget. Until recently, the dominant models in this field were statistically based, for example regression and discriminant analysis. Some data mining and machine learning approaches were also proposed to learn models for direct marketing applications. For instance, Zahavi and Levin applied Neural Networks to target marketing (Zahavi & Levin, 1997). Bhattacharyya proposed his approach of applying a genetic algorithm (Bhattacharyya, 1999). Cui et al. studied model selection for direct marketing (Cui, Wong, Zhang, & Li, 2008). Our ACO-Stacking can be easily applied to handle this direct marketing problem.

5.1. The direct marketing database

A large real-life direct marketing dataset from a U.S.-based catalogue company provided by the Direct Marketing Education Foundation, is used to evaluate ACO-Stacking and other approaches. The

company sells multiple product lines of merchandise, from gifts and apparel to consumer electronics. It regularly sends catalogues to its customers by mail. This dataset contains 106,284 records in a recent promotion, as well as their purchase history over a 12-year history. The dataset also contains the demographic information from the 1995 U.S. Census and credit information from a commercial vendor. Thus there are 361 variables in each record. The most recent promotion sent a catalogue to every customer in this dataset and achieved a 5.4% response rate, representing 5740 buyers.

The statistical summary of the cost/profit from the direct marketing dataset is given in Table 10. The maximum profit is US$612.66, which is about 140 times the minimal profit and 16 times the average profit. The maximal cost is US$9.18, which is about 27 times the minimal cost and about 12 times the average cost. The average profit is about 52 times the average cost in the dataset.

For a direct marketing dataset with so many variables, it is necessary to conduct some features (variables) selection to reduce the dimension of it. In this application, 17 variables are selected by the forward wrapper selection process. For example, the variables about the lifetime total orders, the lifetime total sales, whether the customer placed telephone orders, whether the customer paid by cash, etc. are selected.

5.2. Evaluation methods for direct marketing models

In direct marketing applications, the accuracy may not be the most appropriate method for assessing the performance of classifiers (Wong & and Cui, 2010). First, despite the dataset being huge, the response rate is very small (5.4% in this case). In other words, the dataset is extremely unbalanced. If a classifier makes predictions that all the potential customers do not respond, the accuracy will still be 94.6%, which seems to be pretty good for conventional accuracy-based applications. However, this result is meaningless for this problem. As we mentioned before, due to the budget constraints, only the potential customers in the top decile or top two deciles of the database are likely to be contacted in a direct marketing campaign, but a model with high accuracy may not have superior performance in the top decile (s). Second, the accuracy cannot show the distinction of different misclassification errors. For direct marketing, false negatives are more costly than false positives, because the potential sale and profit of a false negative may be much larger than the mailing cost of a false positive.

The decile analysis which estimates the enhancement of the response rate and profit at different depths of the dataset is used to evaluate the performance of a classifier. To use the decile analysis, the names with their response rates should be sorted into a rank list in decreasing order. The names in the first decile indicate that they are most likely to respond and generate profits while the names in the last decile are unwilling to respond and purchase. The *cumulative lift*, which is usually the most important criterion for the decile analysis, will be used in this approach as well (Zahavi & Levin, 1997; Cui et al., 2008). *Lift* is a measure of the effectiveness of a predictive model, which is calculated as the ratio between the results obtained with the classifier and with a random model at a certain depth of the dataset. In direct marketing, the response rate and profit rate are the most important measures. Thus the *cumulative response lift*, *cumulative profit lift* and *lifted profits* are used to compare different approaches. Cumulative response lift evaluates the ratio between the response received from the customers with a classifier and those with a random model at a certain depth of the dataset. Cumulative profit lift evaluates the ratio between the earning profits obtained with a classifier and those with a random model. The lifted profits evaluate the actual amount of lifted profit obtained with a classifier will earn against that with a random model. A lift chart (Fig. 5) and different tables (Tables 12–17) are used to present the performance of different models across the ten deciles.

Table 10
Summary of the cost/profit (US$) of the direct marketing dataset.

Statistics metric	Value
Maximum Profit	612.66
Minimum Profit	4.36
Average Profit	38.77
Standard Deviation of Profit	37.622
Maximum Cost	9.18
Minimum Cost	0.34
Average Cost	0.74
Standard Deviation of Cost	0.301

5.2.1. ACO-Stacking for direct marketing problem

Due to the flexibility of ACO-Stacking, it is easy to modify this approach to tackle the direct marketing problem. Since the optimization objective is changed from maximizing the overall accuracy to maximizing the cumulative response/profit lift in certain deciles, α in the approach is modified accordingly. The total profit of the customers in the top two deciles in the validation set is used as α, the evaluation criterion of a stacking ensemble. However, the profit of each customer (instance) is transparent to the learning algorithms in the process of training the base-level classifiers and the meta-classifier. In other words, each instance in the training set is treated equally by the cost-insensitive learning algorithms. On the other hand, the profits of the customers of the validation set are used to calculate α, in order to find a good stacking ensemble. We modify ACO-S3 to generate a ranking list of the instances in the validation set by sorting their probabilities of responding in decreasing order. The profits of the instances in the top two deciles of the list are calculated to be the α of this configuration. The other components are the same as those in ACO-S3 discussed in Section 3.

5.3. Experiments and results

An experiment and an analysis of the results are conducted to evaluate the performance of ACO-Stacking in the direct marketing problem. The ten-fold cross validation scheme is used in the experiment as well.

In the experiment, ACO-Stacking uses ten different learning algorithms. They are C4.5 DT, CART, Decision Stump, Logistic, NB, NB Simple, NB Updateable, OneR, PART and VFI. NB simple is a

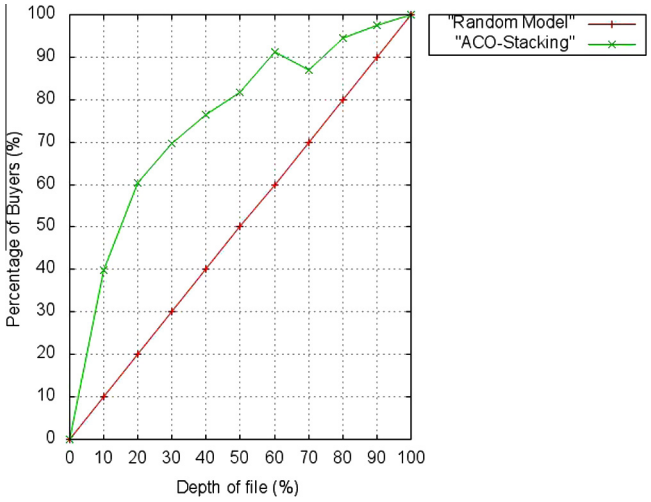

Fig. 5. Lift chart of ACO-Stacking.

Table 11
Parameters of ACO-Stacking for direct marketing application.

Parameter	Value
Colony size	20
Iterations	10
Evaporation rate	0.1
CC	10

	Buyer	Non-buyer
Predicted as Buyer	0	1
Predicted as Non-buyer	10	0

Fig. 6. Cost matrix of AdaCost.

variant of NB which models the numerical attributes by a normal distribution (Duda, Hart, & Stork, 2012). NB updateable is another variant of NB (John & Langley, 1995). VFI generates a classifier that classifies an instance based on feature intervals (Demiröz & Güvenir, 1997). The set of learning algorithms is different from that used in Section 4.1; because the size of the database is much larger than those of the benchmark datasets, some instances-based learning algorithms such as KStar and IBk are replaced by other learning algorithms. The parameters of ACO-Stacking are given in Table 11.

5.3.1. Compared methods

ACO-Stacking is compared with two sets of existing methods. The first set includes the conventional methods that have been applied in direct marketing problems, such as Logistic Regression, Naïve Bayes, Neural Networks and Bayesian Networks (Zahavi & Levin, 1997; Cui et al., 2008). The second set contains some ensemble and/or cost-sensitive methods including Bagging (Breiman, 1996), AdaCost (Fan, Stolfo, Zhang, & Chan, 1999) and AdaC2 (Sun, Kamel, Wong, & Wang, 2007). In Bagging, the learning algorithm is Logistic Regression and the random subset fraction is 0.667.

AdaCost (Fan et al., 1999) is a cost-sensitive version of AdaBoost (Freund & Schapire, 1997). It uses the different costs of the corresponding misclassification errors to adjust the training distribution on successive boosting rounds and thereby build a better cost-sensitive ensemble. The different costs of corresponding misclassification errors are given in a confusion matrix. To develop a confusion matrix, one must determine which errors might be committed and their corresponding costs. In this direct marketing dataset, there are two classes: buyer and non-buyer. Therefore there are two kinds of errors: classifying buyer as non-buyer and the reverse. The confusion matrix in this experiment is given in Fig. 6. The penalty for the error of mistaking non-buyer as buyer is one and the penalty for mistaking buyer as non-buyer is ten. The penalty for mistaking buyer as non-buyer is larger because the potential profit of a buyer is much larger than the cost of marketing material and mailing. Moreover, the number of iterations of the AdaCost is set to 10. Because of the constant costs defined in the cost matrix, AdaCost treats all instances which commit the same misclassification equally. However, the costs of different instances often vary even if the same misclassification errors are committed.

Sun et al. proposed an approach which incorporates the individual misclassification costs into the training distribution adjustment process of AdaCost (Sun et al., 2007). Three algorithms, AdaC1, AdaC2 and AdaC3 are proposed. AdaC2 performs better in their paper, so AdaC2 is compared in our experiment. The differences between AdaC2 and Adaboost is that the update rule in Adaboost (Freund & Schapire, 1997) is modified by adding the specific cost of each instance. For AdaCost and AdaC2, the weak learner is Logistic Regression.

5.3.2. Results and analysis

Tables 12 and 15 show the cumulative response lift of ACO-Stacking compared with the two sets of methods. Tables 13 and 16 display the cumulative profit lift of ACO-Stacking compared with the other methods. Moreover, Tables 14 and 17 respectively show the average lifted profit (US$) of ACO-Stacking and the compared methods. In the tables, the number in bold font in each decile indicates that the method in this column achieves the best results in this decile compared with the other methods. The pairwise *T*-tests are conducted to compare the results as well.

From Table 12, the ensembles generated by ACO-Stacking (ACO-Stacking ensembles) achieve the average cumulative response lift of 401.5 and 301.3 in the first two deciles respectively. The results suggest that by mailing to the first two deciles alone, the ACO-Stacking ensembles generate over three times as many respondents as a random mailing without a model. The average response lift of the ACO-Stacking ensembles is significantly higher than those of Bayesian Networks and Naïve Bayes in the top decile, and is significantly higher than those of Logistic Regression and Naïve Bayes in the top two deciles. From Table 13, the average cumulative profit lift of the ACO-Stacking ensembles is significantly higher than those of the conventional methods in the top decile and significantly higher than those of Naïve Bayes in the top six deciles. According to Table 14, an average lifted profit of US$9,198.7 will be obtained if a marketing campaign is conducted to the top 20% of customers identified by the ACO-Stacking ensembles.

The comparison between ACO-Stacking and the other ensemble and cost-sensitive methods is more interesting. As shown in

Table 12
Average cumulative response lift of ten-fold cross-validation compared with conventional methods.

Models Decile	Logistic Regression Response Lift	Bayesian Networks Response Lift	Neural Networks Response Lift	Naïve Bayes Response Lift	ACO-Stacking Response Lift
1	374.7 (15.9)[a]	357.6 (17.8)[a]	380.1 (21.7)	280.7 (19.0)[a]	**401.5 (47.5)**
2	261.3 (8.9)[a]	263.0 (7.8)	275.3 (9.2)	220.0 (11.4)[a]	**301.3 (70.4)**
3	216.4 (6.4)	214.1 (7.1)	218.6 (5.7)	187.1 (6.9)[a]	**232.3 (36.1)**
4	184.8 (3.9)	182.3 (5.3)	183.6 (4.9)	162.5 (5.3)[a]	**192.3 (21.1)**
5	161.4 (3.7)	158.8 (2.3)	160.5 (2.9)	146.6 (3.1)[a]	**164.2 (13.9)**
6	145.1 (2.1)	141.4 (2.4)	144.4 (2.2)	134.8 (2.2)[a]	**145.9 (7.6)**
7	130.2 (1.4)	128.2 (1.8)	130.6 (1.5)	126.8 (1.0)[a]	**130.9 (4.9)**
8	118.6 (1.2)	116.5 (1.4)	**118.8 (1.2)**	117.7 (1.5)	118.4 (2.9)
9	108.6 (1.2)	108.0 (0.7)	**108.9 (0.7)**	108.6 (0.5)	108.5 (1.5)
10	100.0	100.0	100.0	100.0	100.0

The reported figures are the means of the lifts of the 10 experiments, with the standard deviations in parentheses.
[a] Using paired *t*-test, the cumulative response lift is significantly smaller than that of ACO-Stacking at 0.05 level.

Table 13
Average Cumulative Profit Lift of Ten-fold Cross-validation Compared with Conventional Methods.

Models Decile	Logistic Regression Cum. Lift	Bayesian Networks Cum. Lift	Neural Networks Cum. Lift	Naïve Bayes Cum. Lift	ACO-Stacking Cum. Lift
1	589.9 (33.0)[a]	565.6 (39.7) [a]	597.1 (40.6) [a]	478.2 (44.3) [a]	**637.1 (63.4)**
2	354.8 (18.1)	365.2 (14.3)	377.5 (19.8)	326.6 (22.0)[a]	**414.1 (92.4)**
3	274.5 (11.8)	275.0 (11.2)	278.2 (9.5)	251.1 (14.4)[a]	**295.8 (49.1)**
4	221.3 (7.3)	220.4 (7.6)	220.7 (7.3)	203.4 (11.4)[a]	**232.7 (30.3)**
5	184.1 (5.7)	183.5 (4.2)	183.2 (4.7)	173.5 (5.3)[a]	**189.8 (20.1)**
6	159.3 (3.6)	156.5 (3.5)	159.3 (3.6)	151.7 (4.6)[a]	**162.9 (11.8)**
7	139.0 (3.0)	137.9 (3.0)	139.5 (2.3)	137.2 (2.9)	**141.1 (7.5)**
8	123.3 (1.6)	122.4 (2.3)	123.6 (1.6)	123.3 (2.3)	**123.9 (4.4)**
9	110.4 (1.2)	110.5 (1.3)	111.0 (1.0)	111.1 (0.8)	**111.2 (2.0)**
10	100.0	100.0	100.0	100.0	100.0

The reported figures are the means of the lifts of the 10 experiments, with the standard deviations in parentheses.
[a] Using paired *t*-test, the cumulative profit lift is significantly smaller than that of ACO-Stacking at 0.05 level.

Table 14
Average Lifted Profits ($) of Ten-fold Cross-validation Compared with Conventional Methods.

Deciles	Logistic Regression	Bayesian Networks	Neural Networks	Naïve Bayes	ACO-Stacking
1	7184.6	6821.0	7312.4	5553.0	**7886.2**
2	7770.5	7784.2	8155.9	6650.9	**9198.7**
3	7683.7	7707.2	7845.6	6662.6	**8600.2**
4	7120.2	7063.8	7082.6	6076.3	**7778.8**
5	6171.8	6124.9	6096.7	5392.7	**6581.9**
6	5222.6	4977.4	5227.8	4546.2	**5535.2**
7	4003.2	3886.3	4055.4	3812.4	**4209.8**
8	2732.3	2635.3	2769.3	2727.6	**2805.6**
9	1376.2	1396.6	1438.6	1464.0	**1481.9**
10	0.0	0.0	0.0	0.0	0.0

The reported figures are the means of the lifts of the 10 experiments.

Table 15
Average Cumulative Response Lift of Ten-fold Cross-validation Compared with Ensemble and Cost-Sensitive Methods.

Models Decile	Bagging Response Lift	AdaCost Response Lift	AdaC2 Response Lift	ACO-Stacking Response Lift
1	372.9 (17.1)[a]	139.0 (46.8)[a]	375.0 (17.2)[a]	**401.5 (47.5)**
2	261.7 (9.2)[a]	95.8 (14.2)[a]	263.1 (8.5)[a]	**301.3 (70.4)**
3	217.2 (5.0)	63.9 (9.5)[a]	217.0 (6.4)	**232.3 (36.1)**
4	184.2 (3.7)	80.4 (56.0)[a]	184.2 (4.0)	**192.3 (21.1)**
5	162.0 (3.7)	**194.8 (7.6)**[b]	161.1 (3.4)	164.2 (13.9)
6	145.2 (2.6)	**162.3 (6.4)**[b]	144.8 (2.0)	145.9 (7.6)
7	130.1 (1.5)	**139.1 (5.5)**[b]	129.9 (1.5)	130.9 (4.9)
8	118.8 (1.1)	**121.8 (4.8)**[b]	118.6 (1.0)	118.4 (2.8)
9	**108.7 (0.8)**	108.3 (4.0)	108.6 (0.8)	108.5 (1.5)
10	100.0	100.0	100.0	100.0

The reported figures are the means of the lifts of the 10 experiments, with the standard deviations in parentheses.
[a] Using paired *t*-test, the cumulative response lift is significantly smaller than that of ACO-Stacking at 0.05 level.
[b] Using paired *t*-test, the cumulative response lift is significantly larger than that of ACO-Stacking at 0.05 level.

Table 15, the ACO-Stacking ensembles significantly outperform those generated by Bagging, AdaCost and AdaC2 in the average cumulative response lift in the top two deciles. Compared with AdaCost, the average cumulative response lift of the ACO-Stacking ensembles is significantly higher in the top four deciles, while significantly inferior in the following four deciles. Similar phenomena can be found in Tables 16 and 17. Due to budget constraints, the average cumulative response/profit lift in the fifth and the following deciles may not be important for marketing decision makers. As shown in Table 16, the average cumulative profit lift of the ACO-Stacking ensembles is significantly higher than those of Bagging, AdaCost and AdaC2 in the top decile. From Table 17, the

Table 16
Average cumulative profit lift of ten-fold cross-validation compared with ensemble and cost-sensitive methods.

Models Decile	Bagging Cum. Lift	AdaCost Cum. Lift	AdaC2 Cum. Lift	ACO-Stacking Cum. Lift
1	584.9 (34.7)[a]	241.3 (54.9)[a]	593.6 (31.1)[a]	**637.1 (63.4)**
2	364.1 (20.4)	145.3 (30.0)[a]	367.7 (18.0)	**414.1 (92.4)**
3	275.2 (10.4)	86.7 (19.9)[a]	275.3 (11.7)	**295.8 (49.1)**
4	220.5 (6.6)	100.7 (74.8)[a]	220.7 (7.4)	**232.7 (30.3)**
5	184.5 (5.6)	**246.4 (10.1)**[b]	184.0 (5.5)	189.8 (20.1)
6	159.6 (4.1)	**196.4 (8.3)**[b]	159.2 (3.7)	162.9 (11.8)
7	138.9 (3.4)	**160.7 (7.0)**[b]	138.5 (3.5)	141.1 (7.5)
8	123.5 (1.5)	**133.9 (6.1)**[b]	123.2 (1.3)	123.9 (4.4)
9	110.8 (1.4)	**113.1 (5.2)**	110.5 (1.3)	111.2 (2.0)
10	100.0	100.0	100.0	100.0

The reported figures are the means of the lifts of the 10 experiments, with the standard deviations in parentheses.
[a] Using paired *t*-test, the cumulative profit lift is significantly smaller than that of ACO-Stacking at 0.05 level.
[b] Using paired *t*-test, the cumulative profit lift is significantly larger than that of ACO-Stacking at 0.05 level.

Table 17
Average lifted profits ($) of ten-fold cross-validation compared with ensemble and cost-sensitive methods.

Decile	Bagging	AdaCost	AdaC2	ACO-Stacking
1	7080.5	2047.0	7246.3	**7886.2**
2	7707.4	1352.1	7872.8	**9198.7**
3	7663.0	−565.2	7725.3	**8600.2**
4	7020.0	124.8	7096.1	**7778.8**
5	6131.5	**10713.0**	6160.1	6581.9
6	5177.6	**8461.4**	5208.5	5535.2
7	3921.7	**6208.1**	3962.4	4209.8
8	2680.2	**3952.7**	2717.5	2805.6
9	1340.2	**1708.9**	1380.4	1481.9
10	0.0	0.0	0.0	0.0

The reported figures are the means of the lifts of the 10 experiments.

direct marketers can gain more profits if they mail to the top 10%, 20%, 30% or 40% of customers according to the ACO-Stacking ensembles.

In summary, ACO-Stacking significantly outperforms most of the other methods in the top two deciles in both cumulative response lifts and cumulative profit lifts. This suggests that our approach can generate good cost-sensitive ensembles from ordinary learning algorithms.

6. Conclusions

6.1. Findings

In this work, a comprehensive study is conducted to optimize the performance of ACO-Stacking. In the study, we develop different versions of the ACO-Stacking approach by considering different ideas, such as the adoption of local information. In the first version (ACO-S1), no local information is implemented and only one learning algorithm (C4.5 DT) is used to create the meta-classifier. We focus on the effects of ACO in guiding the search and the combination of base-level classifiers. The first version aims to find as many as possible combinations of base-level classifiers with the same meta-classifier. The second version (ACO-S2) is quite different from the first. We implement the concept of a classifiers pool. The base-level classifiers are all trained prior to the stacking searching process, instead of training them when they are selected by some stackings. The classifiers pool may improve the efficiency of the training process (Ordóñez et al., 2008). The second difference is that we extend the searching space of the stacking by introducing the meta-classifiers set. The local information is also introduced in this version to accelerate the convergence process to find the optimal solution. The third version (ACO-S3) is similar to the second, the major change being that the correlative differences of base-level classifiers are used as the local information. From the comparison between ACO-Stacking and other ensemble methods including AdaBoost, Bagging, Random Forest, StackingC and GA-Ensemble, on the 18-benchmark datasets, it can be observed that ACO-Stacking has better performance than other ensemble methods in many datasets. By using Holm's procedure (Holm, 1979), ACO-S3 outperforms Bagging, Random Forest and AdaBoost at the 5% level of significance. It outperforms GA-Ensemble at the 10% level of significance.

From the comparison between these three versions of ACO-Stacking, ACO-S3 wins ACO-S1 in 11 benchmark datasets, and wins ACO-S2 in 12 benchmark datasets. However, ACO-S1 wins ACO-S2 in 11 benchmark datasets. We found that, without integrating local information into ACO-Stacking, the pure ACO-Stacking approach is more stochastic than other versions in generating ensembles. The pool of base-level classifiers is expected to provide better results. However, since ACO-S2 applies *precision* as the local information, the base-level classifiers with higher precision may have similar decision boundaries for certain difficult problems while some base-level classifiers with lower precision may have better decision boundaries. Moreover, if such situations occur frequently in the search process, the performance of ACO-S2 could be affected, which explains why ACO-S2 is significantly outperformed by ACO-S3 in six datasets at the 5% level. ACO-S3 uses correlative differences of base-level classifiers to overcome such problem in order to have a more diverse combination of base-level classifiers.

For the results of the real-world cost-sensitive data mining application in direct marketing, the proposed approach is able to generate good cost-sensitive ensembles as it significantly outperforms most of the other methods including Logistic Regression, Bayesian Networks, Bagging, AdaCost, and AdaC2 in both cumulative response lifts and cumulative profit lifts.

6.2. Contributions and implications

In this work, the contributions are threefold. Firstly, this is the first work to apply Ant Colony Optimization to a stacking configuration problem. Stacking is a well-known ensemble; however, how to configure an optimal stacking for a specific dataset is still regarded as a "black art". Furthermore, though Ant Colony Optimization performs well in many applications, it has not been implemented in solving stacking configuration problems. In this study, ACO is firstly integrated into the stacking configuration searching process. Secondly, we implement the local information in the ACO-Stacking process. Several kinds of local information are studied to improve the performance of ACO. The correlative differences, which represent the variations of predictions from different classifiers, are adopted in our latest version of ACO-Stacking. Thirdly, this approach could be applied to solve different data mining problems and real-world direct marketing problems.

Direct marketing data is often very unbalanced and cost-sensitive, thus making it hard to solve its problems with regular data mining models. ACO-Stacking is modified with cost-sensitive measures to tackle this problem. It is important to emphasize that it is not necessary for the learning algorithms used to generate the base-level classifiers and meta-level classifier to be cost-sensitive. By using our ACO-Stacking method, these non-cost-sensitive learning algorithms can be employed to handle cost-sensitive data-mining problems. In comparison with other approaches, our approach performs better. In the dataset, our approach gains a higher cumulative response rate and greater profits than other approaches.

6.3. Future work

In this work, we limit our ACO-Stacking approach to a single performance evaluation criterion for each application. For example, only accuracy is used in the benchmark datasets and only the cumulative profit lift is used for the direct marketing application. ACO has been proved to be strong in multi-criteria optimization problems. One possible future direction is to extend ACO-Stacking to find multi-criteria ensembles. Furthermore, only two measures for local information (Precision and correlative differences) are selected and applied in the approach. However, many other criteria can be employed as local information, so the best metric for local information can be further explored.

A relatively short execution time is very essential for an application. One future direction of this work is to modify ACO-Stacking to run in parallel to improve the efficiency. Much research has been done to parallelize the ACO approach on a Graphic Processing Unit thereby to accelerate the execution efficiency without many additional resources required.

Ensembles do not only refer to ensembles of classifiers. Nowadays, ensembles are widely used in clustering and regression tasks (Zhou, Wu, Tang, & Chen, 2001; Fern & Brodley, 2003). In the future, we may try to use our ACO-Stacking approach in clustering and regression tasks.

Acknowledgments

This research is supported by General Research Fund LU310111 from the Research Grant Council of the Hong Kong Special Administrative Region.

References

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning, 6*(1), 37–66.

Al-Ani, A. (2006). Feature subset selection using ant colony optimization. *International Journal of Computational Intelligence, 2*, 53–58.

Bhattacharyya, S. (1999). Direct marketing performance modeling using genetic algorithms. *INFORMS Journal on Computing, 11*(3), 248–257.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123–140.

Breiman, L. (2001). Random forest. *Machine Learning, 45*(1), 5–32.

Campos, L. M., Fernández-Luna, J. M., Gámez, J., & Puerta, J. M. (2002). Ant colony optimization for learning bayesian networks. *International Journal of Approximate Reasoning, 31*(3), 291–311.

Chan, A., & Freitas, A. (2006). *A new classification-rule pruning procedure for an ant colony algorithm. Artificial evolution*. Springer, pp. 25–36.

Cleary, J. G., & Trigg, L. E. (1995). K*: An instance-based learner using an entropic distance measure. In *Proceedings of the 12th international conference on machine learning* (pp. 108–114). Morgan Kaufmann.

Cui, G., Wong, M. L., Zhang, G., & Li, L. (2008). Model selection for direct marketing: Performance criteria and validation methods. *Marketing Intelligence and Planning, 26*(3), 275–292.

Demiröz, G., & Güvenir, A. (1997). Classification by voting feature intervals. *Proceedings of the 9th european conference on machine learning* (Vol. 1224, pp. 85–92). London, UK: Springer-Verlag.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.

Dietterich, T. G. (2000). *Ensemble methods in machine learning*. Springer, pp. 1–15.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. MIT Press.

Duda, R. O., Hart, P. E., & Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

Džeroski, S., & Ženko, B. (2002). In F. Roli & J. Kittler (Eds.), *Stacking with multi-response model trees. International workshop on multiple classifier systems* (Vol. 2364, pp. 201–211). Springer.

Fan, W., Stolfo, S. J., Zhang, J., & Chan, P. K. (1999). AdaCost: Misclassification cost-sensitive boosting. In *Proceedings of the sixteenth international conference on machine learning* (pp. 97–105).

Fern, X. Z., & Brodley, C. E. (2003). Random projection for high dimensional data clustering: A cluster ensemble approach. In *Machine learning-international workshop then conference* (pp. 186–193).

Frank, A., & Asuncion, A. (2010). UCI machine learning repository. <http://archive.ics.uci.edu/ml>.

Frank, E., & Witten, I. H. (1998). Generating accurate rule sets without global optimization. In *Proceedings of The 15th international conference on machine learning* (pp. 144–151). Morgan Kaufmann.

Freund, Y. & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the 13th international conference in machine learning* (pp. 148–156).

Freund, Y., & Schapire, R. E. (1997). Decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Science, 55*(1), 119–139.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association, 32*, 675–701.

García, S., & Herrera, F. (2008). An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons. *Journal of Machine Learning Research, 9*, 2677–2694.

Goss, S., Aron, S., Deneubourg, J. L., & Pasteels, J. M. (1989). Self-organized shortcuts in the argentine ant. *Naturwissenschaften, 76*(12), 579–581.

Gutjahr, W. J. (2002). Aco algorithms with guaranteed convergence to the optimal solution. *Information Processing Letters, 82*(3), 145–153.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. *SIGKDD Explorations, 11*(1), 10–18.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics, 6*, 65–70.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning, 11*, 63–90.

Iba, W., & Langley, P. (1992). Induction of one-level decision trees. In *Proceedings of the 9th international workshop on machine learning* (pp. 233–240). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.<http://portal.acm.org/citation.cfm?id=141975.142031?>.

John, G., & Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Proceedings of the eleventh conference on uncertainty in artificial intelligence* (pp. 338–345). Morgan Kaufmann.

LeCessie, S., & VanHouwelingen, J. C. (1992). Ridge estimators in logistic regression. *Applied Statistics, 41*(1), 191–201.

Ledezma, A., Aler, R., & Borrajo, D. (2002). Heuristic and optimization for knowledge discovery. *Heuristic search-based stacking of classifiers*. Idea Group Publishing, pp. 54–67.

Liu, B., Abbas, H. A., & McKay, B. (2003). Classification rule discovery with ant colony optimization. In *IEEE/WIC international conference on intelligent agent technology, IAT 2003* (pp. 83–88). IEEE.

Lu, Z., Wu, X., Zhu, X., & Bongard, J. (2010). Ensemble pruning via individual contribution ordering. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 871–880). New York, NY, USA: ACM<http://doi.acm.org/10.1145/1835804.1835914>.

Merz, C. J. (1999). Using correspondence analysis to combine classifiers. *Machine Learning, 36*(1-2), 33–58.

Ordóñez, F. J., Ledezma, A., & Sanchis, A. (2008). Genetic approach for optimizing ensembles of classifiers. In *Proceedings of the twenty-first international FLAIRS conference* (pp. 89–94).

Parpinelli, R. S., Lopes, H. S., & Freitas, A. A. (2002). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation, 6*, 321–332.

Pinto, P. C., Nägele, A., Dejori, M., Runkler, T. A., & Sousa, J. M. C. (2009). Using a local discovery ant algorithm for Bayesian network structure learning. *IEEE Transactions on Evolutionary Computation, 13*(4), 767–779.

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine, 6*(3), 21–45.

Quinlan, J. R. (1993). *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

Ramanathan, R. (2003). *An Introduction to Data Envelopment Analysis*. SAGE Publications.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*(2), 197–227.

Seewald, A. K. (2002). How to make stacking better and faster while also taking care of an unknown weakness. In *Proceedings of the 19th international conference on machine learning, ICML '02* (pp. 554–561).

Sivagaminathan, R. K., & Ramakrishnan, S. (2007). A hybrid approach for feature subset selection using neural networks and ant colony optimization. *Expert Systems with Applications, 33*, 49–60.

Sun, Y., Kamel, M. S., Wong, A. K., & Wang, Y. (2007). Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition, 40*(12), 3358–3378.

Ting, K. M., & Witten, I. H. (1999). Issues in stacked generalization. *Journal of Artificial Intelligence Research, 10*(1), 271–289.

Todorovski, L., & Džeroski, S. (2000). Combining multiple models with meta decision trees. In *Proceedings of the 4th european conference on principles of data mining and knowledge discovery, PKDD '00* (pp. 54–64). Berlin, Heidelberg: Springer.

Wang, Z., & Feng, B. (2005). Classification rule mining with an improved ant colony algorithm. In *AI 2004: Advances in Artificial Intelligence* (pp. 357–367). Springer.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks, 5*(2), 241–259.

Wong, M. L. & Cui, G. (2010). Data mining using parallel multi-objective evolutionary algorithms on graphics hardware. In *Proceedings of 2010 IEEE Congress on Evolutionary Computation* (pp. 1–8).

Zahavi, J., & Levin, N. (1997). Applying neural computing to target marketing. *Journal of Interactive Marketing, 11*(1), 5–22.

Zhang, X., Chen, X., & He, Z. (2010). An ACO-based algorithm for parameter optimization of support vector machines. *Expert Systems with Applications, 37*, 6618–6628.

Zheng, Z., & Padmanabhan, B. (2007). Constructing ensembles from data envelopment analysis. *INFORMS Journal on Computing, 19*(4), 486–496.

Zhou, Z. H., Wu, J. X., Tang, W., & Chen, Z. Q. (2001). Combining regression estimators: Ga-based selective neural network ensemble. *International Journal of Computational Intelligence and Applications, 1*(4), 341–356.

Zhu, D. (2010). A hybrid approach for efficient ensembles. *Decision Support Systems, 48*(3), 480–487.