International Journal of Soft Computing 2 (4): 555-561 2007

Comparison Between Traditional Data Mining Techniques and Entropy-Based Adaptive Genetic Algorithm for Learning Classification Rules

[1]V.N. Rajavarman and [2]S.P. Rajagopalan
[1]Department of Information Technology,
[2]School of Computer Science and Engineering, Dr. M.G.R. University, Chennai-95, India

Abstract: Genetic algorithm is one of the commonly used approaches on data mining. In this study, we apply a genetic algorithm approach for classification problems. Binary coding is adopted in which an individual in a population consists of a fixed number of rules that stand for a solution candidate. The evaluation function considers 4 important factors which are error rate, entropy measure, rule consistency and hole ratio, respectively. Adaptive asymmetric mutation is applied by the self-adaptation of mutation inversion probability from 1-0 (0-1). The generated rules are not disjoint but can overlap. The final conclusion for prediction is based on the voting of rules and the classifier gives all rules equal weight for their votes. Based on three databases, we compared our approach with several other traditional data mining techniques including decision trees, neural networks and naive bayes learning. The results show that our approach outperformed others on both the prediction accuracy and the standard deviation.

Key words: Genetic algorithm, adaptive asymmetric mutation, entropy, voting-based classifier

INTRODUCTION

Genetic algorithms have been successfully applied to a wide range of optimization problems including design, scheduling, routing and control, etc. Data mining is also one of the important application fields of genetic algorithms. In data mining, GA can be used to either optimize parameters for other kinds of data mining algorithms or discover knowledge by itself. In this latter task the rules that GA found are usually more general because of its global search nature. In contrast, most other data mining methods are based on the rule induction paradigm, where the algorithm usually performs a kind of local search. The advantage of GA becomes more obvious when the search space of a task is large.

In this study we apply a genetic algorithm approach for classification problems. First we use binary coding in which an individual solution candidate consists of a fixed number of rules. In each rule, k bits are used for the possible k values of a certain attribute. Continuous attributes are modified to threshold-based boolean attributes before coding. Rule consequent is not explicitly coded in the string, instead, the consequent of a rule is determined by the majority of training examples it matches.

Four important factors are considered in our evaluation functions. Error rate is calculated by the predicting results on the training examples. Entropy is used to measure the homogeneity of the examples that a certain rule matches. Rule consistency is a measure on the consistency of classification conclusions of a certain training example given by a set of rules. Finally, hole ratio is used to evaluate the percentage of training examples that a set of rules does not cover. We try to include related information as complete as possible in the evaluation function so that the overall performance of a rule set can be better.

An adaptive asymmetric mutation operator is applied in our reproduction step. When a bit is selected to mutate, the inversion probability from

1-0 (0-1) is not 50% as usual. The value of this probability is asymmetric and self-adaptive during the running of the program. This is made to reach the best match of a certain rule to training examples. For crossover, two-point crossover is adopted in our approach.

We used three real databases to test our approach: Database A,B and C. We compared our performance with four well-known methods from data mining, namely Induction Decision Trees (ID3) (Quinlan, 1986) ID3 with Boosting (Quinlan, 1996), Neural Networks and Naive Bayes (Mitchell, 1997). The appropriate state-of-the-art techniques are incorporated in these non-GA methods to improve their performances. The results show that our GA approach (Yang and Yen, 2000) outperformed other approaches on both the prediction accuracy and the standard deviation.

Corresponding Author: V.N. Rajavarman, Department of Information Technology, Dr. M.G.R. University, Chennai-95, India

OUR GA APPROACH

In this study we present our GA approach for classification problem. The key idea of the algorithm is general and should be applicable for various kinds of classification problems. Some parameter values used in the algorithm might be task dependent.

Individual's encoding: Each individual in the population consists of a fixed number of rules. In other words, the individual itself is a complete solution candidate. In our current implementation, we set this fixed number as 10 which well satisfied the requirement of our testing databases. The antecedent of a certain rule in the individual is formed by a conjunction of n attributes, where *n* is number of attributes being mined. K bits will be used to stand for an attribute if this attribute has k possible values. Continuous attributes will be partitioned to threshold-based boolean attribute in which the threshold is a boundary (adjacent examples across this boundary differ in their target classification) that maximizes the information gain. Therefore, two bits will be used for a continuous attribute. The consequent of a rule is not explicitly encoded in the string. In contrast, it is automatically given based on the proportion of positive/negative examples it matches in the training set. We will illustrate the encoding method by the following example.

Suppose our task has three attributes and they have 4, 2, 5 possible values, respectively. Then an individual in the population can be represented as following:

```
A1 A2  A3    A1 A2 A3    A1   A2  A3
0110 11 10110  1110  01  10011  ……  1100  11  01110
  Rule 1       Rule 2      ……      Rule 10
```

Ten rules are included in this individual. The architecture of each rule is same. We will use rule 1 to explain the meaning of encoding. In this example, the meaning of the antecedent of rule 1 is:

If (A1=value 2 OR value 3) AND (A2=value 1 OR value 2) AND (A3=value 1 OR value 3 OR value 4)

If all the bits belong to one attribute are 0s, it means that attribute can not equal to any possible value and hence this is meaningless. To avoid this, we add one step before the evaluation of the population. We will check each rule in each individual one by one, if the above case happens, we will randomly select one bit of that attribute and change it to one.

The consequent of a rule is not encoded in the string. It will be determined by the proportion situation of the training examples that rule matches. Suppose i is one of the classifications, the consequent of a rule will be i if

$$\frac{N_{matched\_i}}{N_{matched}} > \frac{N_{training\_i}}{N_{training}}$$

Where $N_{matched\_i}$ is the number of examples whose classification is *i* and matched by that rule, $N_{matched\_i}$ is the total number of examples that the rule matches; $N_{trauning\_i}$ is the number of training examples whose classification is i, $N_{training}$ is the total number of training examples.

For example, if the distribution of the positive examples and negative examples in the training set is 42% and 58% and among the examples of rule 1 matches positive/negative examples are half to half, then the consequent of rule 1 should be positive because 0.5 > 0.42. Since the testing databases we use at this time don't have a very uneven distribution on the classification of training examples, in our current implementation we didn't specially consider the interestingness of rules but use this strategy to keep enough rules to match examples with minor classification. Our encoding method is not limited to two-category classification but applicable to multi target value problems.

Fitness function: It is very important to define a good fitness function that rewards the right kinds of individuals. We try to consider affecting factors as complete as possible to improve the results of classification. Our fitness function is defined as following:

Fitness = Error rate + Entropy measure + Rule consistency + Hole ratio We will elaborate each part in the fitness function below.

Error rate: It is well known that accuracy is the most important and commonly used measure in the fitness function as the final goal of data mining is to get good prediction results. Since our objective function here is minimization, we use error rate to represent this information. It is calculated as:

Error rate = Percent of misclassified examples

If a rule matches a certain example, the classification it gives is its consequent part. If it doesn't match, no classification is given. An individual consists of a set of rules, the final classification predicted by this rule set is based on the voting of those rules that match the example. The classifier gives all matching rules equal weight. For instance, in an individual (which has ten rules here), one rule doesn't match, 6 rules give positive classification and 3 rules give negative classification on a training example, then the final conclusion given by this individual on that training example is positive. If a tie happens (i.e., 4 positive classifications and four negative classifications), the final conclusion will be the majority classification in

the training examples. If none of the rules in the individual matches that example, the final conclusion will also be the majority classification in the training examples. The error rate measure of this individual is the percent of misclassified examples among all training examples.

Entropy measure: Entropy is a commonly used measure in information theory. Originally it is used to characterize the (im)purity of an arbitrary collection of examples. In our implementation entropy is used to measure the homogeneity of the examples that a rule matches.

Given a collection S, containing the examples that a certain rule R matches, let P_i be the proportion of examples in S belonging to class i, then the entropy Entropy(R) related to this rule is defined as:

$$\text{Entropy}(R) = -\sum_{i=1}^{n}(p_i \log_2(p_i))$$

where *n* is the number of target classifications.

While an individual consists of a number of rules, the entropy measure of an individual is calculated by averaging the entropy of each rule:

$$\text{Entropy}(\text{individual}) = \frac{\sum_{i=1}^{N_R}\text{Entropy}(R_i)}{N_R}$$

Where N_R is number of rules in the individual (in our current implementation it is 10).

The rationale of using entropy measure in fitness function is to prefer those rules that match less examples whose target values are different from rule's consequent. High accuracy does not implicitly guarantee the entropy measure is good because the final classification conclusion of a certain training example is based on the comprehensive results of a number of rules. It is very possible that each rule in the individual has a bad entropy measure but the whole rule set still gives the correct classification. Keeping low entropy value of an individual will be helpful to get better predicting results for untrained examples.

Rule consistency: The final predicted classification of a training example is the majority classification made by rules in an individual. Let's consider the following classifications made by two individuals on an example:

Individual a: Six rules . +, four rules . -, final classification: +

Individual b: Nine rules . +, one rule . -, final classification: +

We will prefer the second individual since it is less ambiguous. To address this rule consistency issue, we add another measure in the fitness function. The calculation is similar to the entropy measure. Let $P_{correct}$ be the proportion of rules in one individual whose consequent is same with the target value of the training example, then

$$\text{Rule}_{\text{consistency}}(\text{individual}) = -p_{correct}\log_2 p_{correct} - (1-p_{correct})\log_2(1-p_{correct})$$

We should notice that this formula will give the same rule consistency value when $p_{correct}$ and $(1\text{-}p_{correct})$ switch each other. Therefore a penalty will be given when $p_{correct}$ is smaller than 0.5. In this case $\text{Rule}_{\text{consistency}} = 2\text{-}\text{Rule}_{\text{consistency}}$.

The above calculation is based on the predicting results for one training example. The complete measure of rule consistency of an individual should be averaged by the number of training examples.

Hole ratio: The last element in the fitness function is the hole ratio. It is a measure of rule's coverage on training examples. Coverage is not a problem for traditional inductive learning methods like decision trees, since the process of creating trees guarantees that all the training examples will be covered in the tree. However, this also brings a new problem that it may be sensitive to noise. GA approach does not guarantee that the generated rules will cover all the training examples. This allows flexibility and may be potentially useful for future prediction. In real implementation we still hope the coverage should reach a certain point. For instance, if a rule only matches one training example and its consequent is correct, the accuracy and entropy measure of this rule are both excellent but we do not prefer this rule because its coverage is too low.

In our fitness function the hole ratio equals to 1-coverage, in which the latter is calculated by the union of examples that are matched and also correctly predicted by the rules in an individual. Totally misclassified examples (not classified correctly by any rule in the individual) will not be included even though they are matched by some rules. The following is the formula to calculate the hole ratio for binary classification problem (positive, negative).

$$\text{Hole} = 1 - \frac{\left|\bigcup_i P_i^+\right| + \left|\bigcup_i N_i^-\right|}{|S|}$$

Where P_i^+ stands for those examples whose target value is positive and classified as positive by at least one

rule in the individual, stands for those examples whose target value is negative and classified as negative by at least one rule in the individual. |S| is the total number of training examples.

Adaptive asymmetric mutation: In our reproduction step, traditional bit inversion is used on mutation. However, we found many examples will not be matched if we keep number of 1's and 0's approximately equivalent in an individual (i.e., the inversion probability from 1-0 and 0-1 are both 50%). The learning process will become a majority guess if there are too many unmatched examples. Therefore, we put forward a strategy of adaptive asymmetric mutation in which the inversion probability from 1-0 (0-1) is self-adaptive during the process of run. The asymmetric mutation biases the population toward generating rules with more coverage on training examples. The self-adaptation of inversion probability makes the optimal mutation parameter be automatically adjusted.

We presented an adaptive simplex genetic algorithm before (Yang and Yen, 2000) in which the percentage of simplex operator is self-adaptive during the process of run. Similar idea is adopted here that average of fitness is used as a feedback to adjust the inversion probability. The process of self-adaptation is described as following:

- An initial inversion probability is set (e.g., 0.5 for 1-0). Use this probability on mutation to produce a new generation. Calculate the average fitness of this generation.
- Randomly select the direction of changing this probability (increase, decrease). Modify the probability along that direction with a small amount (0.02 in our current implementation). Use the new probability to produce the next generation and calculate the average fitness of the new generation.
- If the fitness is better (value is smaller), continue on this direction and the amount of change is:

$\Delta p = \max\{0.05, (1 - \text{fitness}_{new} / \text{fitness}_{old}) * 0.1\}$ If the fitness is worse (value is larger), reverse the direction and the amount of change is:

$\Delta p = \max\{0.05, (\text{fitness}_{new} / \text{fitness}_{old} - 1) * 0.05\}$ Use the new probability to produce the next generation and calculate the average fitness of the new generation.

Repeat step 3 until the program ends.

RESULTS AND DISCUSSION

We tested our approach on three real databases. We compared our approach with four other traditional data mining techniques. This section will present the testing results.

The information of databases

Database A: This database concerns credit card applications. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data. This database is interesting because there is a good mix of attributes ---- continuous, nominal with small numbers of values and nominal with larger numbers of values. There are 15 attributes plus one target attribute. Total number of instances is 690.

Database B: This database saves 1984 United States Congressional Voting Records. The data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the CQA. The CQA lists nine different types of votes: voted for, paired for and announced for (these three simplified to yea), voted against, paired against and announced against (these three simplified to nay), voted present, voted present to avoid conflict of interest and did not vote or otherwise make a position known (these three simplified to an unknown disposition). There are 16 attributes plus one target attribute. Total number of instances is 435 (267 democrats, 168 republicans).

Database C: This database concerns heart disease diagnosis. The data was provided by V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D. There are 14 attributes plus one target attribute. Total number of instances is 303.

The description of non-GA approaches: We used four well-known methods from machine learning, namely Induction Decision Trees (ID3) (Quinlan, 1980), Decision Trees with Boosting (Quinlan, 1996), Neural Networks and Naïve Bayes (Mitchell, 1997), to compare the performance of our improved GA. Appropriate state-of-the-art techniques have been incorporated in most of the non-GA methods to improve their performance. The following is a description of the non-GA approaches we used for the performance comparison studies.

Induction decision trees: The construction of a decision tree is divided into two stages. First, creating an initial, large decision tree using a set of training set. Second, pruning the initial decision tree, if applicable, using a validation set. Given a noise-free training set, the first stage will generate a decision tree that can classify correctly all examples in the set. Except that the training set covers all instances in the domain, the initial decision tree generated will over fit the training data, which then reduce its performance in the test data. The second stage helps alleviate this problem by reducing the size of the

tree. This process has an effect in generalizing the decision tree that hopefully could improve its performance in the test data.

During the construction of an initial decision tree, the selection of the best attribute is based on either the Information Gain (IG) or Gain Ratio (GR). A binary split is applied in nodes with continuous-valued attributes. The best cut-off value of a continuous-valued attribute is locally selected within each node in the tree based on the remaining training examples. The tree's node expansion stops either when the remaining training set is homogeneous (e.g., all instances have the same target attribute values) or when no attribute remains for selection. The decision on a leaf resulting from the latter case is determined by selecting the majority of the target attribute value in the remaining training set.

Decision tree pruning is a process of replacing sub-trees with leaves to reduce the size of the decision tree while retaining and hopefully increasing the accuracy of tree's classification. To obtain the best result from the induction decision tree method, we varied the use of pruning algorithm to the initial decision tree. We considered using the following decision trees pruning algorithms: Critical-value pruning (Mingers, 1987), minimum-error pruning (Niblett and Bratko, 1986), pessimistic pruning, cost-complexity pruning and reduced-error pruning (Quinlan, 1987).

Induction decision trees with boosting: Decision Tree with Boosting is a method that generates a sequence of decision trees from a single training set by re-weighting and re-sampling the samples in the set (Quinlan, 1996). Initially, all samples in the training set are equally weighted so that their sum is one. Once a decision tree has been created, the samples in the training set are re-weighted in such a way that misclassified examples will get higher weights than the ones that are easier to classify. The new samples weights are then renormalized and next decision tree is created using higher-weighted samples in the training set. In effect, this process enforces the more difficult samples to be learned more frequently by decision trees. The trees generated are then given weights in accordance with their performance in the training examples (e.g., their accuracy in correctly classifying the training data). Given a new instance, the new instance class is selected from the maximum weighted average of the predicted class over all decision trees.

Neural networks: Inspired in part by biological learning systems, neural networks approach is built from a densely interconnected set of simple units. Since this technique offers many design selections, we fixed some of them to the ones that had been well proven to be good or acceptable design choices. In particular, we use a network architecture with one hidden layer, the back-propagation learning algorithm (Rumelhart *et al*., 1986) the delta-bar-delta adaptive learning rates (Jacobs, 1988) and the Nguyen-Widrow weight initialization (Nguyen and Widrow, 1990). Discrete-valued attributes fed into the networks input layer is represented as 1-of-N encoding using bipolar values to denote the presence (e.g., value 1) and the absence (e.g., value -1) of an attribute value. Continuous-valued attribute is scaled into a real value in the range [-1, 1]. We varied the design choices for batch versus incremental learning and 1-of-N versus single network output encoding.

Naive bayes: Naive Bayes classifier is a variant of the Bayesian learning that manipulates directly probabilities from observed data and uses these probabilities to make an optimal decision. This approach assumes that attributes are conditionally independent given a class. Based on this simplifying assumption, the probability of observing attributes' conjunction is the product of the probabilities for the individual attributes. Given an instance with a set of attribute-value pairs, the Naive Bayes approach will choose a class that maximizes the conditional probability of the class given the conjunction of attributes values. Although in practice the independence assumption is not entirely correct, it does not necessarily degrade the system performance (Domingos and Pazzani, 1997).

We also assume that the values of continuous-valued attributes follow Gaussian distribution. Hence, once the mean and the standard deviation of these attributes are obtained from the training examples, the probability of the corresponding attributes can be calculated from the given attribute values. To avoid zero frequency count problem that can dampen the entire probabilities calculation, we use an m-estimate approach in calculating probabilities of discrete-valued attributes (Mitchell, 1997).

Comparison results: For each database, k-fold cross-validation method is used for evaluation. In this method, a data set is divided equally into k disjoint subsets. k experiments are then performed using k different training-test set pairs. A training-test set pair used in each experiment is generated by using one of the *k* subsets as the test set and using the remaining subsets as the training set. Given k disjoint subsets, for example, the first experiment takes the first subset for the test set and uses the second subset through the kth subset for the training set. The second experiment uses subset 1 and subset 3

Table 1: The comparison results on the prediction accuracy and standard deviation (%) of database A

	Our GA approach	Decision trees (IG, Min-Err)	Decision trees with boosting (RG, Cost-Com, 21 trees)	Neural networks (1-of-N, batch learning)	Naive bayes
Run 1	85.77	82.69	84.23	84.23	61.15
Run 2	84.23	79.62	81.15	81.15	73.46
Run 3	84.23	84.23	85.77	85.77	79.62
Run 4	87.31	85.77	85.77	84.23	76.54
Run 5	81.15	76.54	76.54	79.62	70.38
Run 6	84.23	82.69	82.69	82.69	75.00
Run 7	79.62	76.54	79.62	79.62	68.85
Run 8	82.69	81.15	82.69	81.15	78.08
Run 9	85.77	81.15	84.23	82.69	71.92
Run 10	81.76	83.24	86.18	81.76	70.00
Average	83.68	81.36	82.89	82.29	72.50
Standard deviation	2.37	3.06	3.08	2.03	5.36

Table 2: The comparison results on the prediction accuracy and standard deviation (%) of database B

	Our GA approach	Decision trees (IG, Red-Err)	Decision trees with boosting (IG, Red-Err,3 trees)	Neural networks (1-of-N, batch learning)	Naive Bayes
Run 1	90.35	90.35	90.35	88.02	88.02
Run 2	92.67	88.02	92.67	92.67	85.70
Run 3	92.67	92.67	92.67	92.67	88.02
Run 4	90.35	90.35	92.67	92.67	83.37
Run 5	90.35	90.35	90.35	88.02	83.37
Run 6	92.67	92.67	92.67	92.67	83.37
Run 7	90.35	88.02	88.02	90.35	88.02
Run 8	92.67	90.35	90.35	90.35	85.70
Run 9	95.00	95.00	92.67	90.35	85.70
Run 10	90.83	88.75	92.92	90.03	80.42
Average	91.79	90.65	91.54	90.86	85.17
Standard deviation	1.59	2.23	1.67	1.46	2.53

Table 3: The comparison results on the prediction accuracy and standard deviation (%) of database C

	Our GA approach	Decision trees (IG, Red-Err)	Decision trees with boosting (IG, Red-Err,21 trees)	Neural networks (1-of-N, incr learning)	Naive Bayes
Run 1	84.83	76.36	83.14	76.36	84.83
Run 2	78.05	72.97	79.75	79.75	74.66
Run 3	76.36	71.27	71.27	69.58	78.05
Run 4	83.14	71.27	78.05	84.83	78.05
Run 5	78.33	61.67	66.67	76.67	75.00
Average	80.14	70.71	75.77	77.44	78.12
Standard deviation	3.64	5.46	5.54	5.66	4.08

through subset k for the training set; and takes subset 2 for the test set and so on. All results from a particular database are averaged along with its variance over k experiment runs.

Based on their size, the credit database and voting database are partitioned into 10 disjoint subsets, the heart database is partitioned into 5 subsets. Table 1-3 show the performance comparison results of different approaches on these three databases. For decision trees with and without boosting, we present only the best experimental results after varying the data splitting methods as well as the pruning algorithms described earlier. Similarly, results from neural networks approach are selected from the ones that provide the best performance after varying the use of different network output encoding and batch versus incremental learning methods.

In Table 1, the decision tree is generated using information gain for data splitting and minimum-error pruning. Decision trees with boosting generates 21 different decision trees, each is constructed by using gain ratio for data splitting and cost-complexity pruning algorithm. Batch learning and 1-of-N output encoding are used in neural networks.

In Table 2, the best results from both decision trees with and without boosting are obtained from using information gain for data splitting and reduced-error pruning algorithm. Only 3 decision trees are needed in the decision trees with boosting.

In Table 3, the best results from neural networks are obtained from applying incremental learning and 1-of-N network output encoding.

From the above results we can see that our GA

approach outperformed other approaches on both the average prediction accuracy and the standard deviation. The advantage of our GA approach becomes more obvious on heart database, which is most difficult to learn among the three. During the process of running, we also found that the training accuracy and testing accuracy of GA approach are basically in the same level, while the training accuracy is often much higher than testing accuracy for other approaches. This proves that GA approach is less sensitive to noise and might be more effective for future prediction.

CONCLUSION

In this study, we have applied the genetic algorithm approach for classification problem. An individual in a population is a complete solution candidate that consists of a fixed number of rules. Rule consequent is not explicitly encoded in the string but determined by the match situation on training examples of the rule. To consider the performance affecting factors as complete as possible, four elements are included in the fitness function which are predicting error rate, entropy measure, rule consistency and hole ratio, respectively. Adaptive asymmetric mutation and two-point crossover are adopted in reproduction step. The inversion probability of 1-0 (0-1) in mutation is self-adaptive by the feedback of average fitness during the run. The generated classifier after evolution is voting-based. Rules are not disjoint but allowed to overlap. Classifier gives all rules the equal weight for their votes. We tested our algorithm on three real databases and compared the results with four other traditional data mining approaches. It is shown that our approach outperformed other approaches on both prediction accuracy and the standard deviation.

Further testing on various databases is in progress to test the robustness of our algorithm. Splitting continuous attribute into multiple intervals rather than just two intervals based on a single threshold is also considered to improve the performance.

REFERENCES

Domingos, P. and M. Pazzani, 1997. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. Machine Learning, 29: 103-130.

Jacobs, R.A., 1988. Increased Rates of Convergence Through Learning Rate Adaptation. Neural Networks, 1: 295-307.

Mingers, J., 1987. Expert Systems-Rule Induction with Statistical Data. J. Operational Res. Soc., 38: 39-47.

Mitchell Tom, 1997. Machine Learning. New York: McGraw-Hill.

Niblett, T., 1986. Constructing Decision Trees in Noisy Domains. In: I. Bratko and N. Lavrac (Eds.), Progress in Machine Learning. England: Sigma Press.

Nguyen, D. and B. Widrow, 1990. Improving the Learning Speed of Two-Layer Networks by Choosing Initial Values of the Adaptive Weights. International Joint Conference on Neural Networks, San Diego, CA, 3: 21-26.

Quinlan, J.R., 1986. Induction of Decision Trees. Machine Learning, 1: 81-106.

Quinlan, J.R., 1987. Simplifying Decision Trees. Int. J. Man-Machine Stud., 27: 221-234.

Quinlan, J.R., 1996. Bagging, Boosting and C4.5. In Proceedings of the 13th National Conference on Artificial Intelligence, pp: 725-730.

Rumelhart, D.E., G.E. Hinton and R.J. William, 1986. Learning Representations by Back-Propagation Error. Nature, 323: 533-536.

Yang, L. and J. Yen, 2000. An adaptive simplex genetic algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp: 379.