Reusing Genetic Programming for Ensemble Selection in Classification of Unbalanced Data

Urvesh Bhowan, Mark Johnston (Member IEEE), Mengjie Zhang (Senior Member IEEE) and Xin Yao (Fellow IEEE)

***Abstract*—Classification algorithms can suffer from performance degradation when the class distribution is unbalanced. This paper develops a two-step approach to evolving ensembles using genetic programming (GP) for unbalanced data. The first step uses multi-objective (MO) GP to evolve a Pareto-approximated front of GP classifiers to form the ensemble by trading-off the minority and the majority class against each other during learning. The MO component alleviates the reliance on sampling to artificially re-balance the data. The second step, which is the focus this paper, proposes a novel ensemble selection approach using GP to automatically find/choose the best individuals for the ensemble. This new GP approach combines *multiple* Pareto-approximated front members into a single composite genetic program solution to represent the (optimised) ensemble. This ensemble representation has two main advantages/novelties over traditional genetic algorithm (GA) approaches. Firstly, by limiting the depth of the composite solution trees, we use selection pressure during evolution to find small highly-cooperative groups of individuals for the ensemble. This means that ensemble sizes are not fixed *a priori* (as in GA), but vary depending on the strength of the base learners. Secondly, we compare different function set operators in the composite solution trees to explore new ways to aggregate the member outputs and thus, control how the ensemble computes its output. We show that the proposed GP approach evolves smaller, more diverse ensembles compared to an established ensemble selection algorithm, while still performing as well as, or better than the established approach. The evolved GP ensembles also perform well compared to other bagging and boosting approaches, particularly on tasks with high levels of class imbalance.**

I. Introduction

Classification with unbalanced data represents a major challenge in machine learning (ML) [1][2][3][4][5]. Data sets are unbalanced when the learning examples from one class are *rare* (the minority class), while the larger class makes up the rest (the majority class). Genetic Programming (GP) [6], like other ML techniques, can evolve "biased" classifiers when data is unbalanced, i.e., classifiers with strong majority class accuracies but poor minority class accuracies [1][2][7][8]. As the minority class typically represents the class of interest in many real-world problems, building classifiers with good accuracy on both classes is an important area of research [1][4][9][10].

The learning bias can occur because typical training criteria (such as the overall classification accuracy in the fitness function in GP) can be influenced by the majority class [1]. Approaches to addressing this involve sampling the data to artificially re-balance the classes [9][4], and/or cost adjustment within the learning algorithm to factor in importance of the minority class [5][11]. This paper focuses on the latter using evolutionary multi-objective optimisation (EMO) to handle the class trade-off during learning. In EMO, a *set* of the best trade-off solutions is evolved along the Pareto front in a single optimisation run by keeping the objectives separate/independent in the evolution [12]. This EMO approach has two main advantages over traditional sampling-based approaches. Firstly, it allows the original unbalanced data to be used "as is" during training (without *a priori* sampling). Secondly, the combined knowledge of the evolved classifiers along the Pareto-approximated front can then be used cooperatively in an ensemble for better generalisation compared to any single classifier alone.

This paper develops a two-step approach to evolving ensembles using GP. The first step uses EMO to evolve an accurate and diverse set of base learners by trading-off the minority and the majority class during learning, to form the ensemble. We show that the evolved ensembles are still vulnerable to the learning bias when the full Pareto set is used directly in the ensemble (due to the influence of biased members). The second step, which is the main focus this paper, proposes a novel GP-based ensemble selection approach to address this issue. Here we use GP to refine the ensemble by automatically finding/choosing the best ensemble members based on how well these members cooperate together. The new approach evolves *composite solutions* to represent an ensemble. These are single genetic programs which link to multiple base learners (in this case, Pareto-approximated front members).

The new GP approach to ensemble selection has two main advantages over traditional genetic algorithm (GA) approaches which fine-tune a large bit-string/weight-vector for ensemble selection. Firstly, favouring small composite solution trees during the evolution allows natural selection pressure to find highly-cooperative groups of individuals in the ensemble (via the few available positions in a GP tree). In contrast, fine-tuning a large bit-string using GA can be time-consuming, and may not reduce the original ensemble size unless sparsity (of the GA chromosome) is encouraged or the chromosome

U. Bhowan is with the Knowledge and Data Engineering Group, School of Statistics and Computer Science, Trinity College, Dublin, Ireland; and an associate member with the Evolutionary Computation Research Group, Victoria University of Wellington, New Zealand.

M. Johnston and M. Zhang are with the Evolutionary Computation Research Group, Victoria University of Wellington, New Zealand.

X. Yao is with CERCIA, School of Computer Science, The University of Birmingham, UK.

size is fixed *a priori* during optimisation. Secondly, exploring different function set operators in the composite solution trees allows the member outputs to be combined in different ways, thereby controlling how the ensemble computes its output (predicted class label). This incorporates the strategy of *how to* aggregate/combine the member outputs into the GP ensemble selection process, whereas most traditional GA approaches require that the ensemble aggregation strategy (usually majority voting) be configured *a priori* (and which remains fixed during the ensemble selection). This paper compares two ways to combine member outputs: traditional majority voting, and a new strategy using logical operators which give the ensembles greater "decisions making" abilities.

Our approach is evaluated on 12 real-world (binary) unbalanced data sets. We show that our GP approach evolves smaller, more diverse ensembles compared to an established approach, offline evolutionary ensemble selection algorithm [13] (which arguably represents the current state-of-the-art in GP), while still performing as well as, or better than the established approach. We also show that the GP ensembles perform well compared to several other bagging and boosting ensemble approaches from the literature using Support Vector Machines (SVM) and Naive Bayes (NB), and traditional "single-classifier" learning algorithms on the tasks.

The rest of this paper is organised as follows. Section II outlines the related work, and Section III outlines the MOGP approach to evolving the base learners from the literature. Section IV describes our new ensemble selection approach using GP. Section V presents the experimental results on the tasks, and Section VI makes further comparisons with other methods. Section VII concludes this paper.

II. Related Work

Approaches to classification with unbalanced data involve three main aspects. The first uses sampling techniques to artificially re-balance the classes before the training process such as *over-sampling* the minority class (to increase minority class representation) [14], synthetic minority class over-sampling (SMOTE) to artificially create "new" minority examples (from several known examples) [15], and *under-sampling* the majority class (to reduce majority class representation) [16]. The second involves cost adjustment within the learning algorithm to factor in the uneven class sizes during the training process. In GP, this typically involves adapting the fitness function to promote classifiers with good accuracy on both classes, e.g., by using fixed misclassification costs for the minority and the majority classes [17][8], or improved training criteria that are sensitive to the unbalanced classes [7][5]. The third aspect, which is the focus of this paper, uses *ensemble* methods where multiple trained classifiers are aggregated together to determine the final prediction. Ensemble learning uses aspects from both sampling and cost adjustment, and can be advantageous over canonical ("single-predictor") algorithms due to better generalisation from multiple classifiers [18][19][20][21]. In other words, ensembles are often more accurate than the individual classifiers that make them up [19][22].

For an ensemble to perform better than its individual members, the individual members must be *diverse*, i.e., not make the same errors on the same inputs [19][22][23][24]. Ensemble diversity allows different members to "specialise" on different parts of the input domain. For example, if one member S_i classifies an input instance X incorrectly and its output is uncorrelated to two other ensemble members, S_j and S_k, which produce the correct output on X, then a majority vote of all three members in the ensemble will ensure that X is correctly classified due to the influence of S_j and S_k. Otherwise, without good diversity, the ensemble members risk misclassifying all the same inputs together. In the example above, if S_j and S_k have highly-correlated outputs to S_i, each will classify X incorrectly and, in turn, the majority vote in the ensemble will also be incorrect; thereby providing no additional benefit in combining together the different classifiers.

In [19], a formal characterisation of this issue is provided, as well as several important techniques for promoting diversity. Two popular techniques (among others described in [19]) include bagging which samples the learning data into smaller (usually balanced) subsets to train the individual members [3][25][26][24], and injecting randomness into the learning algorithm to generate diverse members [18][20][27]. The latter is favoured in EMO due to its inherent stochastic and population-based nature [18][20][28][27]. Some EMO approaches (such as this paper) also use a diversity objective in fitness to promote cooperation between members, such as Negative Correlation Learning (NCL) [29][28] and Pairwise Failure Crediting (PFC) [20][18]. This allows the full training set to be used during fitness evaluation, compared to bagging and boosting which sample the training set. When data is balanced, these EMO approaches typically trade-off a solution's overall accuracy against their diversity (measured using the NCL [29] or PFC [18]). Other mechanisms to promote diversity in EMO include varying the size and/or structure of the learnt models [28][12], or partitioning of the training set [30][31].

A. Ensemble Approaches to Class Imbalance Classification

When data is *unbalanced*, ensemble systems are traditionally used in conjunction with sampling (e.g. under-sampling, over-sampling or SMOTE) to either create balanced bootstrap samples in bagging [3][24][2], or *re-balance* the training data prior to learning [4][26].

In [2], a taxonomy of the state-of-the-art in sampling-based ensemble learning is outlined. Bagging with under-sampling and boosting with SMOTE are recommended from among 20 other algorithms evaluated on 44 binary benchmark unbalanced tasks from KEEL [32]. In [26], a diversity term is added to the weight update rule in AdaBoost to measure the level of disagreement between classifiers. This method (and AdaBoost with SMOTE) is found to outperform traditional AdaBoost and AdaBoost using under- and over-sampling on 12 benchmark multi-class tasks. In [24], the bootstrap sample sizes are varied for diversity using bagging, where better ensemble diversity (measured by the Q statistic [33]) is found to be beneficial for the minority class but harmful to majority class on several (synthetic and benchmark) class imbalance

tasks. To address this, in [4], minority class diversity (measures using NCL) is traded-off against the overall accuracy in EMO (majority class is ignored) using neural network ensembles. While this method (and traditional NCL) performed better on the minority class than traditional bagging for eight benchmark (binary and multi-class) tasks, accuracy on the majority class suffered as a trade-off.

While effective, the main limitation of sampling to first artificially re-balance the original unbalanced data set is that the learnt models risk not learning/discovering the underlying *rarities* in the unbalanced data set, particularly if the sampling algorithm is not careful. This can lead to over-fitting, as shown in [26] and [2]. To address this limitation in GP, [20] combines the accuracy and diversity (measured separately) for each class into a single value, and trades these off against each other for all classes using EMO for cost adjustment (no sampling). However, [20] uses the *full* evolved Pareto-approximated front to form the ensemble. This can lead to sub-optimal ensemble behaviour due to the influence of some *biased* Pareto-approximated members in the ensemble. This paper uses the EMO configuration from [20] to evolve the ensemble base learners, but extends [20] by developing a new ensemble selection approach using GP (to optimise which members are chosen for ensemble) to further improve classification performance.

Cooperative co-evolution is also used for ensemble learning in GP [34][3]. In [35], grammatical evolution (GE) is used to co-evolve two populations (binary classifiers and "points" which are balanced bootstrap samples) for multi-class problems. In [34], "teams" of weak individuals that cooperate strongly together are also evolved for multi-class problems. However, a major difference between "teaming" and our work is that teaming combines many weak individuals; while our work in this paper focuses on ensemble selection of "strong" GP classifiers.

B. Ensemble Selection

Ensemble selection chooses which base learners to include in the final ensemble [13][21][36]. Most approaches assign a weight to each base learner which specifies whether the individual is included (or not) in the ensemble, or the individual's *contribution* in the ensemble [9][21][34][37]. Learning these weights is typically accomplished in three ways. The first (and simplest) method uses an individual's fitness (on the training set) as its weight [34][36]. A limitation of this method is that these weights do not reflect how well an individual cooperates with others in the ensemble. The second method co-evolves the base learner and their weight values in parallel [13][34]. The main limitation of this method is that co-evolving the weights with the base learners can be prone to noise, particularly in the early stages of the evolution (as shown in [13]). The third method invokes a secondary training phase to optimise the individual weights, typically using a genetic algorithm (GA) and a bit-string representation where each bit specifies whether a member is included or not in the ensemble [9][21]. This method is favoured over co-evolutionary approaches as it decouples the initial training phase (to learn the base classifiers) and the ensemble optimisation, allowing researchers to focus on one aspect at a time.

While effective, this method has two main limitations. Firstly, fine-tuning a weight-vector/bit-string can be time-consuming and difficult (i.e. many weights to configure), particularly when the pool of base classifiers is large [21]. Secondly, this method does not necessarily reduce the size of the ensemble unless sparcity (of the vector/string) is explicitly encouraged during optimisation, which is non-trivial, as shown in [37]. In [37], an ensemble pruning approach is developed using "expectation propagation" to approximate the member weights with Bayesian inference, while explicitly encouraging sparsity of the weight vector.

This paper develops a new GP approach to ensemble selection to avoid the limitations. We use a flexible size constraint on the GP-optimised ensembles and selection pressure to focus on finding small groups of highly diverse individuals for the ensemble. While some preliminary results of this new approach were recently presented at a local Artificial Intelligence conference [38], this paper substantially extends the work in [38] in four main areas. Firstly, this paper develops a new GP representation for the (optimised) ensembles (called Composite Logical Solutions), and compares this to the original representation in [38] (called Composite Voting Solutions). Secondly, this paper improves the GP search by evolving a *pool* of Composite Voting/Logical Solutions for a given set of base learners; whereas in [38], only a single Composite Voting Solution is evolved for a given run. Thirdly, this paper compares an alternative training configuration to evolve Composite Voting/Logical Solutions using a "selection set" to improve generalisation. Lastly, this paper compares the GP-optimised ensembles to several other ensemble learning approaches from the literature (these comparisons are not made in [38]), including an established ensemble selection approach (off-EEL [13]), and several bagging and boosting algorithms recommended in [2].

III. Evolving Base Learners using MOGP

This section outlines the initial EMO process for evolving the ensemble base learners which are used, as input, to the new ensemble selection approach proposed in Section IV.

A. Representing Genetic Program Classifiers

A tree-based structure is used to represent the genetic program solutions [6]. We use feature terminals (example features) and constant terminals (randomly generated floating point numbers), and a function set comprising of the four standard arithmetic operators, $+, -, \times$, and $\%$, and the conditional operator `if`. The $+$,$-$ and $\times$ operators have their usual meanings (addition, subtraction and multiplication) while $\%$ is *protected* division (usual division except that a divide by zero returns zero). The conditional `if` function takes three arguments and returns either the second argument if the first is negative, or the third argument otherwise. Each GP solution represents a mathematical expression that outputs a (floating-point) number for a given input (data example to be classified). This number is mapped to the class labels using zero as the

threshold, i.e., *minority* class if the classifier output is zero or positive, or *majority* class otherwise.

B. Multi-Objective GP (MOGP) Approach

In MOGP, the learning objectives are traded-off against each other, where a set (or *Pareto front*) of individuals is evolved along the objective trade-off surface in a single optimisation run. This is different to traditional ("single-objective") GP where the single fittest individual from the evolved population is returned from a given GP run. In this multi-Objective formulation, the minority and the majority classes are traded-off against each other for cost adjustment between the uneven classes, allowing the unbalanced data to be used directly in learning (without sampling). To accomplish this, MOGP uses a Pareto-based fitness function and Pareto dominance to rank the individuals in the population according to their objective performances. This ranking is important as it affects the way selection is performed if the different objectives are to be treated separately in the optimisation process. Pareto dominance between two individuals asserts that one individual will *dominate* the other if it is at least as good as the other on all the objectives but better on at least one objective [39], as shown below for individuals S_i and S_j, where x is an objective.

$$S_i \succ S_j \Leftrightarrow \forall x[(S_i)_x \geq (S_j)_x] \wedge \exists k[(S_i)_k > (S_j)_k]$$

1) Formulating the Objectives: In binary classification, the simplest MOGP formulation would trade-off the minority and the majority class accuracies against each other, as shown below.

$$f_c(S_i) = 1 - \frac{Err(c,S_i)}{N_c}$$

Where $Err(c, S_i)$ is the number of errors made by individual S_i on class c, and N_c is the number of training examples in class c. However, since the goal of MOGP is to evolve a pool of GP classifiers to form an ensemble, *diversity* between individuals must also be encouraged in the evolution. This is to ensure that the same errors are not made on the same inputs. To achieve this, a penalty function, pairwise failure crediting (PFC)[18], is incorporated into the learning objectives for diversity. This is shown by Eq. (1) which represents the final MOGP objective formulation.

$$f_c(S_i) = \frac{1-Err(c,S_i)}{N_c} + PFC_{c,i} \quad (1)$$

where

$$PFC_{c,i} = \frac{1}{Pop-1} \sum_{j=1,j\neq i}^{Pop} \frac{HD_c(S_i,S_j)}{Err(c,S_i)+Err(c,S_j)}$$

In Eq. (1), $PFC_{c,i}$ represents the diversity of individual S_i on class c where Pop is size of the population, and HD is the *Hamming distance* between the outputs of two individual S_i and S_j on the examples from class c. So HD is the number of outputs where the predicted class labels are different between two solutions. PFC values range between 0 and 1 where higher PFC values mean better diversity. The aim of the PFC function is to reduce the overlap of common errors between individuals in the population on a given class, to make the outputs of the different individual uncorrelated to each other [18]. Eq. (1) treats the accuracy and diversity on a given class as equally important in fitness where the higher the objective value, the better the accuracy and diversity. The PFC measure has been shown to promote good diversity between individuals in MOGP [20]. In [20], classifier diversity was shown found to be better using PFC in fitness compared to the widely-used NCL [29][28]. This is because PFC is a population-based measure, whereas other measures such as NCL evaluate diversity relative to the ensemble's output (requiring that the ensemble members are known *a priori* to obtain the ensemble output).

2) Pareto Fitness using SPEA2: Once each individual is evaluated on the objectives, MOGP uses the established SPEA2 [39] algorithm for Pareto dominance ranking to determine the final fitness values, as shown below.

$$fitness(S_i) = \sum_{j=1,S_j\succ S_i}^{Pop} Strength(S_j) \quad (2)$$

where

$$Strength(S_i) = |\{j|j \in Pop \wedge S_i \succ S_j\}|$$

In Eq. (2), the fitness of individual S_i is calculated as the sum of all *strength* values of all S_i's dominators, i.e., all other individuals in the population (Pop) that dominate S_i. An individual's strength value is the number of other individuals it dominates in the population. The lower the SPEA2 fitness value, the better the performance on the objectives. Non-dominated individuals in the population have the best fitness value of 0 since these solutions have no dominators.

The SPEA2 algorithm is chosen in MOGP for Pareto ranking (from other popular algorithms such as NSGA-II [40]) due to certain properties in SPEA2 which are well-suited to the objective formulation in Eq. (1) [20]. SPEA2 uses both dominance count (i.e. number of others that a given individual dominates) and dominance rank (i.e. number of others that dominate a given individual) in the fitness calculation. Dominance count is used as an individual's strength value, while dominance rank is used to find an individual's dominators. In contrast, other popular EMO algorithms such NSGA-II [40] tend to use either dominance count or dominance rank alone. This is important since these two aspects favour different regions of the Pareto frontier: dominance rank tends to reward exploration at the edges of the frontier, while dominance count rewards exploitation in the middle of frontier [41]. Due to these properties, it was shown in [20] that MOGP with SPEA2 is better at pushing the Pareto front outwards toward good performance on all objectives compared to NSGA-II, which found a wider *spread* of individuals along the whole of the frontier. The former is preferred using the objective formulation in Eq. (1) since this represents individuals with equally high accuracy and diversity on both classes, whereas the latter represents individuals with very high accuracy and diversity on one class alone.

C. MOGP Evolutionary Search Algorithm

A single MOGP run returns *multiple* evolved classifiers, i.e., the non-dominated set from the evolved population, as seen in

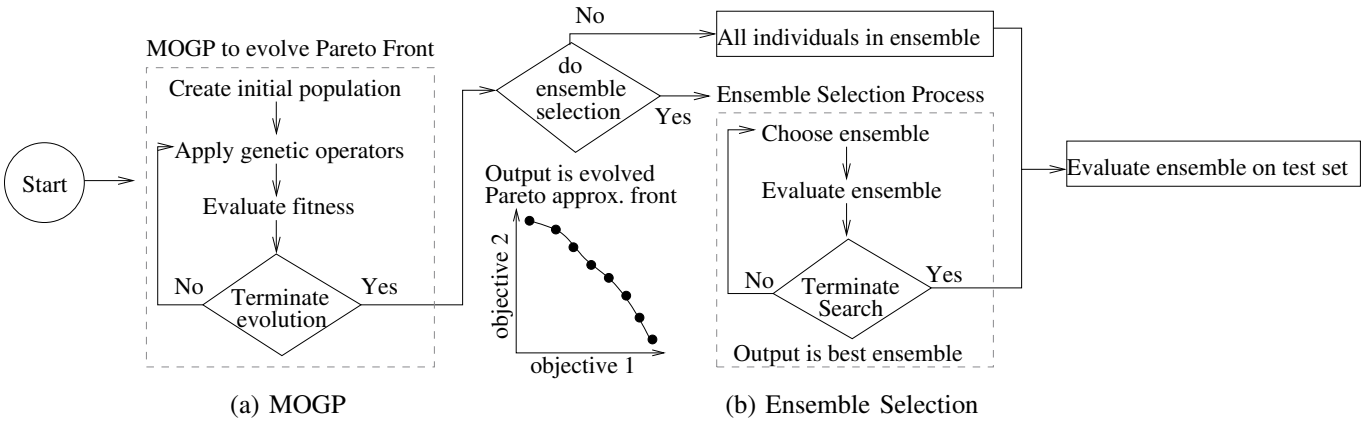


Fig. 1. Overview of MOGP and ensemble selection process.

Figure 1(a) for the two objectives. To facilitate the evolution of this non-dominated set, MOGP uses the SPEA2 search algorithm. In SPEA2, the parent and offspring populations are merged together at every generation [39]. This combined parent-child population is then sorted by fitness where the fittest individuals are copied into a new population, called the *archive population*. The archive serves as the parent population in the next generation, and preserves elitism in the population over generations. The offspring population at every generation is generated using the traditional crossover and mutation genetic operators using binary tournament selection.

The ramped half-and-half method is used for generating genetic programs in the initial population and for the mutation operator in MOGP [6] (similar to canonical GP). The population size is 500, crossover and mutation rates are 60% and 40%, respectively, and tournament selection is used with a tournament size of 2. These MOGP settings follow those recommended in [39]. Due to the nature of the MOGP approaches, and particularly the archive population, further elitism is not required [39]. The maximum program depth is 8 to restrict very large programs in the population, and the evolution is allowed to run for a maximum of 50 generations or until a solution with the best fitness is found.

IV. Composite Solutions for Ensemble Selection

This section outlines the second step in the classification process, i.e., forming the GP ensembles and the ensemble selection process, as seen in Figure 1(b). By aggregating the evolved MOGP classifiers into an ensemble where members can vote on class membership, we maximise the *combined knowledge* of these learned classifiers for better generalisation compared to any single classifier alone [20]. However, these GP ensembles can still be vulnerable to the learning bias if the full evolved non-dominated front is used directly in the ensemble due to the influence of biased members, as shown later (in Section V.B). We use ensemble selection here to find groups of individuals which cooperate very well together in the ensemble, ensuring good ensemble performance/generalisation on both the minority and majority classes. This ensemble selection process involves exploring and evaluating different formulations of individuals in the ensemble, to find the group which achieves the best ensemble performance. However, this

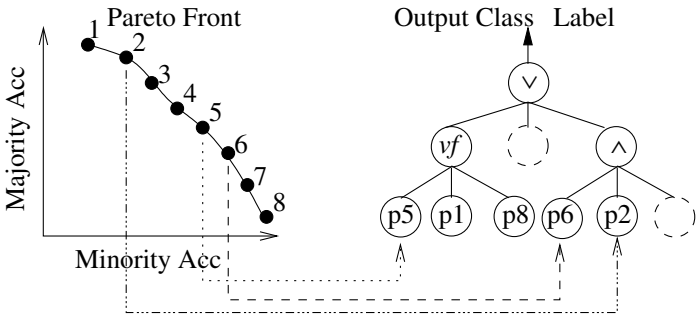


Fig. 2. Subset of evolved Pareto-approximated front (from a given MOGP run) forms a single GP composite solution representing the ensemble.

is a computationally expensive and time-consuming combinatorial problem, particularly if the non-dominated set or the training set is large. An exhaustive search to explore all combinations is impractical since this involves evaluating $2^T - 1$ different ensemble formulations (where T is the size of the non-dominated set), and every evaluation requires one pass through the training set.

A. Re-using GP for Ensemble Selection

To address this issue, we re-use GP for ensemble selection to quickly explore and evaluate different ensemble formulations. In this new approach, an (optimised) ensemble is represented as a GP *composite solution*. A composite solution is a single genetic program which links to multiple base learners (in this case, evolved Pareto-approximated front classifiers), as shown in Figure 2. Here the GP composite solution (right of Figure 2) is formed by linking several Pareto-approximated front members (left). Evolving a *population* of composite solutions using a traditional GP search is a means to efficiently explore and evaluate different ensemble formulations.

This GP approach to ensemble selection has two main advantages/novelties over other GA and GP approaches (as discussed in Section II.B). Firstly, limiting the maximum tree depth of the composite solutions during evolution forces natural selection pressure to find small and diverse groups of individuals for the ensemble (i.e. small high-fitness composite solution trees), via the few available positions within a composite solution tree. This means that the target ensemble sizes are not fixed *a priori* during ensemble selection (as required

in GA), but can vary depending on the diversity/performance of the base learners and the maximum tree depth.

Secondly, choosing different function set operators in the composite solution representation allows the outputs of the individual ensemble members to be aggregated and manipulated in different ways. This controls how the composite solution-based ensemble computes its output (final predicted class label). This is important as it incorporates the strategy of *how to* aggregate/combine the member outputs into the GP search and ensemble selection process, whereas in traditional GA approaches the ensemble aggregation strategy (usually majority voting) must be configured *a priori* and remains fixed during ensemble selection. This paper develops and compares two composite solution representations using different function sets. The first, Composite Voting Solutions (CSVote), uses the traditional majority vote operator to combine the member outputs. The second, Composite Logical Solutions (CSLogic), uses logical operators to aggregate the member outputs, thereby giving the ensemble greater "decisions making" abilities than traditional majority voting. One of the subgoals of this paper is to investigate whether the new logical operators in CSLogic is better for ensemble generalisation compared to traditional majority voting (used in CSVote).

The subsections below outline how the ensemble is represented as a GP composite solution, the two types of composite solutions (CSVote and CSLogic), and the evolutionary search parameters to evolve composite solutions.

B. Ensemble Representation as a GP Composite Solution

As mentioned above, an ensemble is represented as a single tree-based GP composite solution which links to multiple base classifiers, as seen in Figure 2. Here the terminal nodes in the GP composite solution tree represent a base classifier (from the evolved Pareto-approximated front), and the non-terminal nodes represent an aggregation strategy on how to combine the outputs of the base classifiers. The output of an evolved composite solution (when evaluated on a given input instance) is a class label, as determined by the ensemble members and the aggregation strategy. This tree-based representation allows us to re-use traditional GP search mechanisms such as using the genetic operators to easy interchange which base classifiers and aggregation functions are defined in the ensemble (to explore ensemble formulations), and the fitness function to evaluate these ensemble formulations and guide the evolutionary search toward high-fitness composite solutions. By measuring the geometric mean of a given composite solutions on the minority and majority class accuracy in the fitness function, ensembles with good and balanced classification accuracy on both classes can be evolved.

1) Composite Solution Terminal Set: To incorporate *multiple* base classifiers into a single GP composite solution tree, and to allow the genetic operators to easy interchange which base classifiers appear within a given composite solution, the following terminal set is used:

$$\{\emptyset, p_1, p_2, ..., p_T\}$$

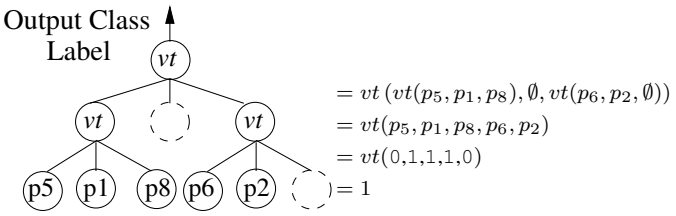
Here p_i represents a link to the ith base classifier from the Pareto-approximated front (of size T) from a given MOGP run, as shown in Figure 2. The symbol $\emptyset$ represents a $null$-valued terminal, used as a "blank" input argument to a particular function node. Allowing $null$-valued terminals in the composite solutions varies the number of base classifiers within any given composite solution, rather than insisting that every leaf-node in a composite solution maps to a base classifier. For example, the composite solution in Figure 2(b) uses five base classifiers even though there are seven leaf-nodes, since two leaf-nodes are $null$-valued terminals.

When a composite solution is evaluated on a given input instance, the ith base classifier representing terminal node p_i is first executed, and the *predicted class label* of this base classifier is taken as the return value of the terminal node. As there are exactly two classes in these data sets, binary values `0` or `1` are used to represent the two predicted class labels. As the raw output of a base classifier (when evaluated on a given input instance) is a real number, this number is mapped onto the two class labels using zero as the class threshold, i.e., `1` (minority class) if the base classifier's raw output is zero or positive, or `0` (majority class) otherwise.

This composite solution representation is chosen for two important reasons. Firstly, the leaf-nodes in the terminal set provide a mechanism to link multiple base classifiers into a single composite solution, irrespective of the type of underlying base learner. This means that the composite solutions can be used in conjunction with any underlying machine learning algorithm to generate the base learners, since the composite solutions take, as input, the predicted class labels of the base learners. Secondly, the function set operators (discussed below) provide a flexible mechanism to control how the outputs of the base classifiers are processed within a composite solution and thus, how the final output of the composite solution is computed. Exploring different function set operators will explore different stategies to combining/aggregating the member outputs in the ensemble. The two function sets to evolve the CSVote and CSLogic composite solutions are outlined below.

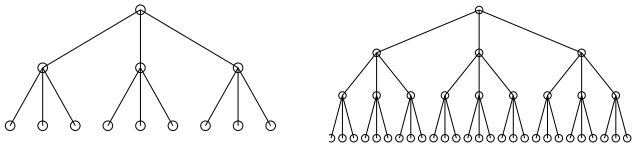
2) CSVote Function Set: To transform a composite solution into an ensemble where members vote on the class label, the function set consists of the single function vt. When vt is the *root* node in a CSVote tree, the function computes the majority vote of each base classifier within the tree. When vt corresponds to an internal (non-leaf) node in a CSVote tree, this function serves no purpose other than to join terminal (leaf) nodes or other vt nodes to the root node. In this case, these internal vt nodes simply pass each of its input arguments up the tree to the root node. This function takes exactly 3 input arguments, which can be other function nodes or terminals nodes. By only allowing the root to process the majority vote of all the base classifiers in the CSVote tree, each vote is treated as equally important during voting.

Figure 3(a) shows an example CSVote tree. Assuming that the predicted class labels of the five base classifiers in this tree, p_1, p_2, p_5, p_6 and p_8, are `1`, `0`, `0`, `1`, and `1`, respectively (when evaluated on a given input), Figure 3(b) shows how this CSVote tree is evaluated to obtain its output. When this CSVote tree is executed, the class labels of the base classifiers are taken as the terminal node values, and the two internal function nodes pass each of its input arguments up to the root



(a) CSVote Tree (b) Evaluating CSVote Tree

Fig. 3. Composite voting solution (CSVote) and evaluation of this CSVote tree to obtain its output (class label 1).



(a) Tree of depth 2 (b) Tree of depth 3

Fig. 4. Fully formed composite trees of depth 2 and 3.

node of the tree. When the $\emptyset$ terminal nodes are encountered, no value is passed up to the root node. The root node then computes a majority vote of these five class labels, and outputs a class label of 1 (denoting the minority class). In the case of a tie when the two classes have the same number of votes, the minority class label (i.e. 1) is chosen.

3) CSLogic Function Set: To transform a composite solution into a logical expression, a function set consisting of three functions is used: $\{\wedge, \vee, vf\}$. All three functions take 3 input arguments. The function $\vee$ represents a logical disjunction and returns 1 whenever one or more of its input arguments is 1 (0 otherwise). The function $\wedge$ represents a logical conjunction and returns 1 only if all of its input arguments are 1 (0 otherwise). The function vf represents a majority vote of all its input arguments, i.e., 1 is returned if two or more input arguments are 1, or 0 is returned if two or more input arguments are 0. In the case of a tie (e.g. when the three input argument are: 0, 1 and the $\emptyset$ symbol), the minority class label is returned (i.e. 1). This means that $\emptyset$-valued input arguments are ignored in all three functions. Unlike the CSVote approach where only the root node computes the majority vote of all base classifiers, each internal node in this configuration computes a new value (based on its input arguments) to pass up the tree. This provides the internal function nodes in the CSLogic trees some "decision making" ability when processing the inputs.

Using the CSLogic tree in Figure 2(b) and the same predicted class labels for the five base classifiers as discussed above, Eq. (3) below shows how this tree is evaluated to obtain its output (on an input instance).

$$\begin{aligned} & vf(p_5, p_1, p_8) \vee \emptyset \vee (p_6 \wedge p_2 \wedge \emptyset) \\ = \; & vf(0, 1, 1) \vee \emptyset \vee (1 \wedge 0) \\ = \; & 1 \vee \emptyset \vee 0 = 1 \vee 0 = 1 \end{aligned} \tag{3}$$

Here the $\emptyset$ symbol nodes are ignored when the CSLogic tree is interpreted. The output of this tree is the class label 1, determined as follows.

C. Evolutionary Parameters

1) Evolving Composite Solutions: The ramped half-and-half method is used to generate an initial population of composite solutions (similar to the MOGP configuration in Section V-A). The crossover, mutation and elitism rates are also 60%, 35% and 5%, respectively, and the tournament selection size is 7 (these settings are recommended in the GP literature [42]). To ensure that the training phase to evolve composite solutions is relatively fast, a population size of 300 is used, and the evolution is limited to 30 generations unless a composite solution with 100% accuracy on both classes on the training set is evolved (in which case the evolution is stopped). This corresponds to the *termination criteria* step in Figure 1(b) for ensemble selection. Since GP is a stochastic search algorithm, multiple GP runs (30) are executed for each input set of base learners (i.e. each evolved Pareto-approximated front from a given MOGP run), each using a different random starting seed. After 30 runs, the composite solution with the highest average accuracy on the minority and the majority classes (on the training set) is taken as the final optimised ensemble (for a given MOGP run).

These parameters are chosen as they generally performed well across all the tasks and for the different composite solutions based on preliminary experiments. These preliminary experiments were predicated on the knowledge that this ensemble optimisation step is only a further refinement process rather than a fully independent training process (like MOGP), and so requires a smaller population size and lower generation count than MOGP. However, it should be mentioned that the optimal parameter values may be different depending on the problem and representation of the composite solutions (e.g. CSVote and CSLogic).

2) Tree size: As the goal of the composite solutions is to discover *small* but highly cooperative subsets of base learners to represent the ensemble, a limit is imposed on the maximum tree depth allowed in the evolution. Smaller composite solutions are preferred (over larger trees) as these use *fewer* base classifiers in the ensembles and the selection pressure for the (limited) positions within the composite solutions is increased. This reduces the risk of individuals that do not positively contribute to the ensemble accuracy being included in the composite solutions. To achieve this, two maximum tree depth settings are compared in the evolution: 2 and 3. When the maximum tree depth is 2, a composite solution can include, at most, nine members since the function nodes take exactly three input arguments, as shown in Figure 4(a). When the maximum tree depth is 3, a composite solution can use, at most, 27 members, as shown in Figure 4(b). The composite solutions of depths 2 and 3 can contain *fewer* than 9 and 27 members, respectively, due to the $null$-valued terminal set symbol $\emptyset$, since this represents a "blank" input argument to a given function node. Composite solutions with only one terminal node (that is not the $\emptyset$ symbol) are permitted in the evolution. These are akin to a "winner-takes-all" strategy where the ensemble output depends on the output of a single

member.

This paper compares these two pre-defined maximum tree depths to study the effects of the different ensemble sizes on performance for the two GP configurations. While we expect that small but diverse ensembles (of depth 2) will perform better than larger ensembles (of depth 3) due to better cooperation between members, we also investigate the following issues related to ensemble size. Firstly, whether the maximum tree depth of 2 is too small for good ensemble generalisation compared to depth 3 for the CSVote and CSLogic approaches. Secondly, whether smaller CSLogic ensembles also perform better than larger CSLogic ensembles since the latter represents more complex and arguably more expressive logical expressions. An alternative to pre-defining the maximum GP tree depth would be to add a parsimony constraint in the fitness function which rewards smaller GP trees with better fitness values (such as [43]). This would also force the natural selection pressure to find smaller groups of individuals for the ensemble. However, allowing the evolution to automatically choose the ensemble size would not allow us to investigate the two above-mentioned issues, and so falls outside the scope of this work but will be considered in future work.

D. Extra Discussion: Multiple Classes

The two-phase GP framework proposed in this paper can be easily extended to perform classification with multiple classes by extending each of the two main GP steps as follows.

The MOGP approach (first step) can readily evolve a Pareto-approximated front along a multi-dimensional trade-off surface (three or more objectives), since each class (objective) is treated independently in the multi-objective optimisation, provided that a multiple-class classification strategy can translate the numeric genetic program output value to a set of class labels. Some multiple-class classification strategies include *static* class boundary determination (CBD) where fixed regions along the number-line are pre-assigned to each class [44][45], *dynamic* CBD where the class regions are dynamically learned using the program output values for each class [44], or *class enumeration* where the class labels are represented as terminal nodes and evolved along with the GP tree [46]. However, the performance of some MO algorithms may deteriorate when the number of learning objectives in a problem increases, due to a large number of candidate (trade-off) solutions along the objectives [47][48] (these works suggest some useful approaches to address the issue of scalability in MO learning). Alternatively, a problem-decomposition based approach can also be used to transform a multi-class task with k classes into k binary tasks [3][49]. A direct comparison between these approaches is beyond the scope of this work.

In the second step (ensemble selection), the CSVote composite solutions are also applicable with multiple classes, since the majority voting strategy in CSVote counts the number of votes for each class, regardless of the number of classes and class value, then returns the class with most votes. The functions used in CSLogic, in particular $\wedge$ and $\vee$, can be extended for multiple classes (since these assume binary class labels) using a new function (e.g. *isClass*) to convert each input argument (nominal class label) to a binary value (required by $\wedge$ and $\vee$). This new function *isClass* would take two inputs, a class label and target class label, and return a boolean value depending on whether a given class matches the target class (these inputs will be evolved along with the GP tree).

However, since the scope of this paper is on binary classification with unbalanced data, we leave the multi-class problems to the future work.

TABLE I
UNBALANCED CLASSIFICATION TASKS USED IN THE EXPERIMENTS (WHERE IR IS CLASS IMBALANCE RATIO).

Name	Description	Total	Minority Class Num	Minority Class %	IR	Feat. # Type
Ion	Ionosphere radar signal [50]	351	126	35.8%	1:3	34 $\mathbb{R}$
Spt	Tomography scan [50]	267	55	20.6%	1:4	22 B
Ped	Pedestrian Image [51]	24800	4800	19.4%	1:4	22 $\mathbb{R}$
Eco$_1$	Ecoli protein* (*eim*) [50]	336	77	22.9%	1:4	7 $\mathbb{R}$
Eco$_2$	Ecoli protein* (*epp*) [50]	336	52	15.5%	1:6	7 $\mathbb{R}$
Yst$_1$	Yeast protein* (*mit*) [50]	1482	244	16.5%	1:6	8 $\mathbb{R}$
Yst$_2$	Yeast protein* (*me3*) [50]	1482	163	10.9%	1:9	8 $\mathbb{R}$
Vow	Vowel* (*class0*) [32]	988	90	9.1%	1:10	13 $\mathbb{R}$
Led	LED display* (*class1*) [32]	443	37	8.4%	1:11	7 $\mathbb{R}$
Bal	Balance scale* (*classB*) [50]	625	49	7.8%	1:12	4 $\mathbb{Z}$
Aba	Abalone* (9 *v* 18) [32]	731	42	5.7%	1:17	8 $\mathbb{R}$
Shut	Statlog Shuttle* (c2 *v* c4) [32]	129	6	4.7%	1:20	9 $\mathbb{Z}$

V. EXPERIMENTAL RESULTS

This section presents the experimental results on the 12 tasks with essential discussions.

A. Unbalanced Data Sets

The experiments use 12 binary classification problems with unbalanced data, summarised in Table I. Tasks with * represent multiple-class classification problems which have been decomposed into binary problems where one class is chosen as the "main" (minority) class and some/all other classes form the majority class (the decomposed class name is given is parenthesis). In Ped, which is an image data set, pixel statistics corresponding to the mean and variance of the raw pixel values around local regions in the images are used as features (see [5] for details). The feature types are real ($\mathbb{R}$), binary (B) or integer ($\mathbb{Z}$).

The tasks in Table I are carefully selected to represent class imbalance problems of varying difficulty, dimensionality, size and (feature) types reasonably well. Minority classes range between 5% and 35% of total examples, and some tasks are more complex (easily separable) than others. The tasks also range from being well-represented (e.g. Ped) to sparsely-represented (e.g. Shut), and use different numbers of features. Half the examples in each class are randomly chosen for the *training* and the *test* sets, where both sets preserve the same (original) class imbalance ratio. In this way, the training and test sets are similar in size and more representative of the underlying knowledge, allowing meaningful performance results to be obtained in the experimental results. Tasks where the class distributions between training and test sets differ is considered beyond the scope of this work. These tasks also contain no missing attributes.

TABLE II

AVERAGE ENSEMBLE SIZES, GEOMETRIC MEAN ACCURACIES, AND INDIVIDUAL CLASS ACCURACIES OVER 50 RUNS (± STANDARD DEVIATIONS). *Rank* IDENTIFIES WHICH APPROACH HAS A STATISTICALLY SIGNIFICANTLY BETTER (TEST) GEOMETRIC MEAN ACCURACY ON A TASK. BEST RANK IS 1.

Task	Approach	Average Ensemble Sizes	Test Set							Training Set
			Geomean Accuracy %		Statistical Significance			Class Accuracy %		Geomean
			Best	Average	Rank	Beats	p-value	Minority	Majority	Accuracy %
Ion	**CSVote$_2$**	8.9 ± 0.4	95.5	**90.1 ± 2.2**	1	{3, 4}	p=5.1×10^{-31}	85.2 ± 4.1	95.5 ± 2.6	96.5 ± 1.8
	off-EEL	21.2 ± 7.4	95.9	89.8 ± 2.9	1	{3, 4}		83.6 ± 5.3	96.6 ± 2.8	98.4 ± 0.9
	FULL	28.1 ± 4.5	94.7	88.4 ± 3.0	2	{4}		84.9 ± 5.1	92.4 ± 6.5	97.5 ± 1.2
	CSVote$_3$	21.6 ± 6.3	94.6	86.6 ± 3.5	3	{4}		81.9 ± 5.4	91.9 ± 6.3	99.3 ± 0.6
	CSLogic$_2$	8.3 ± 1.0	93.0	86.2 ± 3.4	3	{4}		80.6 ± 5.8	92.4 ± 4.5	99.5 ± 0.5
	CSLogic$_3$	26.5 ± 9.2	88.8	60.9 ± 17.0	4	{}		67.6 ± 30.3	70.7 ± 32.6	99.4 ± 0.5
Spt	**off-EEL**	10.7 ± 5.2	79.0	**72.4 ± 3.0**	1	{2 − 4}	p=8.2×10^{-36}	66.3 ± 8.6	79.9 ± 6.7	94.7 ± 1.3
	CSVote$_2$	8.7 ± 0.8	78.5	72.2 ± 2.6	1	{2 − 4}		64.6 ± 6.3	81.0 ± 4.9	92.8 ± 3.0
	CSVote$_3$	17.7 ± 7.8	76.6	68.8 ± 3.5	2	{4}		55.7 ± 7.4	85.7 ± 4.0	96.0 ± 0.5
	CSLogic$_2$	9.0 ± 0.1	74.8	67.3 ± 3.1	3	{4}		54.9 ± 6.0	82.9 ± 4.0	95.3 ± 0.7
	FULL	27.3 ± 4.0	71.3	63.5 ± 3.7	4	{}		44.6 ± 5.5	90.8 ± 2.4	91.9 ± 2.5
	CSLogic$_3$	23.3 ± 10.8	72.7	43.7 ± 28.3	4	{}		41.1 ± 32.2	81.2 ± 23.4	96.4 ± 0.4
Ped	**CSVote$_2$**	8.7 ± 0.1	92.0	**89.4 ± 2.1**	1	{3}	p=1.2×10^{-8}	90.7 ± 2.3	88.1 ± 2.4	93.3 ± 2.6
	CSVote$_3$	22.7 ± 1.8	91.7	89.2 ± 2.2	1	{3}		88.1 ± 2.2	90.4 ± 3.0	91.8 ± 1.7
	off-EEL	55.2 ± 5.0	91.7	89.2 ± 1.5	1	{3}		90.6 ± 1.5	87.9 ± 1.4	91.3 ± 1.6
	CSLogic$_2$	9.0 ± 0.0	91.3	88.3 ± 2.4	1	{3}		87.8 ± 1.6	88.8 ± 3.1	88.0 ± 2.1
	FULL	71.6 ± 10.2	91.2	87.1 ± 2.6	2	{}		82.4 ± 4.6	92.1 ± 2.5	86.7 ± 1.9
	CSLogic$_3$	24.0 ± 3.5	88.5	85.9 ± 2.4	3	{}		85.0 ± 2.1	86.9 ± 2.4	88.1 ± 2.6
Eco$_1$	**CSVote$_2$**	7.6 ± 1.2	82.0	**77.8 ± 2.9**	1	{3 − 6}	p=2.0×10^{-38}	91.5 ± 5.0	66.2 ± 4.1	98.8 ± 1.2
	CSVote$_3$	18.3 ± 9.1	82.9	77.1 ± 4.0	2	{4 − 6}		93.7 ± 3.9	63.7 ± 6.2	99.2 ± 0.9
	off-EEL	7.9 ± 2.5	83.3	75.0 ± 4.4	3	{6}		90.9 ± 5.0	62.1 ± 7.0	99.9 ± 0.2
	CSLogic$_2$	8.4 ± 1.2	80.6	72.5 ± 7.0	4	{6}		88.2 ± 7.0	60.2 ± 10.1	99.9 ± 0.2
	FULL	8.3 ± 1.8	81.0	71.7 ± 6.0	5	{}		91.5 ± 5.1	56.8 ± 9.9	99.5 ± 0.5
	CSLogic$_3$	12.6 ± 4.6	82.2	60.9 ± 16.3	6	{}		82.9 ± 18.3	52.2 ± 26.1	99.9 ± 0.2
Eco$_2$	**CSVote$_2$**	8.6 ± 1.1	100.0	**99.9 ± 0.3**	1	{3, 4}	p=8.9×10^{-17}	99.9 ± 0.5	99.8 ± 0.4	98.8 ± 1.6
	CSVote$_3$	23.7 ± 6.5	100.0	99.8 ± 0.5	1	{3, 4}		99.8 ± 0.8	99.7 ± 0.6	98.5 ± 1.5
	off-EEL	10.6 ± 3.8	100.0	99.8 ± 0.4	1	{3, 4}		99.9 ± 0.5	99.6 ± 0.5	99.8 ± 0.4
	CSLogic$_2$	7.1 ± 1.5	100.0	99.4 ± 0.8	2	{4}		99.5 ± 1.3	99.3 ± 1.1	99.9 ± 0.2
	FULL	15.4 ± 2.7	100.0	98.8 ± 1.5	3	{4}		97.9 ± 3.1	99.6 ± 0.6	98.8 ± 1.5
	CSLogic$_3$	16.2 ± 4.2	98.1	84.2 ± 15.1	4	{}		80.6 ± 23.7	92.5 ± 12.5	99.9 ± 0.2
Yst$_1$	**off-EEL**	29.2 ± 9.3	77.6	**74.4 ± 1.1**	1	{3 − 4}	p=9.6×10^{-17}	70.6 ± 5.4	78.8 ± 5.6	85.3 ± 1.0
	CSVote$_2$	9.0 ± 0.0	76.9	72.9 ± 1.5	2	{4}		78.7 ± 5.8	66.6 ± 7.7	81.7 ± 4.2
	FULL	39.7 ± 5.1	75.6	72.1 ± 2.4	2	{4}		64.6 ± 4.8	82.5 ± 4.3	83.9 ± 1.1
	CSVote$_3$	17.5 ± 6.3	75.0	72.5 ± 1.3	3	{4}		67.8 ± 5.1	77.9 ± 4.9	86.6 ± 0.8
	CSLogic$_2$	8.9 ± 0.3	75.8	72.5 ± 1.7	3	{4}		64.6 ± 3.9	81.4 ± 3.1	87.8 ± 0.9
	CSLogic$_3$	28.9 ± 8.9	73.2	47.6 ± 17.6	4	{}		56.5 ± 36.8	65.9 ± 34.3	87.7 ± 0.9
Yst$_2$	**off-EEL**	17.2 ± 4.5	93.4	**92.1 ± 0.9**	1	{3 − 5}	p=3.0×10^{-34}	93.1 ± 2.7	90.8 ± 2.4	94.9 ± 0.7
	CSVote$_2$	9.0 ± 0.0	94.2	91.8 ± 1.1	2	{4 − 5}		92.8 ± 2.5	90.9 ± 2.4	94.1 ± 1.5
	CSVote$_3$	16.4 ± 5.5	93.2	88.6 ± 3.6	3	{4 − 5}		88.8 ± 7.9	90.2 ± 3.9	96.0 ± 0.6
	CSLogic$_2$	9.0 ± 0.4	93.7	88.3 ± 1.8	3	{4 − 5}		86.1 ± 4.3	93.4 ± 1.9	96.2 ± 0.6
	FULL	27.9 ± 3.7	91.6	85.0 ± 2.3	4	{5}		81.2 ± 5.0	95.5 ± 1.5	93.5 ± 1.1
	CSLogic$_3$	19.2 ± 7.2	91.1	72.4 ± 7.7	5	{}		76.0 ± 28.2	78.7 ± 23.7	96.1 ± 0.6
Vow	**CSVote$_2$**	9.0 ± 0.0	98.2	**84.6 ± 8.5**	1	{}	p=0.1621	74.9 ± 13.2	96.2 ± 7.5	99.4 ± 0.9
	CSLogic$_2$	6.2 ± 1.6	98.7	84.4 ± 9.2	1	{}		75.9 ± 14.0	95.3 ± 8.0	98.3 ± 1.3
	CSLogic$_3$	12.3 ± 3.8	97.9	84.2 ± 8.9	1	{}		75.2 ± 13.1	95.3 ± 8.4	98.3 ± 1.3
	CSVote$_3$	19.9 ± 9.0	97.9	82.7 ± 12.0	1	{}		73.1 ± 14.0	95.1 ± 13.2	99.7 ± 0.7
	off-EEL	13.2 ± 2.0	97.9	82.4 ± 9.0	1	{}		71.3 ± 13.3	96.2 ± 7.4	100.0 ± 0.0
	FULL	15.2 ± 1.6	95.4	79.4 ± 12.9	1	{}		69.5 ± 16.2	93.1 ± 14.8	99.9 ± 0.2
Led	**CSLogic$_3$**	13.2 ± 3.1	89.5	**81.8 ± 4.5**	1	{2}	p=3.7×10^{-7}	77.6 ± 8.2	86.9 ± 7.7	92.7 ± 4.0
	CSLogic$_2$	6.2 ± 1.2	89.5	81.5 ± 3.7	1	{2}		76.0 ± 8.3	88.1 ± 7.7	92.7 ± 4.0
	CSVote$_2$	9.0 ± 0.0	90.0	81.3 ± 4.2	1	{2}		74.9 ± 7.9	88.8 ± 7.3	97.6 ± 1.2
	CSVote$_3$	27.0 ± 0.0	89.5	80.1 ± 4.7	1	{2}		74.7 ± 8.9	86.4 ± 7.3	98.0 ± 0.9
	off-EEL	3.8 ± 2.8	88.5	79.3 ± 4.6	1	{2}		73.7 ± 8.8	86.0 ± 7.5	98.4 ± 0.4
	FULL	11.0 ± 2.0	85.5	76.3 ± 5.2	2	{}		70.6 ± 10.1	83.2 ± 6.3	98.3 ± 0.5
Bal	**CSVote$_2$**	8.9 ± 0.5	97.5	**84.3 ± 7.6**	1	{2}	p=9.8×10^{-31}	81.4 ± 12.2	86.2 ± 9.2	95.3 ± 2.4
	off-EEL	13.9 ± 2.5	99.3	83.3 ± 6.8	1	{2}		81.3 ± 9.4	87.9 ± 7.6	94.8 ± 3.2
	CSLogic$_2$	9.0 ± 0.0	93.6	80.3 ± 9.7	1	{2}		71.1 ± 14.9	91.7 ± 4.7	96.7 ± 1.9
	CSVote$_3$	12.6 ± 6.8	93.9	79.9 ± 7.7	1	{2}		74.3 ± 13.1	86.8 ± 7.6	97.2 ± 1.8
	FULL	20.8 ± 5.1	93.4	69.1 ± 12.5	2	{}		51.7 ± 18.4	95.4 ± 3.5	91.9 ± 3.5
	CSLogic$_3$	17.1 ± 7.5	97.5	55.1 ± 25.0	2	{}		54.5 ± 36.7	79.4 ± 23.5	98.4 ± 1.1
Aba	**off-EEL**	7.3 ± 3.4	85.4	**76.4 ± 5.1**	1	{4 − 5}	p=7.2×10^{-3}	71.3 ± 9.9	82.6 ± 7.2	92.5 ± 3.2
	CSVote$_2$	5.5 ± 1.3	84.6	74.5 ± 5.8	2	{5}		67.0 ± 10.2	83.8 ± 8.1	85.8 ± 8.4
	CSVote$_3$	10.2 ± 4.7	86.2	74.1 ± 6.5	3	{}		66.6 ± 12.6	83.8 ± 8.5	72.4 ± 17.5
	CSLogic$_3$	10.0 ± 7.5	86.2	73.5 ± 7.9	3	{}		64.4 ± 13.9	85.5 ± 7.6	89.6 ± 4.6
	CSLogic$_2$	6.5 ± 2.0	86.2	72.0 ± 10.2	4	{}		64.8 ± 17.1	82.9 ± 9.8	72.4 ± 17.5
	FULL	9.0 ± 3.6	83.6	69.7 ± 9.9	5	{}		63.8 ± 18.2	79.5 ± 13.0	87.8 ± 4.3
Shut	**CSLogic$_3$**	11.7 ± 2.8	100.0	**96.1 ± 6.7**	1	{3}	p=0.013	94.8 ± 12.2	97.8 ± 2.1	93.4 ± 11.8
	CSLogic$_2$	6.2 ± 1.5	100.0	95.8 ± 7.0	2	{}		94.1 ± 12.8	98.0 ± 1.9	93.4 ± 11.8
	CSVote$_2$	9.0 ± 0.0	100.0	95.5 ± 7.3	2	{}		93.5 ± 13.4	98.1 ± 2.0	96.7 ± 7.1
	CSVote$_3$	23.9 ± 8.5	100.0	95.0 ± 8.8	2	{}		92.8 ± 15.4	98.0 ± 1.9	96.4 ± 7.4
	off-EEL	8.1 ± 0.5	100.0	94.2 ± 7.7	2	{}		93.3 ± 16.5	96.0 ± 4.3	98.0 ± 5.7
	FULL	12.6 ± 0.7	100.0	91.1 ± 6.8	3	{}		87.3 ± 21.2	97.0 ± 3.1	99.6 ± 2.6

B. Main Ensemble Results

Table II shows the average ensemble results (± standard deviations) for the different ensemble approaches on the test and training sets over 50 runs. For convenience, the four composite solution approaches are denoted as CSVote$_2$, CSVote$_3$, CSLogic$_2$ and CSLogic$_3$, where the subscript (2 and 3) denotes to the maximum enforced tree depth. For a comparison, Table II also includes: (a) the *full* ensembles

without ensemble selection (denoted by FULL) where all non-dominate front members (for a given MOGP run) form the ensemble, and (b) another successful GP ensemble selection from the literature, off-EEL [13]. In off-EEL, the base learners are first sorted by their fitness values on the training set. Each individual is then iteratively copied into the ensemble where, at each step, this intermediate ensemble is evaluated. The members in the best-performing intermediate ensemble are then chosen for the final ensemble.

The geometric mean of the minority and the majority class accuracy is used as the primary evaluation criterion (and in the statistical significance tests discussed below) since this measure is sensitive to minority class when data is unbalanced. The geometric mean accuracy is calculated using Eq. (4) where A and B represent the accuracies of the two classes.

$$mean_{geometric} = \sqrt{AB} \tag{4}$$

In contrast, traditional performance measures such as the overall classification accuracy or error rate can be influenced by the larger majority class when data is unbalanced. Another advantage of the geometric mean is the desirable property that poorer accuracy on one class alone can give a lower overall mean than the traditional arithmetic mean, since the product of the component factors are computed. This means that biased results toward one class alone will be reflected with lower geometric mean values than arithmetic mean values.

To compare which approaches achieve statistically significant geometric mean accuracies on the *test set* than others, two statistical tests are performed. Firstly, an ANOVA F-test [52] is used to test the null hypothesis, i.e., no significant difference between the approaches over 50 runs (95% confidence level). The p-values from the F-test, shown in Table II, are extremely low for each task (except Vow). This indicates a statistically significant difference in performance between the different approaches (null hypothesis rejected), since smaller p-values mean greater statistical difference. Secondly, a *post-hoc* multiple comparisons test using Tukey's Honestly Significant Difference (HSD) [53] is then used to determine the statistically significant differences between group means. Tukey's HSD test conducts a series of pairwise comparisons[1] between the different ensemble approaches, and outputs a set of 95% confidence intervals for each comparison based on the *studentized range* distribution (similar to a Students *t*-test) [53]. See [5] for more details on Tukey's HSD test.

Table II summarises the outcomes of Tukey's multiple comparisons test by ranking and grouping together those ensemble approaches that achieve statistically *significantly better* geometric mean accuracies than others. This is shown by the ensemble *Rank* in Table II where the higher the rank, the better the ensemble's performance (rank 1 is best). Ensemble approaches with identical ranks imply very similar geometric mean accuracies on a particular task. The column *Beats* shows other groups (identified by their rank) that are statistically significantly worse on a particular task; the empty set denotes that a given approach is *not* statistically better than any other group. For example, in Table II for the Ion task, CSVote_2 and off-EEL both achieve the best rank of 1, meaning that both approaches are not statistically different to each other. However, both are significantly better than those with ranks 3 and 4 (CSVote_3, CSLogic_2 and CSLogic_3), but not significantly better than FULL (which has rank 2).

Table II shows that CSVote_2 and off-EEL generally achieved the best geometric mean accuracies on the test sets. CSVote_2 appears in the top-ranked position in six tasks, while off-EEL appears in the top position in the four other tasks (Spt, Yst_1, Yst_2 and Aba). In the remaining two tasks (Led and Shut), CSLogic_3 has the best geometric mean accuracy on the test set. These three approaches, in particular, CSVote_2 and off-EEL, clearly outperform the FULL ensemble (no ensemble selection) in most tasks.

Here CSVote_2 is statistically significantly better than FULL in seven tasks (Spt, Eco_1, Eco_2, Yst_2, Led, Bal and Aba), and off-EEL is statistically significantly better than FULL in six tasks (Spt, Eco_2, Yst_2, Led, Bal and Aba). Since CSVote_2 and off-EEL have *smaller* ensemble sizes than FULL in all tasks, this shows that better performance can be achieved using fewer (but more diverse) ensemble members chosen during ensemble selection.

CSVote_2 and off-EEL show no statistically significant difference in performance compared to each other in all tasks except Eco_1 (where CSVote_2 is significantly better than off-EEL). This is not surprising since both are greedy ensemble selection algorithms. However, ensemble sizes for CSVote_2 are *much smaller* than off-EEL on all tasks, particularly Ped, Yst_1, Ion and Yst_2. This shows that the ensemble members found using CSVote_2 are more diverse than those found using off-EEL on these tasks, since the smaller CSVote_2 ensembles generally perform as well as, or better than, the larger off-EEL ensembles. This highlights the usefulness of the evolutionary search to finding small diverse groups of individuals that cooperate well together in the ensemble. In addition to having better diversity, smaller ensembles also take less time to evaluate since fewer members must be evaluated to obtain the final ensemble output.

1) Why Ensemble Selection is Needed: The poorer performance by FULL (compared to CSVote_2 and off-EEL) is due to a slight performance bias (higher majority class accuracies than minority class accuracies) in nearly all tasks. While minority class accuracies for FULL are still relatively good in some tasks (e.g. Ion, Ped and Yst_2), these are much worse than CSVote_2 and off-EEL in some others (e.g. Spt, Yst_1, Bal and Aba). This shows that FULL contains more members which have a stronger majority class bias than the opposite case since these members *influence* the ensemble vote on these tasks. Only in one task (Eco_1) is the opposite behaviour be seen, where FULL contains more members that are biased toward the minority class. In contrast, CSVote_2 and off-EEL achieve much more balanced class accuracies, with noticeably better minority class accuracies compared to FULL in nearly all tasks.

Further analysis of the FULL ensemble results reveals that *more individuals* with a stronger majority class bias achieve non-dominated status as the evolution progresses over

[1]There are $\frac{k(k-1)}{2}$ total comparisons where k is the number of different ensemble approaches.

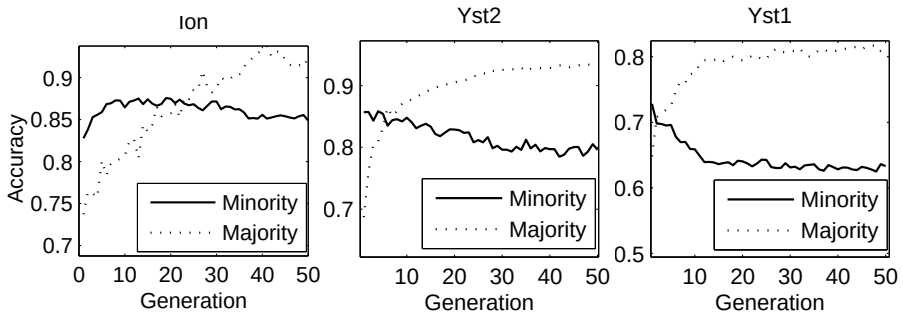


Fig. 5. FULL ensemble performance on the minority and the majority classes across 50 generations (average over 50 runs).

generations, compared to individuals with either a stronger minority class bias or middle-region individuals. This genetic drift in the population (toward non-dominated individuals biased toward the majority class objective) can be seen, to varying degrees, in Figure 5. This figure shows the minority class and majority class accuracies (on the test set) for FULL across 50 generations (averaged over 50 runs). These three figures (for Ion, Yst_1 and Yst_2) show that the majority class receives more votes from the different members than the minority class. The remaining tasks are omitted for brevity but reflects similar behaviours to Figure 5 (except Eco_2 where the two classes are balanced). These results show that ensemble selection can successfully exclude biased individuals from the ensemble, thereby improving ensemble accuracy on the important minority class.

2) Configuration of Composite Solutions: Table II confirms that smaller composite solution ensemble configurations, in particular $CSVote_2$, also generally perform better than larger composite solution ensemble configurations (such as $CSVote_3$) on most tasks. Here $CSVote_2$ and $CSLogic_2$ outperform $CSVote_3$ and $CSLogic_3$, respectively, in nearly all tasks (except Led and Shut), where $CSVote_2$ and $CSLogic_2$ both have smaller ensemble sizes than $CSVote_3$ and $CSLogic_3$. This suggests that limiting the maximum tree depth to 2, particularly for CSVote, is sufficient for allowing good ensemble generalisation (i.e. tree depth not too small), since these ensembles are generally better able to find small and highly diverse groups of individuals which cooperate well together in the ensemble compared to the maximum tree depth of 3.

However, it must be mentioned that these conclusions are only valid using the current GP parameter configuration (from Section IV.C), and that performances for $CSVote_3$ (and $CSLogic_3$) may be improved on these tasks using *different* GP parameters. For example, the number of generations in both $CSVote_2$ and $CSVote_3$ is limited to 30 to ensure that the evolution is relatively fast. This means that it may have been more difficult for GP to find good $CSVote_3$ solutions in 30 generations since these have many more possible combinations than $CSVote_2$. As null-valued terminals are allowed in the composite solution trees, we expect that $CSVote_3$ should also eventually be able to discover the same solutions as $CSVote_2$ in more generations. However, these results nevertheless show that the smaller $CSVote_2$ ensembles achieve good ensemble results when the goal of ensemble selection is to reduce ensemble size and improve diversity while also keeping this GP refinement process relatively fast.

TABLE III
AVERAGE MOGP ENSEMBLE TRAINING TIMES ($\pm$ STANDARD DEVIATION) (OVER 50 RUNS) IN SECONDS (S) OR MINUTES (M) IN ASCENDING ORDER.

Task	Times	Task	Times	Task	Times
Shut	26.3s $\pm$ 3.4	Spt	29.5s $\pm$ 1.6	Ion	30.4s $\pm$ 2.6
Eco_1	40.1s $\pm$ 3.5	Bal	45.1s $\pm$ 3.2	Led	47.3s $\pm$ 7.4
Yst_2	68.5s $\pm$ 5.9	Eco_2	71.2s $\pm$ 24.0	Yst_1	73.6s $\pm$ 4.5
Aba	90.3s $\pm$ 4.9	Vow	96.4s $\pm$ 9.2	Ped	17.7m $\pm$ 24.8

In Led and Shut, the larger $CSLogic_3$ shows the best (test) geometric mean accuracies, but $CSLogic_3$ is not statistically significantly better than the other composite solutions nor off-EEL in these two tasks. The reason $CSLogic_3$ achieves slightly better results than the other GP approaches on these tasks may be because the large, complex logical expressions represented by $CSLogic_3$ is more suited to difficult class imbalance problems (e.g. Led and Shut have a very sparse minority class representation from Table I with only 37 and 6 total minority class instances, respectively).

However, in the general case (as seen in the other 10 tasks), both $CSVote_2$ and $CSVote_3$ have better performance on the test set than $CSLogic_2$ and $CSLogic_3$. This suggests that the majority voting strategy (in CSVote, off-EEL and FULL) can be better for ensemble generalisation in these tasks than the logical operators in CSLogic, particularly in larger, more complex $CSLogic_3$ ensembles. This may be because the logical expressions may be overly sensitive to the training data on these tasks or that some evolutionary parameters can be improved (as mentioned above).

The evolutionary search to evolve composite solutions is also reasonably fast with a single GP run taking only between 0.2 and 5 seconds on the tasks (similar for off-EEL). This is approximately 2–10% of the training time to evolve a FULL ensemble in MOGP, as shown in Table III.

3) Ensemble Training Results: Table II also includes the geometric mean accuracies on the training sets to provide an indication of the overall effectiveness of the GP training process. These results show that the training performance for the ensemble selection approaches is good (achieving near-perfect accuracy) in nearly all tasks. None of the individual ensemble members (i.e. evolved GP classifiers), nor the *FULL* ensembles accomplishes this in training. However, some approaches such as $CSVote_3$, $CSLogic_2$ and $CSLogic_3$, have much higher geometric mean accuracies on the training set than the test set in some tasks (e.g. Spt, Eco_1 and Vow), suggesting that over-fitting has occurred. This may be due to the logic operators or evolutionary parameters in CSLogic (as previously discussed), or to the second training phase for

TABLE IV
GEOMETRIC MEAN ACCURACIES USING SELECTION SET OVER 50 RUNS. THE STATISTICALLY SIGNIFICANTLY BETTER PERFORMANCE ON A GIVEN SET IS SHOWN IN BOLD.

Task	Test Set (TE40)		Selection Set (SL20)	
	$CSVote_2$	Off-EEL	$CSVote_2$	Off-EEL
Ion	**90.4 ± 2.5**	82.9 ± 3.6	**99.6 ± 0.9**	95.8 ± 2.5
Spt	**72.9 ± 3.0**	67.8 ± 5.5	**90.0 ± 3.2**	86.7 ± 3.6
Eco_1	86.3 ± 3.2	84.6 ± 4.0	96.5 ± 2.3	94.5 ± 2.7
Yst_1	**78.1 ± 1.9**	75.6 ± 2.0	**78.2 ± 1.6**	75.3 ± 2.2
Yst_2	91.8 ± 1.0	91.1 ± 1.2	**93.7 ± 0.7**	91.4 ± 1.0
Vow	86.2 ± 4.5	83.1 ± 5.1	96.0 ± 1.1	96.3 ± 0.5
Led	82.0 ± 4.6	79.2 ± 3.4	89.4 ± 5.6	88.1 ± 2.0
Bal	**87.1 ± 5.5**	80.7 ± 7.8	94.3 ± 4.3	92.6 ± 4.8
Aba	74.5 ± 3.8	75.4 ± 6.1	87.8 ± 2.4	86.2 ± 7.4

ensemble selection (since the same training data is used to evolve the initial GP classifiers and composite solutions). Off-EEL and $CSVote_2$ have better generalisation (than $CSVote_3$, $CSLogic_2$ and $CSLogic_3$), as their training performance is not as good as the other approaches but their test performance is much better. In Ped, Eco_2 and Shut, the ensemble approaches generally show similar training and test performance, suggesting that good generalisation is achieved. These results nevertheless show that the composite solutions, in particular CSVote, may be particularly useful in optimisation problems or online learning which does not use an unseen test set.

C. Ensemble Selection using a "Selection Set"

To reduce over-fitting, we also investigate using an extra "selection" set in the secondary GP search to learn/evolve the composite solutions. In this alternative configuration, the original data sets are randomly split into three non-overlapping subsets which all preserve the same class imbalance ratio as the original data set: a *training* set containing 40% of the data instances (called TR40), a *test* set containing 40% of the data instances (called TE40), and a second training set (or *selection set*) containing the remaining 20% (called SL20). The TR40 training set is used in MOGP to train the base learners (not used in ensembles selection), whereas the SL20 selection set is used to evolve the composite solutions in the ensembles selection phase. Note that evolving composite solutions is a further refinement process rather than a fully independent training process from the original MOGP training process, and the goal is to avoid overfitting. The TE40 test set is then used to evaluate the ensembles on unseen input instances.

We chose this configuration over cross-validation for two main reasons. Firstly, to keep the "selection" set is relatively small, so that the TE40 test set is not substantially larger than the both training sets (i.e. TR40 and SL20). This ensures that the test set is still highly representative of the underlying concepts/rarities in the data. Secondly, to ensure that the TE40 test set is not substantially smaller than the original test set from the previous section (i.e. using the 50/50% split), so that meaningful ensemble comparisons can made across these two test sets. In contrast, when cross-validation is used with unbalanced data, the minority class can be under-represented in the test set, particularly when the number of folds is large. For example, 10-fold cross-validation on a task with only 10% minority class representation means a given fold will only contain one minority class instance if the different folds preserve the same class imbalance ratio.

Table IV shows the geometric mean accuracies on the selection set (SL20) and test set (TE40) for $CSVote_2$ and off-EEL over 50 runs. We focus on $CSVote_2$ and off-EEL only since these ensemble approaches *generally* perform the best on the tasks (as discussed in Section V-B). Note that the results on the selection set are presented (and not TR40) since the former is used in ensemble ensemble selection (and the latter in not). Ped, Eco_2 and Shut are omitted from Table IV since no over-fitting occurs on these two tasks (as seen in Section V-B3). In Table IV, the ensemble approach with a statistically significantly better geometric mean accuracy on a given set (using a 95% confidence level) is highlighted in bold.

Table IV shows that the TE40 test set results for $CSVote_2$ in Table IV are as good as, or slightly better than, the previous test results in Table II (using no selection set) in nearly all tasks (except Ion). Similarly, the ensemble training results on SL20 for $CSVote_2$ are also slightly lower compared to the previous ensemble training results (in Table II). This shows that using the selection set to evolve the $CSVote_2$ ensembles can reduce overfitting. However, this is not also the case for off-EEL, where performance on the TE40 test set is worse than the previous test results in Table II (using no selection set) on three tasks (Ion, Spt and Bal). As a result, $CSVote_2$ training is statistically significantly better than off-EEL in four tasks (on TE40) using the selection set. This shows the advantage of using GP to further refine/improve the ensembles. In future work we will explore other evolutionary parameters and training configurations to further improve ensemble performance.

VI. COMPARISONS WITH OTHER APPROACHES

This paper also compares the GP approaches to several other successful ensemble-based approaches for unbalanced data as recommended in [2], as well as traditional "single-classifier" approaches (where the learnt model is single classifier). Furthermore, both the ensemble-based and traditional "single-classifier" approaches are evaluated using GP, SVM and NB. Table V shows the geometric mean accuracies on the test sets (use the initial 50/50% training and test set split), and the training times in seconds (s) or minutes (m) for the different approaches[2]. Experiments using NB and SVM are generated using WEKA machine learning software [54].

The ensemble-based approaches include: (a) bagging with balanced bootstrap sampling using GP, NB and SVM as the base learners, and (b) AdaBoost [55] with under-sampling and SMOTE using NB and SVM as the base learners. Since bagging requires that the number of bootstrap samples B be configured *a priori* (where the final ensemble size is B), we use $B = 25$. This configuration generally gave the best performance from preliminary experiments using B values of 15, 25 and 50 on the tasks. Each balanced bootstrap sample uses all minority class instances and a random sampling of majority class instances (the sampling algorithm uses a different random starting seed for each sample). In AdaBoost, experiments are repeated 50 times (also using a different random starting seed) using 25 AdaBoost iterations for each

[2]No standard deviation values are available for AdaBoost (from WEKA) for the training times.

TABLE V
GEOMETRIC MEAN ACCURACIES ON THE TEST SET (AND TRAINING TIMES BELOW) FOR THE DIFFERENT APPROACHES ON THE TASKS. BOLD TEXT HIGHLIGHTS COMPETITIVE RESULTS COMPARED THE GP ENSEMBLE APPROACHES WHILE UNDERLINED TEXT HIGHLIGHTS POOR RESULTS.

Task	Ensemble Bagging			AdaBoost Methods				Traditional ("Single-Classifier") Learners				
	GP	SVM	NB	SVM Under	SVM SMOTE	NB Under	NB SMOTE	SGP ACC	SGP GM	SGP AUC	SVM	NB
Ion	**91.4 ± 3.4**	74.4	74.5	75.6 ± 3.3	78.4 ± 1.1	91.2 ± 3.3	90.9 ± 2.4	83.9 ± 5.5	83.6 ± 6.2	84.1 ± 6.1	93.1	75.1
Spt	**72.7 ± 3.4**	58.8	**73.8**	56.9 ± 10.8	58.8 ± 0.0	**73.0 ± 4.8**	73.5 ± 2.2	64.7 ± 4.8	68.5 ± 5.5	72.2 ± 10.0	59.1	74.4
Ped	61.0 ± 15.2	89.9	82.1	**98.3 ± 0.3**	**98.1 ± 0.2**	89.4 ± 1.6	89.1 ± 0.6	64.7 ± 4.8	86.6 ± 2.5	**92.1 ± 1.8**	70.5	82.5
Eco_1	73.4 ± 7.2	56.2	76.3	59.5 ± 9.1	75.9 ± 1.0	74.2 ± 5.5	77.3 ± 2.2	59.2 ± 3.8	68.5 ± 3.8	**79.4 ± 3.**4	75.1	74.6
Eco_2	95.0 ± 1.6	91.0	87.3	80.0 ± 15.6	81.5 ± 2.5	82.4 ± 8.5	68.9 ± 4.9	74.9 ± 1.9	88.6 ± 1.9	93.0 ± 2.9	87.6	51.8
Yst_1	73.8 ± 2.0	74.0	**76.2**	70.2 ± 3.9	66.9 ± 1.6	**75.0 ± 3.5**	71.9 ± 0.8	62.1 ± 2.4	70.7 ± 4.2	**77.1 ± 4.8**	56.5	64.7
Yst_2	**92.4 ± 1.5**	88.3	88.9	84.9 ± 2.9	79.0 ± 1.1	88.7 ± 2.9	82.8 ± 0.5	79.0 ± 2.2	89.4 ± 2.9	90.8 ± 2.9	75.4	80.8
Vow	**92.4 ± 8.6**	25.9	91.8	25.9 ± 0.0	25.9 ± 0.0	64.6 ± 3.7	64.9 ± 3.2	74.3 ± 12.0	83.3 ± 6.8	**91.9 ± 7.2**	25.9	87.5
Led	79.3 ± 11.8	73.1	78.8	73.1 ± 0.0	73.1 ± 0.0	78.3 ± 2.2	77.8 ± 1.2	74.0 ± 5.9	72.3 ± 6.6	78.8 ± 8.3	73.1	71.1
Bal	**86.2 ± 7.1**	63.3	38.5	59.0 ± 10.7	25.7 ± 2.1	44.2 ± 9.4	0.0 ± 0.0	29.8 ± 4.4	85.1 ± 11.5	82.3 ± 12.7	0.0	0.0
Aba	**80.5 ± 4.2**	66.7	64.6	60.8 ± 10.8	46.9 ± 3.1	70.7 ± 9.0	73.6 ± 7.4	38.1 ± 2.5	76.9 ± 8.0	**84.8 ± 7.4**	0.0	63.2
Shut	63.1 ± 13.9	57.7	**98.3**	78.6 ± 20.9	57.7 ± 0.0	93.3 ± 0.0	**98.5 ± 0.0**	91.5 ± 10.5	93.6 ± 7.0	**98.0 ± 0.5**	57.7	**98.3**
TRAINING TIMES												
Ion	**72.4s** ± 7.2	3.0s	2.0s	0.5s	1.0s	1.0s	1.3s	2.7s ± 0.8	2.8s ± 0.9	3.1s ± 0.9	<0.1s	0.1s
Spt	48.5s ± 4.4	2.0s	1.0s	0.9s	2.3s	0.6s	0.7s	2.3s ± 0.6	2.6s ± 1.0	2.8s ± 0.9	0.2s	0.1s
Ped	22.8m ± 3.6	5.7m	35s	31m	144m	5.3m	11.5m	5.4m ± 1.8	5.0m ± 3.0	5.8m ± 1.9	3.8m	20.1s
Eco_1	**87.1s** ± 7.7	2.0s	2.0s	0.7s	0.9s	0.5s	0.5s	4.1s ± 0.9	3.8s ± 1.2	71.8s ± 3.3	0.1s	0.1s
Eco_2	93.2s ± 2.4	2.0s	2.0s	0.9s	1.9s	0.4s	0.5s	4.4s ± 1.9	4.6s ± 1.3	70.9s ± 3.4	0.1s	0.1s
Yst_1	**4.4m** ± 3.8	4.0s	3.0s	5.3s	30.7s	0.6s	1.1s	13.5s ± 5.7	13.3s ± 4.7	13.3s ± 3.3	1.2s	0.3s
Yst_2	**3.8m** ± 2.8	4.0s	3.0s	3.2s	19.7s	0.7s	1.1s	11.5s ± 3.5	12.6s ± 7.9	15.2s ± 5.3	1.4s	0.1s
Vow	17.5s ± 1.9	4.0s	1.0s	1.5s	1.8s	1.7s	4.3s	2.8s ± 1.1	2.0s ± 0.8	64.2s ± 10.5	3.0s	8.0s
Led	15.7s ± 1.0	7.0s	<1s	0.7s	8.9s	0.5s	0.6s	3.3s ± 3.2	2.9s ± 3.3	27.7s ± 31.5	1.0s	1.0s
Bal	**1.9m** ± 0.3	2.0s	1.0s	1.0s	4.0s	0.4s	0.5s	5.1s ± 2	4.7s ± 1.5	5.0s ± 1.3	0.1s	0.1s
Aba	13.6s ± 2.0	4.0s	3.0s	2.1s	12.1s	1.5s	0.9s	1.9s ± 0.8	6.3s ± 3.2	35.0m ± 9.2	<1s	2.0s
Shut	24.0s ± 1.9	1.0s	1.0s	0.4s	0.5s	0.4s	0.4s	1.1s ± 0.8	1.2s ± 0.9	50.9s ± 51.2	<1s	1.0s

run. The SMOTE algorithm uses 5 nearest neighbours to create "new" minority examples. The SVM base learners use a sequential minimal optimisation algorithm with an RBF kernel and Gamma value of 10 (this value generally gave the best performance from experiments using 0.1, 1, 10, and 100).

"Single-classifier" GP (or SGP) is evaluated using three different fitness functions, where all SGP experiments are repeated 50 times. The first, *ACC*, represents the traditional measure, the overall accuracy. The second, *GM*, uses the geometric mean accuracy of the minority and the majority classes. The third, *AUC*, calculates the area under the Receiver Operating Characteristics (ROC) curve [56]. These SGP approaches use the same genetic program representation as MOGP (as outlined in Section III-A). This means that the same GP complexity constraints (e.g. a maximum depth of 8) are placed on the SGP classifiers, bagging-based GP classifiers, and the $CSVote_2$ (and off-EEL) ensemble members. Where possible, the SGP evolutionary parameters are kept the same as MOGP except for the tournament size which is 7 in SGP, and the mutation and elitism rates which are 35% and 5%, respectively (as these are recommended in the GP literature and elitism is not used in MOGP).

Table V shows that while the bagging and boosting approaches perform well compared to $CSVote_2$ and off-EEL (in Table II) in some tasks, these are much worse than $CSVote_2$ and off-EEL in some others. For example, while NB+bagging and NB+AdaBoost achieves competitive geometric mean accuracies compared to $CSVote_2$ and off-EEL on Spt and Yst_1 and very good results on Shut (but not better than the *best-of-run* 100% accuracy achieved by all GP approaches), these NB approaches perform worse than GP on Eco_2, Yst_2, Vow, Led, Aba and Bal. On Bal in particular (which has a high class imbalance ratio), nearly all other bagging and boosting methods have very poor performance with minority class accuracies as low as 0% (these are underlined in Table V). Similarly, SVM+AdaBoost outperforms $CSVote_2$ and off-EEL on Ped (highlighted in bold) but performs much worse on all the remaining tasks, particularly Vow where all the SVM-based approaches show very poor results (underlined). On this task in particular, SVM+AdaBoost incurs much longer training times than GP on this task (taking roughly 9 times longer). In contrast, $CSVote_2$ and off-EEL generally show good results (in Table II) across *all* tasks, suggesting that MOGP in combination with $CSVote_2$ and/or off-EEL for ensemble selection achieves good generality on these tasks. We expect the GP approaches developed in this paper to adapt well to new class imbalance tasks.

Interestingly, GP+bagging shows competitive results to $CSVote_2$ and off-EEL in two tasks (Spt and Yst_2), and even performs slightly better than $CSVote_2$ and off-EEL in four tasks (Ion, Vow, Bal and Aba). These six tasks for GP+bagging are highlighted in bold in Table V. In the remaining six tasks, $CSVote_2$ and off-EEL both outperform GP+bagging. This suggests that classification results for GP+bagging might be further improved on these tasks with ensemble selection using CSVote and/or off-EEL. This represents a useful new direction for future work. However, GP+bagging has longer training times than MOGP, which takes twice as long as MOGP (from Table III) in most tasks (e.g. Ion, Eco_1, Yst_1, Yst_2 and Bal underlined in Table V). This is because N different GP classifiers must be evolved using different (balanced) training samples to form a single ensemble. In contrast, all ensemble members are evolved simultaneously in *one* MOGP run using the full training set.

As expected, Table V also shows that the traditional "single-classifier" approaches which are not adapted for unbalanced

data sets (SGP+ACC, NB and SVM), generally perform poorly on the tasks with low minority class accuracies, particularly as the level of class imbalance increases. On some task such as Led, Bal and Aba (which have some of the highest class imbalance levels), these approaches cannot discriminate very well between the two classes at all, scoring as low as 0% minority class accuracy (underlined in Table V). The improved (class-sensitive) fitness functions in SGP (namely, SGP+GM and SGP+AUC) generally perform better on the tasks than the traditional approaches. However, SGP+GM does not perform better than CSVote_2 on any task, and SGP+AUC achieves competitive performance compared to CSVote_2 in only some tasks (e.g. Ped, Eco_1, Yst_1, Vow, Aba and Shut, shown in bold in Table V) but not others. This demonstrates that multiple GP classifiers working together in the CSVote_2 ensembles have better generalisation than "single-classifier" GP in these tasks, since the model complexities of the ensemble members in CSVote_2 and the evolved SGP classifiers are the same. This is a well-established concept in ensemble learning [18][19][20][21].

VII. Conclusions

The two main goals of this paper were to develop an ensemble learning approach for classification with unbalanced data using multi-objective GP, and develop a new ensemble selection strategy to prune the ensembles using GP. This is the first paper using GP for ensemble selection which combines multiple evolved GP classifiers along the Pareto-approximated front into a single *composite* genetic program solution to represent the ensemble. The two main novelties of this approach include using selection pressure in the evolution to quickly find small groups of highly cooperative individuals for the ensemble, and exploring different GP function sets to manipulate the outputs of the individual members to control how the ensemble determines its final classification decision. Two GP function sets are compared to evolve composite voting solutions (CSVote) and composite logical solutions (CSLogic).

These goals have been achieved by evaluating the classification performances of the full and pruned ensembles on 12 (binary) tasks, and comparing these results to an established ensemble selection algorithm (off-EEL [13]); bagging and boosting methods; and canonical GP, NB and SVM. We show that without ensemble selection, ensembles formed using the full Pareto-approximated front are vulnerable to the learning bias (due to the influence of biased members). As the ensemble sizes are automatically reduced during ensemble selection, ensemble performance improves on the tasks, particularly on the important minority class. The new GP approach to ensemble selection finds smaller and more diverse ensembles which perform as well as, or better than, off-EEL (and also better than canonical GP), particularly on tasks with high levels of class imbalance. Traditional majority voting (used in CSVote) is found to be better for good ensemble generalisation compared to the logical operators (used in CSLogic) in most tasks; whereas larger, more complex logical expressions represented by CSLogic performed slightly better than CSVote on two tasks with very sparse minority class representation. When an extra selection set is used in the composite solution evolution, performance improves for CSVote. The GP ensembles show good generality across all tasks, whereas the other bagging and boosting approaches (using NB and SVM) perform well on some tasks but also have difficulties on some others.

This paper focuses on ensemble performance and does not explicitly examine the diversity of the evolved ensembles. In future work we will analyse the relationships between diversity and ensemble performance, and investigate other techniques for promoting diversity between individuals in the evolution. We will also develop new composite solution representations, investigate using a parsimony objective in the fitness function to automatically evolve smaller composite solutions, refine the evolutionary parameters to further improve performance, and evaluate these approaches on multiple-class problems with unbalanced data in the future.

Acknowledgements

This work is supported in part by the Marsden Fund of New Zealand Government (under contract number VUW0806) administrated by the Royal Society of New Zealand, and a NSFC grant (No. 61329302). Xin Yao's research received funding from the European Union Seventh Framework Programme (grant agreement no. 270428). He was also supported by a Royal Society Wolfson Research Merit Award.

References

[1] G. M. Weiss and F. Provost, "Learning when training data are costly: The effect of class distribution on tree induction," *Journal of Artificial Intelligence Research*, vol. 19, pp. 315–354, 2003.

[2] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 4, pp. 463–484, 2012.

[3] A. McIntyre and M. Heywood, "Classification as clustering: A pareto cooperative-competitive GP approach," *Evolutionary Computation*, vol. 19, no. 1, pp. 137–166, 2011.

[4] S. Wang, K. Tang, and X. Yao, "Diversity exploration and negative correlation learning on imbalanced data sets," in *International Joint Conference on Neural Networks*, pp. 3259–3266, 2009.

[5] U. Bhowan, M. Johnston, and M. Zhang, "Developing new fitness functions in genetic programming for classification with unbalanced data," *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, vol. 42, no. 2, pp. 406 –421, 2011.

[6] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[7] J. Doucette and M. I. Heywood, "GP classification under imbalanced data sets: Active sub-sampling and AUC approximation," in *Proceedings of 11th European Conference in Genetic Programming (EuroGP 08)*, pp. 266–277, 2008.

[8] J. H. Holmes, "Differential negative reinforcement improves classifier system learning rate in two-class problems with unequal base rates," in *Proceedings of the Third Annual Conference on Genetic Programming*, pp. 635–644, 1998.

[9] N. Chawla and J. Sylvester, "Exploiting diversity in ensembles: improving the performance on unbalanced datasets," in *Proceedings of the 7th International Conference on Multiple Classifier Systems*, MCS, pp. 397–406, Springer-Verlag, 2007.

[10] G. Patterson and M. Zhang, "Fitness functions in genetic programming for classification with unbalanced data," in *Proceedings of the 20th Australasian Joint Conference on Artificial Intelligence*, vol. 4830 of *LNCS*, pp. 769–775, 2007.

[11] U. Bhowan, M. Johnston, and M. Zhang, "Differentiating between individual class performance in genetic programming fitness for classification with unbalanced data," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2802–2809, IEEE Press, 2009.

[12] Y. Jin and B. Sendhoff, "Pareto-based multiobjective machine learning: An overview and case studies," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 3, pp. 397–415, 2008.

[13] C. Gagné, M. Sebag, M. Schoenauer, and M. Tomassini, "Ensemble learning for free with evolutionary algorithms?," in *Proceedings of Genetic and Evolutionary Computation Conference*, pp. 1782–1789, ACM Press, 2007.

[14] R. Barandela, J. Sanchez, V. Garcia, and E. Rangel, "Strategies for learning in class imbalance problems," *Pattern Recognition*, vol. 36, no. 3, pp. 849–851, 2003.

[15] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[16] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 2, pp. 539–550, 2009.

[17] U. Bhowan, M. Zhang, and M. Johnston, "Genetic programming for classification with unbalanced data," in *Proceedings of the 13th European Conference on Genetic Programming*, vol. 6021 of *LNCS*, pp. 1–13, Springer, 2010.

[18] A. Chandra and X. Yao, "Ensemble learning using multi-objective evolutionary algorithms," *Journal of Mathematical Modelling and Algorithms*, vol. 5, pp. 417–445, 2006.

[19] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems, LNCS*, vol. 1857, pp. 1–15, Springer-Verlag, 2000.

[20] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 3, pp. 368 – 386, 2012.

[21] X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 28, no. 3, pp. 417–425, 1998.

[22] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, issue 2, pp. 123–140, 1996.

[23] E. K. Tang, P. N. Suganthan, and X. Yao, "An analysis of diversity measures," *Machine Learning*, vol. 65, no. 1, pp. 247–271, 2006.

[24] S. Wang and X. Yao, "Relationships between diversity of classification ensembles and single-class performance measures," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 206–219, 2013.

[25] S. Wang and X. Yao, "Theoretical study of the relationship between diversity and single-class measures for class imbalance learning," in *Proceedings of the IEEE International Conference on Data Mining Workshops*, ICDMW, pp. 76–81, 2009.

[26] S. Wang and X. Yao, "Multi-class imbalance problems: Analysis and potential solutions," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 42, no. 4, pp. 1119–1130, 2012.

[27] A. Chandra and X. Yao, "Divace: Diverse and accurate ensemble learning algorithm," in *Intelligent Data Engineering and Automated Learning*, vol. 3177 of *LNCS*, pp. 619–625, Springer, 2004.

[28] H. Chen and X. Yao, "Multiobjective neural network ensembles based on regularized negative correlation learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 12, pp. 1738–1751, 2010.

[29] Y. Liu and X. Yao, "Negatively correlated neural networks can produce best ensembles," *Australian Journal of Intelligent Information Processing Systems*, vol. 4, pp. 176–185, 1997.

[30] H. Abbass, "Pareto-optimal approaches to neuro-ensemble learning," in *Multi-Objective Machine Learning* (Y. Jin, ed.), vol. 16 of *Studies in Computational Intelligence*, pp. 407–427, 2006.

[31] H. Abbass, "Pareto neuro-evolution: constructing ensemble of neural networks using multi-objective optimization," in *IEEE Congress on Evolutionary Computation*, vol. 3, pp. 2074–2080, 2003.

[32] J. Alcal-Fdez, A. Fernandez, J. Luengo, J. Derrac, L. S. S. Garca, and F. Herrera., "Keel: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing*, vol. 13, no. 3, pp. 307–318, 2008.

[33] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine Learning*, vol. 51, pp. 181–207, 2003.

[34] M. Brameier and W. Banzhaf, "Evolving teams of predictors with linear genetic programming," *Genetic Programming and Evolvable Machines*, vol. 2, no. 4, pp. 381–407, 2001.

[35] A. McIntyre and M. Heywood, "Multi-objective competitive coevolution for efficient GP classifier problem decomposition," in *IEEE International Conference on Systems, Man and Cybernetics*, pp. 1930 –1937, 2007.

[36] D. W. Opitz and J. W. Shavlik, "Generating accurate and diverse members of a neural-network ensemble," in *Advances in Neural Information Processing Systems*, pp. 535–541, MIT Press, 1996.

[37] H. Chen, P. Tino, and X. Yao, "Predictive ensemble pruning by expectation propagation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 999–1013, 2009.

[38] U. Bhowan, M. Johnston, and M. Zhang, "Ensemble learning and pruning in multi-objective genetic programming for classification with unbalanced data," in *Proceedings of the 24th Australasian Joint Conference on Artificial Intelligence* (D. Wang and M. Reynolds, eds.), vol. 7106 of *LNCS*, pp. 192–202, Springer, 2011.

[39] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization," tech. rep., 2001. TIK-Report 103, Department of Electrical Engineering, Swiss Federal Institute of Technology.

[40] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast elitist multi-objective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2000.

[41] C. Coello Coello, G. Lamont, and D. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic & Evolutionary Computation Series)*. Springer, 2007.

[42] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Published via `http://lulu.com`, 2008.

[43] M. Zhang and U. Bhowan, "Program size and pixel statistics in genetic programming for object detection," in *Applications of Evolutionary Computing*, vol. 3005 of *Lecture Notes in Computer Science*, pp. 379–388, Springer Berlin Heidelberg, 2004.

[44] M. Zhang and W. Smart, "Multiclass object classification using genetic programming," in *Applications of Evolutionary Computing*, vol. 3005 of *LNCS*, pp. 369–378, Springer Berlin / Heidelberg, 2004.

[45] M. Zhang and W. Smart, "Genetic programming with gradient descent search for multiclass object classification," in *Genetic Programming*, vol. 3003 of *Lecture Notes in Computer Science*, pp. 399–408, Springer Berlin Heidelberg, 2004.

[46] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 12, pp. 1070–1077, IEEE Press, 2001.

[47] V. Khare, X. Yao, and K. Deb, "Performance scaling of multi-objective evolutionary algorithms," in *Proceedings of the second international conference on Evolutionary multi-criterion optimization*, EMO'03, pp. 376–390, Springer-Verlag, 2003.

[48] J. Knowles and D. Corne, "Quantifying the effects of objective space dimension in evolutionary multiobjective optimization," in *Evolutionary Multi-Criterion Optimization*, vol. 4403 of *Lecture Notes in Computer Science*, pp. 757–771, 2007.

[49] W. Smart and M. Zhang, "Using genetic programming for multiclass classification by simultaneously solving component binary classification problems," in *Proceedings of the 8th European conference on Genetic Programming*, vol. 3447, pp. 227–239, 2005.

[50] A. Asuncion and D. Newman, "UCI Machine Learning Repository," 2007. University of California, Irvine, School of Information and Computer Sciences. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[51] S. Munder and D. Gavrila, "An experimental study on pedestrian classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1863–1868, 2006.

[52] D. C. Hoaglin, F. Mosteller, and J. W. Tukey, *Fundamentals of Exploratory Analysis of Variance*. Wiley, 1991.

[53] J. W. Tukey, "Components in regression," *Biometrics*, vol. 7, pp. 33–69, 1951.

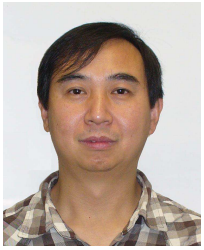
[54] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorations*, vol. 11 (1), 2009.

[55] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Proceedings of the Second European Conference on Computational Learning Theory*, EuroCOLT '95, pp. 23–37, Springer-Verlag, 1995.

[56] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, pp. 1145–1159, 1997.

Dr Urvesh Bhowan holds a BSc (Honours) degree and a PhD in in computer science from Victoria University of Wellington, New Zealand. After completing his study, Urvesh has held postdoctoral research roles in both Victoria University of Wellington and Trinity College Dublin, where he is currently based. His research interests include evolutionary computation, particularly genetic programming, classification with unbalanced data, data mining, natural language processing, and semantic search.

Dr Mark Johnston holds a BSc in mathematics and computer science and a PhD in operations research from Massey University, New Zealand. Since 2005 he has working at Victoria University of Wellington, New Zealand, and is now a senior lecturer where he teaches various operations research courses. He is a key member of the interdisciplinary Evolutionary Computation Research Group. His research is primarily in combinatorial optimisation and evolutionary computation, with particular interest in scheduling, routing, image analysis, feature reduction, genetic programming, particle swarm optimisation, classification and multiple-objective optimisation. He has over 80 refereed publications in international journals and conferences. Dr Johnston is a member of IEEE.



Dr Mengjie Zhang Mengjie Zhang received a BE and an ME in 1989 and 1992 from Artificial Intelligence Research Center, Agricultural University of Hebei, China, and a PhD in computer science from RMIT University, Australia in 2000.

Since 2000, he has been working at Victoria University of Wellington, New Zealand. He is currently Professor of Computer Science and heads the Evolutionary Computation Research Group. His research is mainly focused on evolutionary computation, particularly genetic programming and particle swarm optimisation with application areas of image analysis, multi-objective optimisation, classification with unbalanced data, and feature selection and dimension reduction for classification with high dimensions. He has published over 250 research papers in refereed international journals and conferences. He has been serving as an associated editor or editorial board member for five international journals (including IEEE Transactions on Evolutionary Computation and the Evolutionary Computation Journal) and as a reviewer of over fifteen international journals. He has been serving as a steering committee member and a program committee member for over eighty international conferences. He has supervised over thirty postgraduate research students.

Dr Zhang is a senior member of IEEE, a member of the IEEE Computer Society, the IEEE CI Society and the IEEE SMC Society. He is also a member of the IEEE CIS Evolutionary Computation Technical Committee, a vice-chair of the IEEE CIS Task Force on Evolutionary Computer Vision and Image Processing, and a committee member of the IEEE New Zealand Central Section. He is a member of ACM and the ACM SIGEVO group.

Dr Xin Yao (M'91-SM'96-F'03) is a Chair (Professor) of Computer Science and the Director of CERCIA (the Centre of Excellence for Research in Computational Intelligence and Applications), University of Birmingham, UK. He is an IEEE Fellow and a Distinguished Lecturer of IEEE Computational Intelligence Society (CIS). His work won the 2001 IEEE Donald G. Fink Prize Paper Award, 2010 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), 2011 IEEE Transactions on Neural Networks Outstanding Paper Award, and many other best paper awards at conferences. He won the prestigious Royal Society Wolfson Research Merit Award in 2012 and was selected to receive the 2013 IEEE CIS Evolutionary Computation Pioneer Award. He was the Editor-in-Chief (2003-08) of IEEE Transactions on Evolutionary Computation. He has been invited to give more than 70 keynote/plenary speeches at international conferences. His major research interests include evolutionary computation and ensemble learning. He has more than 400 refereed publications in international journals and conferences.