Developing New Fitness Functions in Genetic Programming for Classification With Unbalanced Data

Urvesh Bhowan, *Member, IEEE*, Mark Johnston, *Member, IEEE*, and Mengjie Zhang, *Senior Member, IEEE*

***Abstract*—Machine learning algorithms such as genetic programming (GP) can evolve biased classifiers when data sets are unbalanced. Data sets are unbalanced when at least one class is represented by only a small number of training examples (called the minority class) while other classes make up the majority. In this scenario, classifiers can have good accuracy on the majority class but very poor accuracy on the minority class(es) due to the influence that the larger majority class has on traditional training criteria in the fitness function. This paper aims to both highlight the limitations of the current GP approaches in this area and develop several new fitness functions for binary classification with unbalanced data. Using a range of real-world classification problems with class imbalance, we empirically show that these new fitness functions evolve classifiers with good performance on both the minority and majority classes. Our approaches use the original unbalanced training data in the GP learning process, without the need to artificially balance the training examples from the two classes (e.g., via sampling).**

***Index Terms*—Classification, fitness function, genetic programming (GP), unbalanced data.**

I. Introduction

CLASSIFICATION is a systematic way of predicting class membership for a set of examples or instances using the properties of those examples [1]. Given the abundance of information now being captured and stored digitally, systems that can automatically search for and identify valid and useful patterns in data for classification with little human intervention are fast becoming highly desirable. However, creating intelligent learning systems that reliably perform classification with a sufficient level of accuracy is difficult. Many real-world classification problems involve large numbers of learning examples, high dimensionality, and complicated relationships between class membership and example properties.

Genetic programming (GP) is a promising machine learning and search technique which has been successful in building reliable classifiers to solve a range of classification problems [2]–[6]. GP is an evolutionary learning algorithm which uses the principles of Darwinian evolution or natural selection to automatically *evolve* computer programs to solve problems. In GP, programs representing different solutions to a problem are combined with other programs to create new hopefully better programs; this process is repeated over a number of generations until a good solution is evolved [2].

In many real-world domains, it is not uncommon for data sets to have unbalanced class distributions [7]–[13]. This occurs when at least one class is represented by only a small number of examples (called the *minority class*) while other classes make up the rest (called the *majority class*). Recent research in the machine learning community has highlighted that using an uneven distribution of class examples in the learning process can leave learning algorithms with a performance bias, i.e., solutions exhibit high accuracy on the majority class(es) but poor accuracy on the minority class(es) [7], [14]. This is because traditional training criteria such as the overall success or error rate can be greatly influenced by the larger number of examples from the majority class [14]. As the minority class often represents the main class of interest in many real-world problems, accurately classifying examples from this class can be *at least* as important as, and in some scenarios more important than, accurately classifying examples from the majority class [11], [12].

Addressing this learning bias to find solutions with good accuracy on both the minority and majority classes has become an important area of research [7]. Techniques to address this issue involve two main aspects. The first involves transforming, or sampling from, the original unbalanced data set by creating an artificially balanced distribution of class examples for training, the so-called "external" approaches as the training data are adjusted, not the learning algorithm. Common external techniques include oversampling the minority class to boost representation [15], undersampling or editing of the majority class to decrease representation [16], or bagging and boosting where many balanced subsets of class examples are used in training [17]. While these approaches can be effective, they have certain disadvantages. Sampling techniques can add a computational overhead to the training process, lead to overfitting as potentially useful learning examples can be excluded from the learning process, and require *a priori* task-specific knowledge about the data to design a suitable sampling algorithm.

The second technique uses cost adjustment within the learning algorithm to factor in the uneven distribution of class

Manuscript received November 29, 2010; revised May 10, 2011 and August 2, 2011; accepted August 17, 2011. Date of publication September 26, 2011; date of current version March 16, 2012. This paper was recommended by Associate Editor Q. Zhao.

U. Bhowan and M. Zhang are with the School of Engineering and Computer Engineering, Victoria University of Wellington, Wellington 6140, New Zealand.

M. Johnston is with the School of Mathematics, Statistics and Operations Research, Victoria University of Wellington, Wellington 6140, New Zealand.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2011.2167144

examples in the original (unmodified) unbalanced data set, *during the training process* (the so-called "internal" approaches) [18]–[20]. In GP, cost adjustment can be enforced by adapting the fitness function. Here, solutions with good classification accuracy on both classes are rewarded with better fitness, while those that are biased toward one class only are penalized with poor fitness. Common techniques include using fixed misclassification costs for minority and majority class examples [21], [22], or improved performance criteria such as the *area under the receiver operating characteristic (ROC) curve* (AUC) [18], in the fitness function. While these techniques have substantially improved minority class performances in evolved classifiers, they can incur both a tradeoff in majority class accuracy and, thus, a loss in overall classification ability, and long training times due to the computational overhead in evaluating these improved fitness measures. In addition, these approaches can be problem specific, i.e., fitness functions are handcrafted for a particular problem domain only [11].

This paper focuses on the second technique and presents a GP approach to binary classification which takes advantage of the ability of GP to readily adapt the fitness function for cost adjustment. We develop several new generic GP fitness functions for class imbalance which utilize the unbalanced data sets *as is* in the learning process, requiring no prior knowledge about the problem domain. The new fitness functions aim to evolve classifiers with good classification ability on both the minority and majority classes and with faster training times than current GP approaches. We highlight the limitations of both the standard GP fitness function and two well-known current approaches for cost adjustment using the fitness function and present an empirical analysis of the performance of our new and current GP approaches across a range of real-world class imbalance problems. Our analysis focuses on the AUC of evolved solutions using the different GP methods. This is a useful measure of classification performance in class imbalance scenarios. We also compare our GP methods to other popular machine learning algorithms, namely, naive Bayes (NB) and support vector machines (SVMs), on the tasks.

The rest of this paper is organized as follows. Section II discusses the related work for classification with unbalanced data. Section III outlines the unbalanced data sets used in the experiments. Section IV outlines the GP framework for classification. Section V presents two alternative GP approaches for the class imbalance problem and discusses their limitations. Section VI proposes several new fitness functions. Section VII presents the full classification results and analysis using the different fitness functions. Section VIII concludes this paper and provides directions for future work.

II. Related Work

Recent research has highlighted that the class imbalance problem is a major obstacle in classifier induction using machine learning techniques. Many real-world problems have a naturally occurring imbalance in the representation of examples in each class. Examples include the following:

1) *fraud detection* such as network intrusion [11], telephone fraud [12], and credit card fraud [13];
2) *fault diagnosis* such as network troubleshooting [22];
3) *medical diagnosis* of rare conditions [21], [23], [24];
4) *bioinformatics* tasks such as protein classification [25];
5) *financial modeling* such as insurance approval [10] or bankruptcy prediction [26];
6) *object detection* such as target [8], face [9], or pedestrian [27] detection (in large images).

Much related work in developing methods to address the class imbalance problem involves two main approaches. The first involves transforming, or sampling from, the original unbalanced data set to create a balanced class distribution in training. These are known "external" approaches as the external training data are *rebalanced* while the learning algorithm remains relatively unchanged. The second approach uses various forms of cost adjustment within the learning algorithm to utilize the original unbalanced data "as is" in the training process. These are known "internal" approaches as the learning algorithm is adapted to factor in the uneven class distributions. Many approaches combine these two techniques, i.e., sampling methods along with cost adjustment in the learning algorithm [28], [29].

Much work in this area also focuses on gaining a better understanding of the nature of the class imbalance problem. This includes studying the effects of using different ratios of class distributions in the training process [13], [30] and investigating the influence of other factors during the learning phase [31], [32]. In [31], the influence of the level of class imbalance, complexity (class overlap), and training set size during training is investigated using three learning algorithms. Class imbalance is shown to be less of a hindrance in larger training sets and lower complexity problems. A similar conclusion is shown in both [32] and [33], where the authors find that performance degradation is not solely due to the level of class imbalance but is related to the degree of complexity.

A. External Approaches

Common sampling techniques include oversampling the minority class to boost representation by replicating known minority examples [29], [34] and undersampling the majority class to reduce majority class representation [15], [35]. However, as oversampling does not introduce any new information into the learning process and undersampling can discard potentially useful learning examples from the majority class, more robust sampling techniques, such as synthetic oversampling and editing, are also common [16], [36]–[38]. Synthetic oversampling of the minority class creates "new" minority examples by interpolating between several similar examples [36], while *editing* carefully removes noisy or atypical majority class examples (compared to random undersampling) [16], [37]. Other effective, although more complex, sampling approaches include random subset selection (RSS) and dynamic subset selection (DSS) [7], or a combination of these [39], [40]. In [39], a hierarchical two-tier sampling approach in GP is used to first sample "blocks" of training examples using RSS and then sample class examples within those "blocks" using DSS. In [40], DSS is used with a bias toward difficult-to-classify examples, while RSS is used with a bias toward minority instances.

Bagging and boosting algorithms are also effective in class imbalance problems [13], [33], [41], [42]. These approaches train multiple classifiers using smaller and usually balanced subsets of the original data, which are combined in an ensemble in the final classification step. These subsets usually contain all minority instances and the same number of randomly selected majority instances [41], [42] or focus on those instances not already accurately learned using weights to influence selection probability in boosting [33]. In [13], a "metaclassifier" is generated using four different learning algorithms in the pool of base classifiers, each trained on subsets of different distributions of class examples for fraud detection in e-commerce transactions. In [33], two new undersampling methods are developed to create balanced subsets used to train an ensemble of classifiers for boosting; these are compared to 13 other sampling and boosting approaches common in the literature across 16 benchmark University of California Irvine (UCI) tasks. Similarly, in [42], a new SVM-based undersampling approach iteratively collects support vectors using balanced subsets of training examples which are aggregated in the final classification step.

While sampling techniques are effective in improving minority class performance, they have major limitations. Sampling approaches can suffer from both *overfitting* as potentially useful learning examples can be excluded from the learning process and *poor generalization* as the learned models often do not capture the underlying rarities that occur in unbalanced data sets (if the training set is artificially balanced). Recent work comparing sampling techniques to cost adjustment across a variety of learning algorithms and problem domains shows that the latter can often outperform sampling methods in many scenarios [15], [31], [35] and that good results can be achieved using a combination of sampling and cost adjustment as opposed to sampling on its own [29]. Another limitation is that many sampling or data transformation techniques require *a priori* expert knowledge about the data, to design an appropriate sampling algorithm [11]. Sampling techniques can also add a computational overhead to the training process as, in most cases, the sampling algorithm must be applied repeatedly for best results and optimal coverage.

B. Internal Approaches

For the reasons described earlier, much work within the machine learning community focuses on cost adjustment within the learning algorithm to factor in the uneven representation of class examples. Common approaches include assigning different misclassification costs to incorrect class predictions [21], [22] or developing improved training criteria that are more sensitive to the unbalanced class distributions (compared to the standard overall accuracy or overall error rate). Improved training criteria include the average classification accuracy of the minority and majority classes [16], [29], [37], [43], the AUC [18], [44]–[46], statistical measures of accuracy such as Wilcoxon–Mann–Whitney (WMW) statistic (to approximate the AUC) [28], [47], or the F-measure widely used in information retrieval [20], [48].

Indeed, much research in this area studies the effects that different training criteria have on the learned classifiers in class imbalance scenarios [14], [20], [30], [48]. In [20], nine well-known training criteria, such as the AUC, F-measure, and mean square error (mse), are compared using a variety of learning algorithms; a new composite measure based on the average class accuracy, AUC, and rmse is found to be the most effective in training good solutions. In [48], the correlation between several well-known training measures is studied; ranking measures (e.g., AUC) are the most effective and the least correlated to both qualitative (e.g., accuracy) and probabilistic (e.g., log loss) measures when data are unbalanced, whereas these measures are all closely correlated when data are balanced. In three separate studies [14], [30], [47], the authors show that training using overall accuracy can find good solutions only when the training set has a balanced class distribution, whereas the AUC is better when class distributions are unbalanced.

In GP specifically, cost adjustment focuses on developing new fitness functions to reward solutions which have good accuracy on both classes with better fitness while penalizing those which have poor accuracy on one class with poor fitness [4], [11], [19], [49]. In [19], an adaptive fitness function is developed to periodically reweigh misclassification costs for hard-to-classify examples. This method improves overall classification performance compared to canonical GP on four benchmark UCI [50] tasks, but determining good initial costs is nontrivial. To address this limitation, a fixed-cost fitness function is used in [26] for bankruptcy prediction using data from Spanish companies generated between 1999 and 2000. Penalizing incorrect minority class predictions by a factor of the class imbalance ratio is shown to outperform previous approaches. In [11], three new fitness functions are developed for a multiclass network intrusion detection problem, based on differentiating between the classification accuracies of each minority class. Using real-world TCP dump data, these methods evolve classifiers with good accuracy on most of the minority classes, but the many free parameters in two out of the three fitness functions are considered a hindrance. Similarly, three new fitness functions for binary class imbalance problems are developed in [49]. These use the average classification accuracy of the minority and majority classes, except each has an increasing penalty for poor accuracy on one class only. Using two benchmark (from UCI [50]) and two artificial problems, the authors show that, not surprisingly, larger penalties lead to better minority class performance. However, neither the AUC of the evolved classifiers nor the statistical difference between the three fitness functions is fully explored.

In two recent GP approaches, the weighted sums of different measures are combined in the fitness function. The weighted average of the overall error rate, the mse, and a new measure of class separability similar to the AUC is used in [4] to evolve good solutions. In [28], the average of the geometric mean of the minority and majority class accuracies, and the WMW statistic, is used in fitness in conjunction with a sampling-based approach for faster training.

While these approaches for cost adjustment are effective, there are three main limitations. The first is that misclassification costs for incorrect class predictions must usually be determined *a priori* [11], [21], [22]. These can be problem specific and often require a trial-and-error process to determine an

appropriate set of costs for each class. The second is that better metrics in the fitness function (such as the AUC) can increase training times due to the computational overhead required to calculate these measures, particularly on large data sets [20], [47]. The third limitation is that many new fitness functions are handcrafted to suit a particular classification problem [11], [19]. These can require expert or *a priori* knowledge about the problem domain, whereas problem-independent fitness functions are more desirable.

Evolutionary multiobjective optimization (EMO) has also shown some success in this area [51], [52]. In [51], a multiobjective GP approach is developed using the accuracy of the minority and majority classes as the two learning objectives. A Pareto frontier of genetic program classifiers is evolved along this tradeoff surface, leaving the final choice for the end user, thus alleviating the need to predetermine the tradeoff *a priori* (as required in single-objective approaches). These Pareto frontier classifiers are then combined into an ensemble where members vote to classify unseen examples. In [52], an EMO approach using grammatical evolution is combined with bagging for tasks with multiple minority classes. Here, two populations are coevolved: one-class classifiers and "points" (subsets of balanced training examples which the classifiers act on). A winner-takes-all approach of the frontier solutions in these evolved populations is used to determine the final prediction.

III. Unbalanced Data Sets

In this section, we outline the six benchmark binary classification problems used in the GP experiments. These are taken from the *UCI Repository of Machine Learning Databases* [50] and the Intelligent Systems Laboratory at the University of Amsterdam [53]. Half of the examples in each class were randomly chosen for the *training* and the *test* sets. This ensured that both training and testing sets always preserve the same class imbalance ratio as the original data set. Note that no set contains missing attributes.

Ionosphere (Ion): Ionosphere (Ion) contains 351 recorded radar signals collected using high-frequency antennas targeting free electrons in the ionosphere. There are 126 "good" signals (35.8%) and 225 "bad" signals (64.2%), a class imbalance ratio of approximately 1 : 3. Signals were processed using an autocorrelation function returning two attributes per pulse, giving 34 real-number features $(\mathrm{F}_1{-}\mathrm{F}_{34})$ [50].

SPECT Heart (Spt): Single-proton emission computed tomography (SPECT) heart (Spt) contains 267 records derived from cardiac SPECT images. There are 55 "abnormal" records (20.6%) and 212 "normal" records (79.4%), an imbalance ratio of approximately 1 : 4. Each SPECT image was processed to extract 44 continuous features; these were further preprocessed into 22 binary features $(\mathrm{F}_1{-}\mathrm{F}_{22})$ [50].

Yeast ($\mathbf{Yst}_1$ and $\mathbf{Yst}_2$): Yeast ($\mathbf{Yst}_1$ and $\mathbf{Yst}_2$) contains 1482 instances of protein localization sites in yeast cells, with eight amino-acid sequences as numeric features $(\mathrm{F}_1{-}\mathrm{F}_8)$ [50]. The problem has nine classes, each with a different degree of class imbalance. We decompose this task into *many* binary classification problems using only one "main" (minority) class and *everything else* as the majority class. We use two "main" classes: Yst_1 has 244 examples from the mit class (16%) and an imbalance ratio of 1 : 6, and Yst_2 has 163 examples from the *me3* class (11%) and an imbalance ratio of 1 : 9.

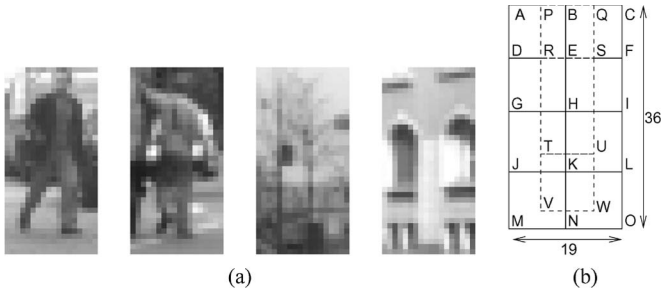


Fig. 1. (a) Example (left two) pedestrian and (right two) nonpedestrian images and (b)local image regions for extracting pixel statistical features.

Pedestrian Images (Ped): Pedestrian images (Ped) data set contains 24 800 portable Gray map (PGM) image cutouts, 19 × 36 pixels in size, of 4 800 pedestrian (19.4%) and 20 000 (80.6%) background images, an imbalance ratio of approximately 1 : 4. Example images are shown in Fig. 1(a). There are 22 pixel statistical features $\mathrm{F}_1{-}\mathrm{F}_{22}$ corresponding to the mean and variance of pixel values around 11 local regions in the image. These local regions correspond to the following rectangular regions: A-B-E-D, B-C-F-E, D-E-H-G, E-F-I-H, D-H-K-J, H-I-L-K, J-K-N-M, K-L-O-N, P-Q-S-R, R-S-U-T, and T-U-W-V, as shown in Fig. 1(b).

Balance Scale (Bal): Balance scale (Bal) contains 625 records generated to model psychological experiments. Each example is classified into three classes: the balance scale tipped to the right, left, or balanced. Of these, left (46%) or right (46%) makes up the vast majority; we combine these two classes into a single (majority) class called "unbalanced" (92%), using "balanced" as the minority class with 49 examples (8%). This corresponds to an imbalance ratio of approximately 1 : 12. There are four integer-based attributes corresponding to the left and right weights and the left and right distances $(\mathrm{F}_1{-}\mathrm{F}_4)$ [50].

IV. GP Framework for Classification

A tree-based structure is used to represent genetic programs [2]. We use feature terminals (example features) and constant terminals (randomly generated floating-point numbers) in the terminal set, and a function set consisting of the four standard arithmetic operators $+$, $-$, %, and $\times$, and a conditional operator `if`. The $+$, $-$, and $\times$ operators have their usual meanings (addition, subtraction, and multiplication), while % means *protected* division (usual division except that a divide by zero gives a result of zero). Each of these operators takes two arguments and returns one. The conditional `if` function takes three arguments. If the first is negative, the second argument is returned; otherwise, it returns the third argument. The `if` function allows a solution to contain a different expression in different regions of the feature space and allows discontinuities rather than insisting on smooth functions.

As the terminals and the return types of all the functions are numeric, the genetic programs represent mathematical expressions. For example, Fig. 2 shows the genetic program $(-(+\mathtt{F}_1\ \mathtt{F}_2)0.5)$; this solution represents the mathematical

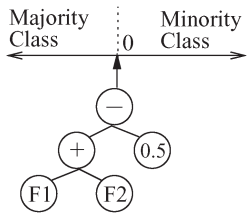


Fig. 2. Example of a (tree-based) genetic program representing the mathematical expression $(F_1 + F_2) - 0.5$. The output of the genetic program (when evaluated on a data instance) is mapped onto two class labels using zero as the class threshold.

expression $(F_1 + F_2) - 0.5$, where the arithmetic operators ($-$ and $+$) are the functions and F_1, F_2, and 0.5 are the feature terminals and the constant terminal.

As a mathematical expression, a classifier computes a single output value (floating-point number) for a particular data example that must be classified. This number is then mapped onto a set of class labels. A common mapping strategy for binary classification problems with two class labels uses *zero* as the class threshold, i.e., as example is assigned to the *minority* class if the classifier output is zero or positive, or the *majority* class if otherwise, as shown in Fig. 2.

An alternative strategy originally designed for classification with multiple classes [54] uses a *dynamically* assigned class threshold determined on a solution-by-solution basis during the evolutionary phase. In [54], the class threshold is the point of *least overlap* between the two class distributions. Recent work in GP, which compared the effectiveness of these two classification strategies in binary class imbalance problems [55], finds that either strategy can evolve good solutions, providing that a good fitness function is also used in the evolution. This is attributed to the evolved GP classifiers that are generally being able to "shift" their class predictions relative to the zero class threshold during evolution in binary classification. GP accomplishes this by tweaking the mathematical expressions representing the genetic program classifiers during the learning phase. For example, assume that the genetic program p representing the expression $(F_1 + F_2) - 0.5$ (Fig. 2) outputs values in the range [5, 10] when evaluated on examples from one class (e.g., class$_a$) and values in the range [10, 15] for the other class (e.g., class$_b$). If a mutation or crossover operation on the root node of p creates a new solution p' during evolution, where $p' = p - 10$, then the outputs of p' will lie in the range $[-5, 0]$ for class$_a$ and [0, 5] for class$_b$. The new genetic program p' would then represent the expression $((F_1 + F_2) - 0.5) - 10$, which can simplified to $(F_1 + F_2) - 10.5$. Many other ML techniques cannot easily achieve this during training. For this reason, we use the *zero-threshold* strategy in this paper.

A. Evolutionary Parameters

The ramped half-and-half method is used for generating programs in the initial population and for the mutation operator [2]. The population size is 500; initial GP experimental results indicate that this population size evolves solutions with good classification performance while keeping training times relatively low, whereas very large populations can substantially increase training times with little improvement in classification performance. Crossover, mutation, and elitism rates were 60%, 35%, and 5%, respectively, and tournament selection is used with a tournament size of seven. This configuration conforms to the recommended settings within the literature to balance exploration and exploitation during the evolution. The maximum program depth is eight to restrict very large programs in the population. The evolution runs for a maximum of 50 generations or is terminated early if a solution with *optimal* fitness (depending on the fitness function) is found. Initial GP experimental results indicate that increasing the maximum number of generations allowed (e.g., to 60 and 75) can lead to *overfitting*: Evolved solutions show better performance on the training set but no significant improvement on the test set. Note that fine tuning this configuration of evolutionary parameters is outside the scope of this work; this paper focuses on exploring the effects of different GP fitness functions on the evolved classifiers.

TABLE I
OUTCOMES OF A TWO-CLASS CLASSIFICATION PROBLEM

	Predicted Object	Predicted non-object
Actual Object	True Positive (TP)	False Negative (FN)
Actual non-object	False Positive (FP)	True Negative (TN)

B. Standard GP Fitness Function for Classification

A typical fitness measure in classification is the overall classification accuracy [4], [19], [56]. This is simply the number of examples correctly predicted by a classifier as a fraction of the total number of training examples. Using the four outcomes for binary classification shown in Table I and assuming that the minority class is the *positive* class, the overall classification accuracy can be defined by

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}. \quad (1)$$

Clearly, the standard GP fitness function for classification Acc considers all examples as equally important when calculating the overall accuracy and does not take into account the *smaller* number of examples in the minority class when data sets are unbalanced. In this scenario, Acc can favor the evolution of solutions *biased* toward the the majority class [14], [30], [47]. Biased classifiers have strong classification accuracy on one class but suffer poor accuracy on the other class. Using Acc in the fitness function, biased solutions can have high fitness yet rarely classify a minority class instance. For example, if a class imbalance problem only has 10% of all instances belonging to the minority class, a classifier with no discrimination ability between the two classes can score a high fitness by classifying *all* the instances as belonging to the majority class (e.g., 90% overall accuracy).

In this paper, our experimental results show that GP classifiers that evolved using the standard fitness function Acc demonstrate *significantly* poorer classification ability compared to a range of improved fitness functions which are more sensitive to the skewed class distributions. Using the six benchmark classification tasks with unbalanced data in Section III, we

show that new fitness functions that are sensitive to the smaller minority class can evolve classifiers with better discrimination ability between the class examples compared to the standard Acc. We also show that, on these tasks, this improvement in classification performance using the new fitness functions is greater in the tasks with high levels of class imbalance compared to the tasks with a better balance of class examples. Refer to Table II for details.

V. Current GP Fitness Functions and Limitations

To address the evolution of biased classifiers when data are unbalanced, there is much work in GP which focuses on adapting the fitness function to factor in the uneven distribution of class examples during the training process. In this section, we present two well-known alternative GP fitness functions for class imbalance and discuss the limitations of each approach and why they can be improved. These two fitness functions include the average classification accuracy of minority and majority classes [21], [28], [49], and the AUC [18], [28].

A. Average Class Accuracy in Fitness

The function $Ave(2)$ uses a weighted-average classification accuracy of the minority and majority classes in fitness. In (2), minority accuracy corresponds to the true positive (TP) rate, and majority accuracy is the true negative (TN) rate. The weighting factor is controlled by W, where $0 < W < 1$. When W is 0.5, the accuracy of both classes is considered as equally important in fitness. When $W > 0.5$, minority class accuracy will contribute more in the fitness function than majority class accuracy by factor W. Similarly, majority class accuracy will contribute more when $W < 0.5$

$$Ave = W \times \left(\frac{TP}{TP+FN}\right) + (1-W) \times \left(\frac{TN}{TN+FP}\right). \quad (2)$$

This weighting factor facilitates two important requirements in the fitness function. The first is when solutions with good classification accuracy on both classes must be evolved, and the second is when solutions with strong accuracy on one class over the other class must be evolved. However, choosing a good weighting factor *a priori* in the fitness function for a particular classification problem is hard. A suitable configuration can typically be problem specific and determined using a trial-and-error process. Many practitioners employ an equal weighting (i.e., $W = 0.5$) [23], [28], [57]. In this paper, the effectiveness of Ave is evaluated in both capacities, i.e., using an equal class weighting as well as different weighting configurations between 0.2 and 0.8 at intervals of 0.1.

While this fitness function can evolve solutions with a better balance of classification accuracy in both classes compared to the standard Acc, a major limitation of both Acc and Ave is that these measures represent a solution's classification performance at a *single* class threshold only. Varying this class threshold usually results in a different classification performance on the two classes. For example, if Fig. 3(a) represents the distributions of genetic program outputs for a given solution when evaluated on input instances from the majority and minority classes, then

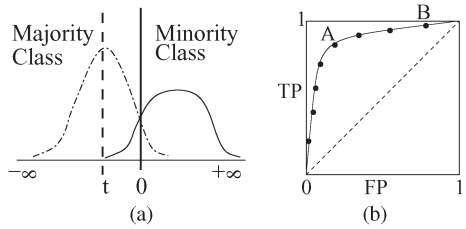


Fig. 3. Figure on the left shows an example of class distributions of genetic program outputs for minority and majority class instances, and two class thresholds (0 and t). The horizontal axis is the genetic program output, and the height of a point along either distribution is the frequency of class instances which evaluate the same output. The figure on the right shows an example ROC curve, where points A and B correspond to the performance of the genetic program using class thresholds 0 and t.

the classification accuracy of this solution for each class will be different when the class threshold is zero or t. When the class threshold is t, instances are labeled as "minority class" if the classifier output is $\geq t$ (otherwise, "majority class"). Selecting a suitable class threshold to represent a classifier's performance is nontrivial.

Due to this limitation, fitness function Ave often does not consistently evolve solutions with improved classification performance compared to the standard Acc. In this paper, we show that GP classifiers that evolved using Acc and Ave show no significant difference in performance in exactly half of the six benchmark classification tasks, particularly when the imbalance ratio in a classification task is not sufficiently large. Refer to Table II for details.

B. AUC in Fitness

The AUC is a useful measure of classification performance that is not dependent on a single class threshold, unlike the traditional accuracy-based measures (such as Acc and Ave) [44]. ROC curves were originally used in signal detection theory to characterize the tradeoff between hit rate and false alarm rate over a noisy channel [18]. It has since been widely utilized in the machine learning community to both visualize and measure the performance of a classifier across *varying* class thresholds [45], [58]–[60].

In binary classification, each operating point on an ROC curve, such as Fig. 3(b), represents the classification accuracy of the two classes at a single class threshold. In Fig. 3(b), the vertical axis represents the TP rate (i.e., *positive* or minority class accuracy), and the horizontal axis represents the false positive (FP) rate (i.e., error rate of the *negative* or majority class). The ROC curve is generated by varying the class threshold which biases the final classification decision and evaluating the classifier to obtain the accuracy of the two classes. The bottom-left point on the ROC curve represents the classifier performance when all instances are assigned to the *negative* or majority class (TP rate or minority accuracy is 0%), whereas the top-right point represents the performance when all instances are assigned to the *positive* or minority class (majority accuracy is 0%). The top-left point on the ROC plot represents perfect classification (all examples are assigned with correct class labels), and the line $x = y$ represents randomly guessing the class label.

By taking into account the TP and FP rates at multiple-class thresholds, the AUC effectively measures the classification performance of a solution across different class thresholds [44]. The better the ROC curve, the better the classifier's ability to discriminate between the two classes at different class thresholds. The AUC approximates this classification *ability* as a single-figure measure between zero and one; this can also be thought of as the probability that an example from the positive (minority) class is correctly predicted across different class thresholds [44].

A simple technique to approximate the AUC for a given classifier computes the sum of the areas of individual trapezoids[1] fitted under the ROC points [18]. Using this technique, the fitness function Auc_F can be defined using (3). In (3), N is the number of class thresholds to be used in the ROC curve, where the greater the number, the better the AUC approximation, and TP_i and FP_i are the accuracy of the two classes (i.e., true and FP rates, respectively) at the class threshold i

$$Auc_F = \sum_{i=1}^{N-1} \frac{1}{2}(FP_{i+1} - FP_i)(TP_{i+1} + TP_i). \quad (3)$$

We use the following technique to generate an ROC curve for a given genetic program classifier.[2] A solution is evaluated on all input examples, and the corresponding genetic program outputs are sorted in ascending order. The first class threshold is initialized as the smallest genetic program output, and the classifier is evaluated to obtain the TP/FP values. This threshold is then incremented by some amount T, and the solution is reevaluated to obtain the new TP/FP values; this process is repeated until the largest genetic program output is reached. To calculate the *full* AUC or Auc_F, T is incremented as the next genetic program output that is greater than T; this means that all possible class thresholds are used in the computation, allowing for a highly accurate estimation.

Due to the AUC being threshold independent, it is invariant to unbalanced class distributions and is widely used in class imbalance scenarios to evolve solutions with good classification ability [14], [30], [47]. In this paper, our experimental results show that the classification ability of GP solutions that evolved using the fitness function Auc_F (3) is always superior to the solutions that evolved using either of the traditional accuracy-based measures Acc (1) and Ave (2) on the six tasks. Refer to Table II for details.

However, a major limitation of using Auc_F in the fitness function is the increased training times. This is due to the computational effort required to construct an ROC curve: Each classifier must be evaluated on all fitness cases at every distinct class threshold to obtain the corresponding TP/FP values. Our experimental results show that GP training times using Auc_F can take approximately five to eight times longer than the two other fitness functions on the smaller data sets (fewer than 1500 training examples), while taking approximately 15 times longer on the largest data set (more than 10 000 training examples); see Table II for details. This represents a substantial increase in training time.

Two alternative techniques to approximate the AUC in the fitness function include using fewer class thresholds in (3) to limit the number of evaluations of all fitness cases (i.e., to reduce training time), and a statistical approximation based on the well-known WMW statistic [28], [44], [47]. In the first case, seven distinct class thresholds are recommended for a fast and accurate approximation to the full AUC in [44]. Therefore, we define a new fitness function Auc_E which uses exactly seven class thresholds spread uniformly over the range of genetic program outputs for a given solution. Naturally, this faster approximation will have a lower precision than the full AUC (Auc_F).

For WMW-based estimation, this statistic uses a series of pairwise comparisons between the genetic program outputs (when evaluated on examples from the two classes), effectively measuring the ordering of minority to majority class outputs. The WMW statistic as a fitness function Wmw is calculated using (4), where P_i and P_j represent the outputs of a genetic program when evaluated on an example from the minority (Min) and majority (Maj) classes, respectively. The indicator function I_{wmw} in (4) returns 1 if $P_i > P_j$ and $P_i \geq 0$ or 0 if otherwise; this enforces both the zero class threshold and the required ordering of minority and majority class outputs in evolved solutions. The denominator ensures that Wmw returns values between 0 and 1, where 1 indicates optimal AUC and 0 indicates poor AUC

$$Wmw = \frac{\sum_{i \in Min} \sum_{j \in Maj} I_{wmw}(P_i, P_j)}{|Min| \times |Maj|}. \quad (4)$$

In this paper, we investigate the use of all three techniques to evaluate the AUC in fitness, i.e., using the full AUC (Auc_F), the fast AUC approximation (Auc_E), and the WMW-based approximation (Wmw).

C. Summary of Limitations of Current Fitness Functions

The following summarizes the main advantages and limitations of the two accuracy-based fitness functions Acc and Ave and the three AUC-based fitness function discussed earlier, and highlights what aspects of these need improvement.

The measure Ave (2) which uses the average classification accuracy of the minority and majority classes is sensitive to the smaller minority class in class imbalance scenarios (unlike the standard Acc) but does not consistently show very good classification results on the classification tasks (refer to Table II). How can this widely used measure be improved to evolve better performing classifiers when data are unbalanced?

The GP fitness function using the full AUC (Auc_F) can evolve superior-performing classifiers compared to the traditional measures Acc and Ave in class imbalance scenarios. However, Auc_F can also incur substantially longer training times than the other two fitness functions, due to the computational cost in evaluating the AUC during fitness evaluation.

[1]The area of a trapezoid is $(1/2) \cdot w \cdot (h + h')$, where w is the width, and h and h' are the heights of the sides of the trapezoid [18].

[2]The method of varying the class threshold is different, depending on the learning algorithm. For example, neural networks and NB usually output a probability that an example belongs to a particular class. In this case, the class thresholds would vary between values 0 and 1.

Can new and faster (threshold-independent) evaluation measures which approximate the AUC in fitness be developed to evolve solutions with good classification ability, but with faster training times? How will these, and the two existing techniques to approximate the AUC in fitness (Auc_E and Wmw), compare to the full AUC (Auc_F) on these tasks? We try to answer these questions in the next section.

VI. New Fitness Functions for Classification With Unbalanced Data

We present four *new* fitness functions for classification with unbalanced data to address the limitations described in the previous section. The first two $Amse$ and $Incr$ aim to improve the traditional measure Ave (2). The second two $Corr$ and $Dist$ use novel threshold-independent measures aimed at evolving solutions with good class separability but with faster training times than the AUC-based functions.

Fitness Function $Amse$: Equation (5) is based on the mse, which is a popular machine learning measure for determining the difference between input and output patterns [20], [57], [61]. However, (5) uses the *average* mse for each class, whereas many other approaches (such as those in [20] and [57]) use the overall mse for all training examples. This fitness function is similar to the fitness Ave (2) except that the *magnitude* or values of the genetic program outputs are also factored into the fitness, whereas Ave only considers the TP and TN rates (magnitude of genetic program outputs ignored). The goal of this fitness function is to evolve classifiers whose outputs are closely "calibrated" with the desired or target values for each class, where solutions with smaller deviations between the target and classifier outputs are rewarded with better fitness over solutions with larger differences.

In (5), P_{ci} represents the output of a genetic program classifier when evaluated on the ith example belonging to class c, N_c is the number of examples in class c, and K is the number of classes. The target T_c values for the majority and minority classes are -0.5 and 0.5, respectively, according to the zero-class-threshold approach (minority predictions should be positive, and majority predictions should be negative)

$$Amse = \frac{1}{K}\sum_{c=1}^{K}\left(1 - \frac{\sum_{i=1}^{N_c}\left(\mathrm{sig}(P_{ci}) - T_c\right)^2}{N_c \times 2}\right) \tag{5}$$

where

$$\mathrm{sig}(x) = \frac{2}{1+e^{-x}} - 1.$$

However, as the output of a genetic program classifier P_{ci} has no bounds (can be anything between $-\infty$ or $+\infty$), P_{ci} must be bounded (or scaled) for consistency in fitness values across the population. If this is not enforced, genetic programs which produce large output values risk inflating the difference between target and actual values (i.e., poorer fitness) compared to other genetic programs with similar accuracy but which produce smaller output values. For example, if two classifiers S_1 and S_2 have the same class accuracy but S_1 outputs values in the range $[-100, 100]$ and S_2 in the range $[-5, 5]$, then the difference between target outputs (T_c) and genetic program outputs (P_{ci}) will be larger for S_1 by virtue of the larger genetic program outputs only.

For this reason, (5) uses a sigmoid function (sig) to scale the genetic program outputs (P_{ci}) to the range $[-1, 1]$. This sigmoid function is applied to the value returned from the root node of the genetic program during the fitness evaluation and serves only to scale the range of genetic program outputs to -1 and $+1$ (sign of genetic program output values unaltered). The scaling ensures that positive output values are "spread out" between 0 and 1, and not simply "cut off" at 1, and likewise for negative output values between 0 and -1.

The difference between target and actual classifier outputs for each class is normalized to values between 0 and 1, where $N_c \times 2$ in the denominator in (5) is the absolute difference between the smallest (-1) and largest $(+1)$ genetic program output values (according to the sigmoid function). This is then inverted (1—error) in order to make the fitness values returned from this function consistent with the other fitness functions (0-worst and 1-best).

Fitness Function $Incr$: Equation (6) extends the function Ave by assigning incremental *rewards* to solutions whose class predictions fall further away from the class boundary. $Incr$ improves the traditional Ave by differentiating between solutions which have the same class accuracy, but which use different internal classification models. By counting the average number of incremental rewards earned per class, $Incr$ favors solutions with better classification models (i.e., classifier predictions that fall further away from the class boundary).

In (6), P_{ci} represents the output of a genetic program classifier when evaluated on the ith example belonging to class c, N_c is the number of examples in class c, and K is the number of classes. The term D_{cj} represents the jth element of the set of *distinct* genetic program outputs for all examples in class c (i.e., distinct P_{ci} values for class c), and M_c is the number of distinct genetic program outputs for all examples in class c. The denominator in (6) corresponds to the maximum reward that a solution can obtain for each class; this serves to normalize the rewards earned in each class to values between 0 and 1. As a result, fitness values for $Incr$ range between 0 (worst fitness) and 1 (best fitness)

$$Incr = \frac{1}{K}\sum_{c=1}^{K}\left(\frac{\sum_{j=1}^{M_c}\left[I_{zt}(j, D_{cj}, c).\sum_{i=1}^{N_c} Eq(D_{cj}, P_{ci})\right]}{\frac{1}{2}N_c(N_c+1)}\right) \tag{6}$$

where

$$I_{zt}(r,k,c) = \begin{cases} r, & \text{if } k \geq 0 \text{ and } c \in Min \\ & \text{or if } k < 0 \text{ and } c \in Maj \\ 0, & \text{otherwise} \end{cases}$$

$$Eq(p,q) = \begin{cases} 1, & \text{if } p = q \\ 0, & \text{otherwise.} \end{cases}$$

Equation (6) uses two main components to calculate the incremental rewards for each class; these correspond to the two indicator functions I_{zt} and Eq. The first component I_{zt} returns its first argument if the given prediction is correct with respect

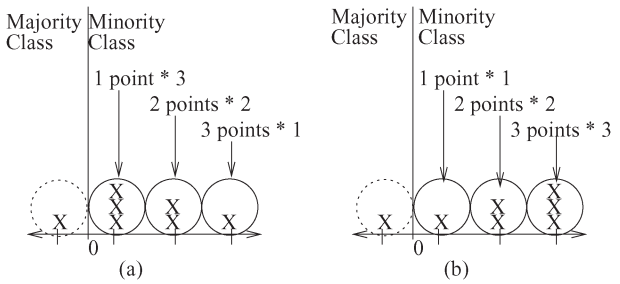


Fig. 4. Genetic program outputs for two classifiers; X denotes the classifier's outputs for different (minority class) examples, where equivalent X values are stacked above each other. The solid circle shows correct class predictions, and the dotted circle shows incorrect predictions. Classifier (b) earns more rewards than (a), as (b) has more predictions that lie further away from the (zero) class threshold; these earn the highest reward per prediction.

to the zero class threshold or 0 if otherwise. In this case, j is returned for correct predictions, where j is the incremental reward earned for the given prediction. As the genetic program outputs are processed in ascending order, the reward earned will increase as predictions lie further and further away from the class threshold. The second component Eq returns 1 if two genetic program outputs are the same or 0 if otherwise; this counts the number of different genetic program outputs that evaluate the given value.

Fig. 4 shows an example of how the incremental reward for a particular class is calculated using two different genetic program classifiers. In Fig. 4, X denotes the genetic program outputs (at some value along the horizontal axis) when evaluated on seven minority class instances. Notice that these seven genetic program outputs only correspond to four distinct values (equivalent X values are stacked above each other in Fig. 4). Each distinct genetic program output is circled (solid and dotted circles); these circles are referred to as *clusters* in Fig. 4. According to the zero class threshold, both solutions (a) and (b) in Fig. 4 have three clusters that correspond to "correct" predictions (solid circles) and one incorrect prediction (dotted circle). As there are exactly three "correct" clusters of genetic program outputs, the incremental rewards are as follows: one point for each prediction in the *first* cluster (nearest to the class threshold), two points for each prediction in the second cluster, and three points for each prediction in the third cluster.

Using these incremental rewards, solution (a) accumulates a total of ten points: three points in the first cluster (one point for each prediction), four points in the second cluster (two points for each prediction), and three points in the third cluster (three points for the single prediction). Similarly, solution (b) accumulates a total of 14: one point in the first cluster, four points in the second cluster, and nine points in the third cluster. Solution (b) is therefore rewarded with a higher fitness than solution a (for this particular class), as b has more predictions that are further away from the class boundary than a. Using the fitness function Ave, these two solutions will have equal fitness on this class as both correctly labeled a total of six examples.

Fitness Function $Corr$: Equation (7) is a novel fitness function based on the well-known statistical measure, the correlation ratio, which measures linear dispersal between two populations of data [62]. The correlation ratio can be used for classification if we consider the genetic program outputs, when evaluated on the examples from the two classes, as the two populations of data and how these are separated with respect to each other. The higher the dispersal between these two populations, the better the separability of the genetic program outputs for the two classes. The correlation ratio outputs values between 0 (poor separability) and 1 (good separability)

$$Corr = \frac{1}{K}\left(r + I_{zt}(1, \mu_{\min}, \mu_{\mathrm{maj}})\right) \tag{7}$$

where

$$r = \sqrt{\frac{\sum_{c=1}^{K} N_c(\mu_c - \bar{\mu})^2}{\sum_{c=1}^{K}\sum_{i=1}^{N_c}(P_{ci} - \bar{\mu})^2}}$$

$$\mu_c = \frac{\sum_{i=1}^{N_c} P_{ci}}{N_c} \quad \bar{\mu} = \frac{\sum_{c=1}^{K} N_c \mu_c}{\sum_{c=1}^{K} N_c}.$$

In (7), r computes the correlation ratio, where P_{ci} is the output of the classifier when evaluated on the ith example belonging to class c, N_c is the number of examples in class c, and K is the number of classes; μ_c represents the mean of classifier outputs for class c only, and $\bar{\mu}$ represents the mean of μ_c for both minority and majority classes. As r only measures the separability of output values, indicator function I_{zt} [from (6)] enforces the zero class threshold. In this case, I_{zt} returns 1 if majority and minority class predictions are negative and nonnegative, respectively, and 0 if otherwise.

Fitness Function $Dist$: Equation (8) treats the genetic program outputs from the examples from the two classes as two independent distributions and measures the distance between these class distributions as the level of class separability. $Dist$ was originally developed for multiple-class problems with relatively balanced class distributions [54] and has not previously been evaluated on binary class imbalance tasks. Equation (8) computes the point equidistant from the means of two distributions, measured in terms of standard deviations away from the mean (where the standard deviations can be different for the two distributions). In the worst case, where the means and standard deviations of both class distributions are the same (poor separability), this distance will be zero. In the ideal case, where there is no overlap between the two class distributions (high separability), this distance will be large (go to $+\infty$).

In (8), μ_c and σ_c correspond to the mean and standard deviation of the class distribution c, respectively, where c is either the minority (min) or majority (maj) class. Similarly, P_{ci} is the output of the classifier when evaluated on the ith example belonging to class c, and N_c is the number of examples in class c. $Dist$ also uses the indicator function I_{zt} to enforce the zero class threshold; here, the distance value for a solution is doubled if I_{zt} returns 1

$$Dist = \frac{|\mu_{\min} - \mu_{\mathrm{maj}}|}{\sigma_{\min} + \sigma_{\mathrm{maj}}} \times I_{zt}(2, \mu_{\min}, \mu_{\mathrm{maj}}) \tag{8}$$

where

$$\mu_c = \frac{\sum_{i=1}^{N_c} P_{ci}}{N_c} \quad \sigma_c = \sqrt{\frac{1}{N_c}\sum_{i=1}^{N_c}(P_{ci} - \mu_c)^2}.$$

TABLE II
AVERAGE (± STANDARD DEVIATION) OF THE AUC, BEST AUC, AND TRAINING TIMES OVER 50 RUNS FOR GP FITNESS FUNCTIONS ON THE SIX TASKS. THE GROUP RANK OF FITNESS FUNCTIONS WITH STATISTICALLY SIMILAR AUC IS SHOWN IN "g.r" (1 IS THE BEST), WHILE "s.b.t" SHOWS WHICH GROUPS ARE STATISTICALLY WORSE. THE CLASS IMBALANCE RATIO IS SHOWN IN PARENTHESES

Task	Fitness Funct.	AUC Average	AUC Best	Signif. Test g.r	Signif. Test s.b.t	Train Time
Ion (1:3)	Corr	0.87 ± 0.04	0.94	1	{3-4}	2.4s ± 0.5
	Dist	0.86 ± 0.05	0.95	1	{3-4}	1.4s ± 0.5
	Amse	0.85 ± 0.05	0.94	2	{4}	2.8s ± 0.9
	AucF	0.85 ± 0.04	0.94	2	{4}	20.0s ± 5.3
	AucE	0.85 ± 0.05	0.93	2	{4}	3.1s ± 0.9
	Wmw	0.85 ± 0.06	0.96	2	{4}	17.8s ± 4.6
	Acc	0.82 ± 0.06	0.93	3	{}	2.7s ± 0.8
	Ave	0.80 ± 0.06	0.92	4	{}	2.8s ± 0.9
	Incr	0.79 ± 0.07	0.90	4	{}	3.5s ± 0.7
		p=7.2×10^{-20}				
Ped (1:4)	Wmw	0.93 ± 0.01	0.94	1	{3-6}	49.2m ± 7.1
	AucF	0.92 ± 0.01	0.94	1	{3-6}	71.3m ± 9.9
	AucE	0.92 ± 0.01	0.94	2	{4-6}	5.8m ± 1.9
	Dist	0.90 ± 0.02	0.92	3	{5-6}	2.4m ± 1.1
	Corr	0.89 ± 0.01	0.92	3	{5-6}	4.5m ± 2.9
	Amse	0.88 ± 0.02	0.91	4	{6}	4.5m ± 0.9
	Ave	0.87 ± 0.04	0.92	4	{6}	5.0m ± 3.0
	Incr	0.86 ± 0.04	0.92	5	{6}	6.1m ± 2.1
	Acc	0.79 ± 0.12	0.92	6	{}	5.4m ± 1.8
		p=1.7×10^{-45}				
Yst$_2$ (1:9)	Amse	0.96 ± 0.01	0.98	1	{3-4}	11.4s ± 2.9
	Corr	0.95 ± 0.02	0.98	1	{3-4}	10.3s ± 3.1
	Wmw	0.95 ± 0.02	0.98	2	{4}	1.4m ± 0.3
	AucF	0.95 ± 0.03	0.98	2	{4}	1.6m ± 0.3
	Dist	0.94 ± 0.03	0.97	2	{4}	5.8s ± 2.3
	Ave	0.93 ± 0.04	0.97	3	{}	12.6s ± 7.9
	AucE	0.92 ± 0.03	0.98	4	{}	15.2s ± 5.3
	Incr	0.92 ± 0.04	0.97	4	{}	15.5s ± 5.0
	Acc	0.93 ± 0.04	0.97	4	{}	12.6s ± 7.9
		p=5.3×10^{-17}				
Spt (1:4)	AucF	0.77 ± 0.04	0.83	1	{4-5}	12.8s ± 3.8
	Incr	0.76 ± 0.05	0.86	1	{4-5}	4.4s ± 1.8
	AucE	0.76 ± 0.04	0.86	1	{4-5}	2.8s ± 0.9
	Amse	0.75 ± 0.04	0.84	2	{5}	2.2s ± 0.4
	Wmw	0.74 ± 0.05	0.86	3	{}	15.3s ± 3.6
	Corr	0.74 ± 0.05	0.84	3	{}	2.3s ± 0.7
	Dist	0.73 ± 0.05	0.82	4	{}	1.2s ± 0.4
	Acc	0.72 ± 0.06	0.85	5	{}	2.3s ± 0.6
	Ave	0.71 ± 0.05	0.82	5	{}	2.6s ± 1.0
		p=2.0×10^{-13}				
Yst$_1$ (1:6)	Wmw	0.84 ± 0.02	0.88	1	{3-5}	1.8m ± 0.4
	AucF	0.83 ± 0.02	0.87	2	{4-5}	2.1m ± 0.6
	Dist	0.83 ± 0.03	0.87	2	{4-5}	6.0s ± 1.6
	Amse	0.82 ± 0.02	0.86	2	{4-5}	13.2s ± 4.7
	AucE	0.82 ± 0.02	0.87	2	{4-5}	13.3s ± 3.3
	Corr	0.81 ± 0.02	0.86	3	{5}	12.8s ± 3.0
	Incr	0.79 ± 0.05	0.87	4	{5}	15.7s ± 4.9
	Ave	0.79 ± 0.03	0.85	4	{5}	13.3s ± 4.7
	Acc	0.76 ± 0.07	0.84	5	{}	13.5s ± 5.7
		p=1.2×10^{-32}				
Bal (1:12)	Wmw	0.86 ± 0.08	0.98	1	{3-7}	26.9s ± 8.0
	AucF	0.84 ± 0.09	0.98	2	{5-7}	28.1s ± 7.6
	AucE	0.84 ± 0.11	0.98	2	{5-7}	5.0s ± 1.3
	Incr	0.83 ± 0.11	0.98	2	{5-7}	5.8s ± 2.2
	Amse	0.78 ± 0.10	0.97	3	{6-7}	5.2s ± 1.4
	Dist	0.77 ± 0.13	0.96	4	{7}	2.7s ± 1.3
	Corr	0.75 ± 0.11	0.98	5	{7}	4.9s ± 1.9
	Ave	0.71 ± 0.15	0.98	6	{7}	4.7s ± 1.5
	Acc	0.55 ± 0.09	0.90	7	{}	5.1s ± 2.0
		p=1.5×10^{-43}				

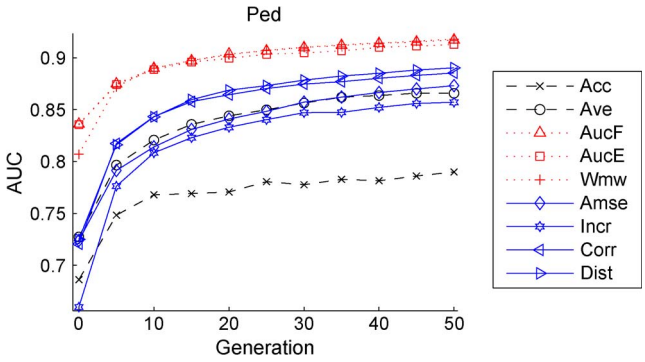


Fig. 5. Average AUC (training set) of the fittest evolved solution in the population using the GP fitness functions over 50 independent runs (Ped task).

VII. GP EXPERIMENT RESULTS

In this section, we discuss the training and test performances of our GP approaches, and a comparison with NB and SVMs, for the six unbalanced data sets.

A. Training Performance of GP Fitness Functions

Fig. 5 shows the effectiveness of the GP fitness functions during the training process for the Ped task only. As GP is a stochastic learning algorithm, each experiment is repeated 50 times using a different random seed in each run. Therefore, this figure reports the average AUC (on the training set) of the fittest solution in the population over the 50 independent runs. This figure shows the smooth increase in fitness over 50 generations for each of the fitness functions, suggesting that training is not an issue for this task. Fig. 5 also shows that the performances on the training set and the test set (discussed in the next section) are very consistent for this task, suggesting that no serious overfitting has occurred in the experiments. These conclusions are very similar for all six tasks; for this reason, the corresponding figures for the remaining tasks are omitted for space constraints. There is also a pattern in the AUC performances for the different methods in this figure. The three AUC-based functions (Auc_F, Auc_E, and Wmw) show the best performance, as expected, followed by the four new fitness functions $Dist$, $Amse$, $Corr$, and $Incr$ and the traditional measure Ave. The worst performance is from the standard GP fitness function Acc.

B. AUC of GP Fitness Functions on Test Set

Table II shows the experimental results on the test set for the different GP fitness functions, for the six unbalanced data sets. This table reports the average and standard deviation of the AUC (on the *test* set) and the training times, the highest AUC from an evolved solution, and the outcome from the statistical significance tests, using the different GP fitness functions over 50 runs. Training times are reported in seconds or minutes. The AUC is used as the primary measure of classification performance as this metric is not dependent on a single class threshold (as previously discussed) and is insensitive

to skewed class distributions unlike the overall classification accuracy.

An ANOVA F-test [63] of the AUC from the different GP fitness functions is used to statistically test the null hypothesis, i.e., no difference between the fitness functions over 50 runs (5% level of significance). The p-values from the F-test (shown as p in Table II) are extremely low for each task. This indicates that there is a statistically significant difference in AUC using the different GP fitness functions over the 50 independent runs for each task, i.e., null hypothesis rejected, as the smaller the p-values, the greater the statistical difference.

Therefore, a *post hoc* multiple-comparison test using Tukey's honestly significant difference (HSD) [64] is used to determine the statistically significant differences between group means. Tukey's HSD test conducts a series of pairwise comparisons[3] using the mean AUC from the different GP fitness functions and outputs a set of 95% confidence intervals for each comparison based on the *studentized range* distribution (similar to a student t-test) [64]. Note that a Shapiro–Wilk test [63] verified that our experiment data are normally distributed (required for Tukey's HSD test[4]).

We summarized the outcome of Tukey's multiple-comparison test on the different GP fitness functions and highlighted those fitness functions which produce AUC results that are *significantly better* than other fitness functions in the "Signif. Test" column in Table II for each task. Here, "g.r" denotes the *group rank* of fitness functions with statistically similar AUC for a particular task (1 is best), and "s.b.t" is the *set* of other groups that are statistically worse.

The empty set denotes that a particular fitness function was *not* statistically better than any other group.

For example, Table II shows that, for the Ion task, fitness function $Corr$ has a group rank of 1 and is *statistically better than* groups with ranks 3 and 4. This means that $Corr$ shows the best average AUC (0.87) for this task (along with $Dist$), and this is statistically better than those fitness functions in groups 3 (Acc) and 4 (Ave and $Incr$). As $Corr$ and $Dist$ both have group ranks of 1, they are statistically no different from one another in this task. However, both are ranked better than group 2 ($Amse$, Auc_F, Auc_E, and Wmw) as group 2 is only significantly better than group 4.

The different GP fitness functions are also ranked in terms of which scored the most number of top-three AUC performances with respect to all other fitness functions, on a run-by-run basis for each task. These results are shown in Table III. The top-three AUC performances for a single run (for a particular task) are those fitness functions which scored the highest, the second highest, and the third highest AUC in that particular run only. The first-, second-, and third-place rankings achieved for each fitness function are summed over all the 50 independent runs, for all six tasks. For example, the first line in Table III shows that, out of a total of 300 GP experiments (i.e., 50 runs $\times$ 6 tasks), Auc_F scored the most number of top-three ranks (178). This fitness function evolved solutions with the highest AUC with respect to all other fitness functions 58 times, the second highest AUC 65 times, and the third highest AUC 54 times.

TABLE III
FIRST-, SECOND-, AND THIRD-PLACE AUC RANKINGS (ON A RUN-BY-RUN BASIS) FOR THE GP FITNESS FUNCTIONS ACROSS ALL TASKS AND THE SUM OF TOP-THREE RANKS. THE HIGHEST POSSIBLE RANK IS 300 (RANKED IN THE SAME POSITION FOR EACH OF THE 50 RUNS, FOR ALL SIX TASKS)

Fitness Function	Num. 1st	Num. 2nd	Num. 3rd	Sum of Top Three Ranks
AucF	59	65	54	178
Wmw	66	69	41	176
AucE	44	42	49	135
Dist	35	34	37	106
Amse	30	25	37	92
Corr	28	24	31	83
Incr	29	15	21	65
Ave	11	15	17	43
Acc	4	9	15	28

Table III shows a similar pattern of AUC behavior on the test set (compared with the training performance discussed in the previous section) for the different GP fitness functions. As expected, the three AUC-based functions (Auc_F, Wmw, and Auc_E) usually evolved solutions with the highest AUC on the tasks, showing the largest number of top-three rankings from all other fitness functions (178, 176, and 135, respectively). Following closely are the four new fitness functions $Dist$, $Amse$, $Corr$, and $Incr$. This suggests that these are useful new measures for evolving solutions with good AUC on the tasks compared to the two traditional measures Acc and Ave which are at the bottom of Table III.

1) AUC-Based Fitness Functions: Not surprisingly, Auc_F and Wmw are the two highest ranking fitness functions, producing the most number of top-three ranks according to Table III. Table II shows that these two fitness functions also incur the longest training times on the tasks, as expected. However, notice that the training times using Auc_E are substantially faster than both Auc_F and Wmw in all tasks. This difference is significant in the largest task, Ped (more than 10 000 training examples): Auc_E taking approximately 5 min on average compared to 71 and 49 min for Auc_F and Wmw, respectively. This suggests that, while Wmw gives a very close approximation to the full AUC in the fitness function, no substantial gain can be made in terms of reducing training time, whereas Auc_E offers a significant reduction in training time while still evolving solutions with high AUC.

2) New Fitness Functions: Table II shows that the AUC using $Dist$, $Amse$, and $Corr$ is as good as Auc_E in nearly all of the six tasks. Each is significantly better than Auc_E in exactly one task (Yst_2) and statistically no different to Auc_E in exactly four tasks. Table II also shows that the training times using these three fitness functions are faster than Auc_E in all tasks. This is most apparent using $Dist$, where the average training time is approximately twice as fast as Auc_E in all tasks. This suggests that these new fitness functions are fast and effective measures of classifier separability, comparable to the fastest of the AUC-based measures Auc_E. Of particular interest is $Dist$, which scored the largest number of top-three ranks from all of the new fitness functions (104 according to Table III)

[3]There are $(k(k-1)/2)$ total comparisons, where k is the number of different GP fitness functions.

[4]The Kruskal–Wallis test [63] is a nonparametric multiple-comparison test which can be used for the same purposes when this assumption does not hold.

TABLE IV
AVERAGE (± STANDARD DEVIATION) AUC OVER 50 RUNS FOR GP FITNESS FUNCTION Ave (2) USING DIFFERENT CONFIGURATIONS ON THE SIX TASKS. THE GROUP RANK OF CONFIGURATIONS WITH STATISTICALLY SIMILAR AUC IS SHOWN IN "g.r" (1 IS THE BEST), WHILE "s.b.t" SHOWS WHICH GROUPS ARE STATISTICALLY WORSE. CLASS IMBALANCE RATIO IS SHOWN IN PARENTHESES

Weight W	Task	Ave AUC	Signif. Test g.r	s.b.t	Task	Ave AUC	Signif. Test g.r	s.b.t	Task	Ave AUC	Signif. Test g.r	s.b.t
0.2		0.83 ± 0.05	1	{2-3}		0.70 ± 0.09	3	{}		0.80 ± 0.09	3	{}
0.3		0.82 ± 0.05	1	{2-3}		0.73 ± 0.05	1	{4-5}		0.86 ± 0.06	1	{3}
0.4		0.82 ± 0.05	1	{2-3}		0.72 ± 0.05	2	{5}		0.86 ± 0.05	1	{3}
0.5	Ion	0.80 ± 0.06	1	{2-3}	Spt	0.71 ± 0.05	3	{}	Ped	0.87 ± 0.04	1	{3}
0.6	(1:3)	0.80 ± 0.05	1	{2-3}	(1:4)	0.69 ± 0.06	4	{}	(1:4)	0.86 ± 0.03	1	{3}
0.7		0.76 ± 0.07	2	{3}		0.69 ± 0.06	4	{}		0.85 ± 0.05	2	{}
0.8		0.71 ± 0.09	3	{}		0.68 ± 0.06	5	{}		0.82 ± 0.05	3	{}
			p-val	1.7×10^{-26}			p-val	0.0013			p-val	1.1×10^{-9}
0.2		0.76 ± 0.07	3	{}		0.92 ± 0.05	1	{}		0.61 ± 0.13	3	{}
0.3		0.78 ± 0.05	2	{}		0.92 ± 0.05	1	{}		0.72 ± 0.14	1	{3}
0.4		0.80 ± 0.04	1	{3}		0.93 ± 0.04	1	{}		0.72 ± 0.13	1	{3}
0.5	Yst_1	0.79 ± 0.03	2	{}	Yst_2	0.93 ± 0.04	1	{}	Bal	0.71 ± 0.15	1	{3}
0.6	(1:6)	0.78 ± 0.05	2	{}	(1:9)	0.92 ± 0.04	1	{}	(1:12)	0.69 ± 0.14	2	{}
0.7		0.77 ± 0.06	3	{}		0.93 ± 0.04	1	{}		0.67 ± 0.14	2	{}
0.8		0.73 ± 0.10	3	{}		0.92 ± 0.05	1	{}		0.74 ± 0.14	1	{3}
			p-val	3.2×10^{-6}			p-val	0.49			p-val	5.5×10^{-5}

while consistently showing the quickest training times from all fitness functions on the tasks.

Interestingly, Table II shows that the AUC using $Amse$ (5) is significantly better than the traditional measure Ave (2) in all tasks except Ped (where AUC is similar). This is interesting as both $Amse$ and Ave are relatively similar classification measures; the only difference is that $Amse$ utilizes the magnitude of the genetic program output in fitness to "calibrate" a classifier's output to target values for each class, whereas Ave uses only the TP and TN rates in fitness (magnitude ignored).

Notice also that the AUCs of $Dist$, $Amse$, and $Corr$ are statistically no different to one another in all tasks. However, $Amse$ outperforms the traditional Ave more often than $Dist$ and $Corr$: $Amse$ is significantly better than Ave in all tasks (except Ped), whereas $Dist$ and $Corr$ are significantly better than Ave in only two tasks each. This suggests that good threshold-based measures in the fitness function such as $Amse$ can be more effective in evolving high-AUC solutions compared to measures such as $Dist$ and $Corr$, but further investigation is needed.

Although Table III ranks $Incr$ below $Dist$, $Amse$, and $Corr$, this fitness function achieves very good performance on the two tasks with the smallest number of minority class examples, Spt and Bal; these tasks only have 24 and 27 training examples, respectively. Table II shows that $Incr$ has a group rank of 1 in Spt and 2 in Bal—this is as good as both Auc_F and Auc_E and significantly better than Ave—suggesting that $Incr$ is a useful new measure, particularly on tasks with very few minority class examples.

3) Traditional Fitness Functions Acc and Ave: As previously discussed, Table II shows that the AUC using the standard Acc (1) is nearly always significantly worse than the AUC-based functions and new fitness functions, in all tasks. However, notice that, on average, the AUCs using Ave and Acc are statistically no different to one another in exactly half the tasks (Ion, Spt, and Yst_2). This shows that using the average classification accuracy of the minority and majority classes in fitness does not always evolve solutions with good discrimination ability between the two classes compared to the standard Acc in class imbalance scenarios. However, our new measures successfully improved performance compared to both these fitness functions on the tasks.

C. Analysis of Weighted-Average Fitness Function

This section explores whether different configurations in the GP fitness function Ave (2), which favor either majority or minority class accuracy, can improve the AUC of evolved solutions compared to an equal class weighting. Recall that, in Ave, W controls the contribution of minority class accuracy to majority class accuracy in the final fitness value, where $0 < W < 1$; an equal class weighting is specified by $W = 0.5$.

Table IV shows the experimental results using the seven different weighting configurations for the six tasks. Similar to the previous GP experimental results (Table II), an ANOVA F-test is used to statistically test the null hypothesis (i.e., no difference using the different W values), and Tukey's multiple-comparison test is used to find the statistically different group means (with respect to average AUC). Table IV shows that the W values are statistically *different* to one another (i.e., null hypothesis rejected) for all tasks except Yst_2 (p-value of 0.49).

According to Table IV, no nonequal weighting configuration in Ave $(W \neq 0.5)$ significantly improved AUC compared to an equal weighting $(W = 0.5)$ on the tasks. In other words, no configuration of W, where $W \neq 0.5$, shows a *significantly better* AUC compared to equal weighting. Interestingly, configurations favoring majority accuracy over minority accuracy $(0.3 < W \leq 0.5)$ produce better AUC compared to the opposite case, where minority accuracy is favored. However, this small improvement is statistically significant in only two tasks (Ion and Spt).

As a guideline, an equal class weighting or configuration slightly favoring majority accuracy gives the best results. The more "extreme" the weighting configuration, the poorer the AUC (with the exception of $W = 0.8$ for Bal, which shows high AUC). This is not surprising since extreme weighting configurations favor the evolution of classifiers strongly biased toward one class only.

TABLE V
AUC AND TRAINING TIME FOR A SINGLE RUN USING NB AND SVM ON THE TASKS

	Ion		Spt		Ped		Yst_1		Yst_2		Bal	
	Auc	Time	Auc	Time	Auc	Time	Auc	Time	Auc	Time	Auc	Time
NB	0.91	0.02s	0.83	0.04s	0.92	20.1s	0.83	0.03s	0.95	0.02s	0.5	0.001s
SVM	0.93	0.08s	0.68	0.2s	0.93	3.8m	0.71	1.2s	0.85	1.4s	0.5	0.05s

D. *NB and* SVMs

We compare the classification results of our GP approaches with two other popular machine learning approaches, namely, NB and SVM. Table V shows the AUC and training time using NB and SVM on the tasks. The SVM uses a sequential minimal optimization algorithm with a radial-basis-function kernel and Gamma value[5] of ten.

Table II shows that the *best* classifiers evolved by GP (over 50 experiments) using the different fitness functions are as good as, or in most case better than, NB and SVM for these tasks. This suggests that these new GP fitness functions, which are designed to measure the level of overlap between two class distributions without bias toward either class, can evolve highly accurate classifiers on the tasks.

Comparing the average AUC using the most effective GP configuration Auc_F (this fitness function is the best ranked in Table III and consistently achieves high average AUC results on the tasks in Table II), GP significantly outperforms both NB and SVM on the task with the lowest proportion of minority class examples, Bal (8% of all examples). On this task, both NB and SVM show equally poor AUC performances, indicating that these methods are biased toward the majority class. GP and NB show similar performances on the tasks with minority class representation between 10%–20% of all examples (Spt, Ped, Yst_1, and Yst_2), where both methods are usually better than SVM. These results suggest that the ability to choose/develop an effective fitness function to evolve accurate classifiers gives GP an advantage over NB and SVM, particularly when data sets are highly unbalanced.

However, GP shows a slightly lower average AUC than a single run of SVM and NB on the Ion task. This is more related to the complexity of this problem rather than to the relatively low level of class imbalance. Ion has 34 features, the largest of the tasks. This represents a very large search space of classifiers for GP, given that we currently restrict the maximum program depth of GP classifiers to eight (for consistency in the experiments). Increasing this maximum GP program depth parameter (e.g., to 10 or 12) would improve the average GP performances on this task by allowing GP to more effectively explore this search space. Similarly, increasing the maximum number of generations to 75 or 100 (this is currently set as 50 also for consistency) can also allow GP more time to spend on searching. As this is not the primary goal of this paper, we will further consider this aspect as future work.

Table V also shows that a single run of SVM, and particularly NB, is faster than the average GP training times on these tasks. However, this is not really a serious concern as GP only takes a few seconds on most of these tasks. The exception is Ped, which is the largest data set (more than 24 000 examples); here, most of the GP methods and SVM take a few minutes.

[5] $Gamma = 10$ generally gave the best classification results from experiments using 0.1, 1, 10, and 100.

VIII. DISCUSSIONS

This section discusses several important related aspects.

A. *Validation Tests*

We have tested these fitness functions on six benchmark problems with different data and variations. They contain certain levels of noise. The validation sets used in this paper correspond to the unseen test sets, i.e., 50% of the original data set, for each task. The performances reported are on the unseen test tests (validation sets) in addition to the training set. A separate validation set or the cross-validation method is not used as it is not necessary: The performances on the training and test sets are very consistent, and no serious overfitting occurred in the experiments.

B. *Fitness Functions for Multiple-Class Problems*

These fitness functions can be used directly for multiple-class problems, provided that a suitable multiple-class classification strategy is also used in GP to effectively translate the single program output value into a set of class labels [5], [6]. Alternatively, a two-class-problem decomposition-based approach can also be used. For example, [60] uses a *one-versus-rest* split to calculate the AUC for each class, where the final AUC is averaged across all classes. A direct comparison between these approaches is beyond the scope of this work.

C. *Adaptability of Cost Functions for Other Paradigms*

These fitness functions can also be used to train other machine learning approaches which may be more suitable for some problems. For example, [37] uses an evaluation criterion similar to Ave, i.e., the geometric mean of the accuracy of the two classes, to train a multilayer-perceptron network for a particular class imbalance task. However, the GP training times are not a serious concern as the actual time for each GP run is only several seconds in most tasks. As the focus of this paper is to develop new techniques within a GP framework, whether these cost functions can be more efficient in training other machine learning tools/methods such neural networks is beyond the scope of this work.

D. *Problem Sets*

The benchmark data sets used in these experiments are carefully selected to encompass a varied collection of problem domains to ensure that our evaluation of the different fitness

functions is not problem specific. These problems have varying levels of class imbalance (minority class representation ranges between 7% and 35% of total examples) and complexity, where some tasks are easily separable (e.g., Yst_2) compared to others. The training/test instances in the different sets also range from being well represented such as Ped (each set has approximately 12 000 instances) to sparsely represented such as Spt (each set has 134 instances, of which only 27 belong to the minority class). We also ensure that these tasks range between high-dimensionality (Ion has the most features at 34) and low-dimensionality (Bal has the least features at 4) problems, and we use binary and real-valued feature types. However, we will evaluate these GP methods on more class imbalance problems as future work.

E. Performance Measures

The AUC is a common approach to measure classification ability across multiple TP and FP rates in class imbalance scenarios. Our conclusions about the effectiveness of these fitness functions (such as the rankings discussed in Table III) are relative to the AUC. However, a variety of other performance measures can also be used to evaluate and compare the different GP fitness functions (and SVM and NB); [20] and [48] define and categorize several metrics for classification (including the AUC). The use of another good inclusive measure for evaluating classifier performance can depend on the problem domain and/or the end goal of the research; this is an open issue in machine learning.

F. Initial Results Using Bagging Techniques

As bagging with balanced bootstrap samples, i.e., balanced subsets of class examples, is a common approach in class imbalance [13], [33], [42], we compared our results to a bagging approach on these tasks. The training set is sampled with replacement to generate N balanced bootstrap samples, where the outputs of the N base classifiers are combined using voting. Initial results using $5 \leq N \leq 50$ show that bagging with SVM (as the base classifiers) can improve AUC on the test sets but that this performance is not better than the GP methods on the tasks with high level of class imbalance (such as Bal, Yst, and Ped). Bagging with NB shows no improvement on the tasks except for Bal, where the best AUC reached (0.61 using $N = 50$) is still considerably poorer than both GP and SVM with bagging. These results are omitted for space constraints (and in fact, a detailed comparison with bagging is also beyond the scope of this paper).

IX. Conclusion

The goals of this paper are to highlight the limitations of both the standard GP fitness function and two current approaches for binary classification with unbalanced data and to develop new GP fitness functions to address these limitations. The proposed GP methods utilize the unbalanced data "as is" in the learning phase, requiring no prior knowledge about the problem domain, to evolve classifiers with good classification ability on both minority and majority classes. These goals were achieved by developing new measures for class separability and by examining the experimental results of the different GP approaches across six real-world class imbalance problems.

The classifiers that evolved using the standard GP fitness (Acc) performed poorly on the tasks compared to the improved fitness functions. Not surprisingly, the AUC-based fitness functions usually evolved solutions with the best AUC on the tasks but also incurred the longest training times. The WMW statistic to approximate the AUC in the fitness function (Wmw) showed similar performances to the full AUC (Auc_F), including very long training times, whereas the reduced-precision AUC estimation (Auc_E) offered a significant reduction in training time while still evolving high-AUC solutions.

A new fitness function measuring the distance between class distributions $(Dist)$ evolved solutions that performed as well as the AUC-based measure Auc_E but with training times twice as fast. Two new fitness functions based on the mse for each class $(Amse)$ and the correlation ratio $(Corr)$ also show similar performance to Auc_E with slightly better training times. Of these, $Amse$ was significantly better than the traditional Ave in nearly all tasks. A new fitness function, which incrementally rewards correct predictions further away from the class boundary $(Incr)$, outperformed the traditional Ave in tasks with very few minority class examples.

Varying the misclassification costs for the minority and majority classes using a weighted average of these accuracies in the fitness function did not significantly improve AUC compared to an equal weighting. However, weighting configurations favoring majority accuracy over minority accuracy showed slightly better results than the opposite case. Our GP methods can also outperform NB or SVMs on tasks with high levels of class imbalance.

For future work, we will develop further improvements to the fitness functions, evaluate our GP methods on class imbalance problems, compare these results to cross-validation techniques in training, and compare our work to other approaches for class imbalance such as bagging algorithms and multiobjective GP.

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

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

[3] J. Eggermont, J. N. Kok, and W. A. Kosters, "Genetic programming for data classification: Partitioning the search space," in *Proc. ACM SAC*, 2004, pp. 1001–1005.

[4] S. Winkler, M. Affenzeller, and S. Wagner, "Advanced genetic programming based machine learning," *J. Math. Model. Algorithms*, vol. 6, no. 3, pp. 455–480, Sep. 2007.

[5] M. Zhang and W. Smart, "Multiclass object classification using genetic programming," in *Proc. Appl. Evol. Comput.*, vol. 3005, *LNCS*, 2004, pp. 369–378.

[6] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Proc. Congr. Evol. Comput.*, 2001, vol. 12, pp. 1070–1077.

[7] N. V. Chawla, N. Japkowicz, and A. Kolcz, "Editorial: Special Issue on learning from imbalanced data sets," *ACM SIGKDD Explor. Newslett.*, vol. 6, no. 1, pp. 1–6, Jun. 2004.

[8] D. Howard, S. Roberts, and R. Brankin, "Target detection of SAR imagery by genetic programming," *Adv. Eng. Softw.*, vol. 30, no. 5, pp. 303–311, May 1999.

[9] K.-K. Sung, "Learning and example selection for object and pattern recognition," Ph.D. dissertation, AI Lab. and Center Biol. Comput. Learn., MIT, Cambridge, MA, 1996.

[10] E. Pednault, B. Rosen, and C. Apte, "Handling imbalanced data sets in insurance risk modeling," in *Proc. Workshops 17th Nat. Conf. Artif. Intell., Learn. From Imbalanced Data Sets*, 2005, pp. 58–63.

[11] D. Song, M. Heywood, and A. Zincir-Heywood, "Training genetic programming on half a million patterns: An example from anomaly detection," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 225–239, Jun. 2005.

[12] T. Fawcett and F. Provost, "Adaptive fraud detection," *Data Mining Knowl. Discov.*, vol. 1, no. 3, pp. 291–316, Sep. 1997.

[13] S. J. Stolfo, D. W. Fan, W. Lee, A. L. Prodromidis, and P. K. Chan, "Credit card fraud detection using meta-learning: Issues and initial results," in *Proc. AAAI Workshop AI Approaches Fraud Detection Risk Manage.*, 1997, pp. 83–90.

[14] M. C. Monard and G. Batista, "Learning with skewed class distributions," in *Proc. Adv. Logic, Artif. Intell. Robot.*, 2002, pp. 173–180.

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

[16] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," in *Proc. 14th Int. Conf. Mach. Learn.*, 1997, pp. 179–186.

[17] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[18] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997.

[19] J. Eggermont, A. Eiben, and J. van Hemert, "Adapting the fitness function in GP for data mining," in *Proc. 2nd Eur. Workshop Genetic Program.*, vol. 1598, *LNCS*, 1999, pp. 193–202.

[20] R. Caruana, "Data mining in metric space: An empirical analysis of supervised learning performance criteria," in *Proc. ROC Anal. AI Workshop (ECAI)*, 2004, pp. 69–78.

[21] J. H. Holmes, "Differential negative reinforcement improves classifier system learning rate in two-class problems with unequal base rates," in *Proc. 3rd Annu. Conf. Genetic Program.,* J. R. Roza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds., 1998, pp. 635–644.

[22] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk, "Reducing misclassification costs," in *Proc. 11th Int. Conf. Mach. Learn.*, 1994, pp. 217–225.

[23] H. Gray and R. Maxwell, "Genetic programming for classification of brain tumours from nuclear magnetic resonance biopsy spectra," in *Proc. 1st Annu. Conf. Genetic Program.*, 1996, pp. 424–430.

[24] S. M. Winkler, M. Affenzeller, W. Jacak, and H. Stekel, "Classification of tumor marker values using heuristic data mining methods," in *Proc. 12th Annu. GECCO*, 2010, pp. 1915–1922.

[25] S. Mutter, B. Pfahringer, and G. Holmes, "The positive effects of negative information: Extending one-class classification models in binary proteomic sequence classification," in *Proc. 22nd Australasian Joint Conf. AI*, vol. 5866, *LNAI*, 2009, pp. 260–269.

[26] E. Alfaro-Cid, K. Sharman, and A. Esparcia-Alcazar, "A genetic programming approach for bankruptcy prediction using a highly unbalanced database," in *Proc. Appl. Evol. Comput.*, vol. 4448, *LNCS*, M. Giacobini, Ed., 2007, pp. 169–178.

[27] T. Suttorp and C. Igel, "Multi objective support vector machines," in *Multi-Objective Machine Learning*, Y. Jin, Ed. New York: Springer-Verlag, 2006, ch. 9, pp. 199–220.

[28] J. Doucette and M. I. Heywood, "GP classification under imbalanced data sets: Active sub-sampling and AUC approximation," in *Proc. 11th EuroGP*, 2008, pp. 266–277.

[29] A. Orriols and E. Bernado-Mansilla, "Class imbalance problem in UCS classifier system: Fitness adaptation," in *Proc. IEEE Congr. Evol. Comput.*, Sep. 2005, vol. 1, pp. 604–611.

[30] G. M. Weiss and F. Provost, "Learning when training data are costly: The effect of class distribution on tree induction," *J. Artif. Intell. Res.*, vol. 19, pp. 315–354, 2003.

[31] N. Japcowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intell. Data Anal.*, vol. 6, no. 5, pp. 429–450, Nov. 2002.

[32] R. C. Prati, G. Batista, and M. C. Monard, "Class imbalances versus class overlapping: An analysis of a learning system behavior," in *Proc. 3rd Mexican Int. Conf. Artif. Intell.—Advances in Artificial Intelligence*, vol. 2972, *LNCS*, 2004, pp. 312–321.

[33] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 2, pp. 539–550, Apr. 2009.

[34] J.-X. Chen, T.-H. Cheng, A. L. F. Chan, and H.-Y. Wang, "An application of classification analysis for skewed class distribution in therapeutic drug monitoring—The case of vancomycin," in *Proc. IDEAS Workshop Med. Inf. Syst.: Digital Hospital*, 2004, pp. 35–39.

[35] K. McCarthy, B. Zabar, and G. Weiss, "Does cost-sensitive learning beat sampling for classifying rare classes," in *Proc. 1st Int. Workshop Utility-Based Data Mining*, 2005, pp. 69–77.

[36] G. Batista, R. C. Prati, and M. C. Monard, "Balancing strategies and class overlapping," in *Proc. 6th Int. Symp. IDA—Advances in Intelligent Data Analysis VI*, vol. 3646, *LNCS,* A. F. Famili, J. N. Kok, J. M. Peña, A. Siebes, and A. J. Feelders, Eds., 2005, pp. 24–35.

[37] R. Alejo, V. Garcia, J. M. Sotoca, R. Mollineda, and J. S. Sanchez, "Improving the classification accuracy of RBF and MLP neural networks trained with imbalanced samples," in *Proc. 7th Int. Conf. IDEAL*, Sep. 2006, pp. 467–471.

[38] J. V. Hulse, T. M. Khoshgoftaar, and A. Napolitano, "Experiment perspectives in learning from imbalanced data," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 435–492.

[39] R. Curry, P. Lichodzijewski, and M. Heywood, "Scaling genetic programming to large datasets using hierarchical dynamic subset selection," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 4, pp. 1065–1073, Aug. 2007.

[40] C. Gathercole and P. Ross, "Dynamic training subset selection for supervised learning in genetic programming," in *Proc. PPSN III*, vol. 866, *LNCS,* Y. Davidor, H.-P. Schwefel, and R. Manner, Eds., 1994, pp. 312–321.

[41] V. Nikulin, G. McLachlan, and S. K. Ng, "Ensemble approach for the classification of imbalanced data," in *Proc. 22nd Australasian Joint Conf. AI*, vol. 5866, *LNAI*, 2009, pp. 260–269.

[42] Y. Tang, Y.-Q. Zhang, N. Chawla, and S. Krasser, "SVM modeling for highly imbalanced classification," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 1, pp. 281–288, Feb. 2009.

[43] S. Lawrence, I. Burns, A. D. Back, A. C. Tsoi, and L. C. Giles, "Neural network classification and prior class probabilities," in *Neural Networks: Tricks of the Trade*. New York: Springer-Verlag, 1998, pp. 299–313.

[44] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve," *Radiology*, vol. 143, no. 1, pp. 29–36, Apr. 1982.

[45] S. Rosset, "Model selection via the AUC," in *Proc. 21st Int. Conf. Mach. Learn.*, Banff, AB, Canada, 2004, pp. 89–97.

[46] J. Huang and C. X. Ling, "Constructing new and better evaluation measures for machine learning," in *Proc. 20th IJCAI*, 2007, pp. 859–864.

[47] L. Yan, R. Dodier, M. C. Mozer, and R. Wolniewicz, "Optimizing classifier performance via the Wilcoxon–Mann-Whitney statistic," in *Proc. 20th ICML*, 2003, pp. 848–855.

[48] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognit. Lett.*, vol. 30, no. 1, pp. 27–38, Jan. 2009.

[49] G. Patterson and M. Zhang, "Fitness functions in genetic programming for classification with unbalanced data," in *Proc. 20th Australian Joint Conf. Artif. Intell.*, vol. 4830, *LNCS*, 2007, pp. 769–775.

[50] A. Asuncion and D. Newman, UCI Machine Learning Repository, Irvine, CA. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[51] U. Bhowan, M. Zhang, and M. Johnston, "Evolving ensembles in multi-objective genetic programming for classification with unbalanced data," in *Proc. Genetic Evol. Comput. Conf.*, 2011, pp. 1331–1339.

[52] A. McIntyre and M. Heywood, "Multi-objective competitive coevolution for efficient GP classifier problem decomposition," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2007, pp. 1930–1937.

[53] S. Munder and D. Gavrila, "An experimental study on pedestrian classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 11, pp. 1863–1868, Nov. 2006.

[54] M. Zhang and W. Smart, "Using Gaussian distribution to construct fitness functions in genetic programming for multiclass object classification," *Pattern Recognit. Lett.*, vol. 27, no. 11, pp. 1266–1274, Aug. 2006.

[55] U. Bhowan, M. Johnston, and M. Zhang, "A comparison of classification strategies in genetic programming with unbalanced data," in *Proc. 23rd Australasian Joint Conf. Artif. Intell.*, vol. 6464, *LNCS*, J. Li, Ed., 2010, pp. 243–252.

[56] U. Bhowan, M. Johnston, and M. Zhang, "Differentiating between individual class performance in genetic programming fitness for classification with unbalanced data," in *Proc. IEEE CEC*, 2009, pp. 2802–2809.

[57] S. M. Winkler, M. Affenzeller, and S. Wagner, "Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis," *Genetic Program. Evolvable Mach.*, vol. 10, no. 2, pp. 111–140, Jun. 2009.

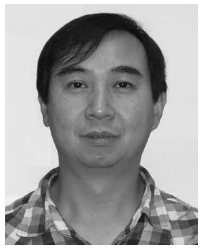
[58] W. B. Langdon and B. Buxton, "Genetic programming for improved receiver operating characteristics," in *Proc. Multiple Classifier Syst.*, vol. 2096, *LNCS*, J. Kittler and F. Roli, Eds., 2001, pp. 68–77.
[59] G. Weiss and F. Provost, "The effect of class distribution on classifier learning: An empirical study," Dept. Comput. Sci., Rutgers Univ., New Brunswick, NJ, Tech. Rep. ML-TR-44, 2001.
[60] F. Provost and P. Domingos, "Tree induction for probability-based rankings," *Mach. Learn.*, vol. 52, no. 3, pp. 199–215, Sep. 2003.
[61] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Mateo, CA: Morgan Kaufmann, Jun. 2005.
[62] R. A. Fisher, *Statistical Methods for Research Workers*, 14th ed. Edinburgh, U.K.: Oliver & Boyd, 1970.
[63] D. C. Hoaglin, F. Mosteller, and J. W. Tukey, *Fundamentals of Exploratory Analysis of Variance*. Hoboken, NJ: Wiley, 1991.
[64] J. W. Tukey, "Components in regression," *Biometrics*, vol. 7, no. 1, pp. 33–69, Mar. 1951.

Urvesh Bhowan (M'09) received the B.Sc. (Hons.) degree in computer science from Victoria University of Wellington, Wellington, New Zealand, in 2004, where he has been working toward the Ph.D. degree in computer science since 2008.

His current research interests include evolutionary computation, particularly genetic programming; multiple-objective optimization; classification with unbalanced data; and computer vision.

Mark Johnston (M'10) received the B.Sc. degree in mathematics and computer science and the Ph.D. degree in operations research from Massey University, Palmerston North, New Zealand.

Since 2005, he has been a Lecturer with Victoria University of Wellington, Wellington, where he teaches various operations research courses. His research is primarily in combinatorial optimization and evolutionary computation, with particular interest in scheduling models, genetic programming, and multiple-objective optimization.



Mengjie Zhang (M'04–SM'10) received the B.E. and M.E. degrees from the Department of Mechanical and Electrical Engineering and Artificial Intelligence Research Center, Agricultural University of Hebei, Baoding, China, in 1989 and 1992, respectively, and the Ph.D. degree in computer science from RMIT University, Melbourne, Australia, in 2000.

Since 2000, he has been an Academic Staff Member with Victoria University of Wellington, Wellington, New Zealand, where he is currently an Associate Professor/Reader in computer science and the Head of the Evolutionary Computation Research Group. He has been serving as an Associate 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 15 international journals. He has also been serving as a Steering Committee Member and a Program Committee Member for over 80 international conferences in the areas of evolutionary computation and artificial intelligence. He has been supervising over 30 postgraduate research project and thesis students. His research is mainly focused on evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of computer vision and image processing, multiobjective optimization, classification with unbalanced data, and feature selection and dimension reduction for classification with high dimensions. He is also interested in data mining, machine learning, and Web information extraction. He has published over 170 academic papers in refereed international journals and conferences in these areas.

Dr. Zhang is a member of the IEEE Computer Society, the IEEE Computational Intelligence Society, and the IEEE Systems, Man, and Cybernetics Society. He is also a member of the IEEE CIS Evolutionary Computation Technical Committee, the 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 (formerly the International Society of Genetic and Evolutionary Computation).