On the Performance of Classification Algorithms for Learning Pareto-Dominance Relations

Sunith Bandaru*, Amos H.C. Ng* and Kalyanmoy Deb†
*Virtual Systems Research Centre, University of Skövde, Skövde, Sweden
Email: {sunith.bandaru,amos.ng}@his.se
†Department of Electrical and Computer Engineering, Michigan State University, East Lansing, USA
Email: kdeb@egr.msu.edu

***Abstract*—Multi-objective evolutionary algorithms (MOEAs) are often criticized for their high-computational costs. This becomes especially relevant in simulation-based optimization where the objectives lack a closed form and are expensive to evaluate. Over the years, meta-modeling or surrogate modeling techniques have been used to build inexpensive approximations of the objective functions which reduce the overall number of function evaluations (simulations). Some recent studies however, have pointed out that accurate models of the objective functions may not be required at all since evolutionary algorithms only rely on the relative ranking of candidate solutions. Extending this notion to MOEAs, algorithms which can 'learn' Pareto-dominance relations can be used to compare candidate solutions under multiple objectives. With this goal in mind, in this paper, we study the performance of ten different off-the-shelf classification algorithms for learning Pareto-dominance relations in the ZDT test suite of benchmark problems. We consider prediction accuracy and training time as performance measures with respect to dimensionality and skewness of the training data. Being a preliminary study, this paper does not include results of integrating the classifiers into the search process of MOEAs.**

***Keywords*—Meta-modeling, Multi-objective optimization, Classification algorithms, Pareto-dominance, Machine learning**

I. Introduction

Meta-modeling or surrogate modeling is now a common methodology employed in optimization problems involving expensive objective(s) and/or constraints. This is usually the case in simulation-based optimization where complex processes are evaluated using time-consuming computational models. Due to high non-linearity of the processes and non-availability of closed-form (analytical) objectives and derivative information, evolutionary algorithms (EAs) are popular in such optimization tasks. However, being population-based, they are also notorious for high computational costs. Thus, meta-modeling plays a more crucial role in improving the resource efficiency of EAs than that of any other optimization algorithm. Consequently, many meta-modeling methods have been developed specifically to be used with evolutionary computation. A survey of these can be found in [1] and [2]. With all methods, the general idea is to approximate the fitness (objective) function, either locally or globally, and use it in place of actual function evaluations (simulations). The meta-model is updated as the population evolves and new computations become available.

Recently however, Runarsson [3] argued that since EAs only rely on the relative ranking of candidate individuals, very accurate approximations of the objective function may not be required. The idea was demonstrated through an *approximate ranking* procedure, which assumes the surrogate model to be sufficiently accurate as long as the selection of solutions between generations is not drastically affected. Runarsson carried forward the idea in [4], which suggested replacing fitness approximating surrogates with solution ranking surrogates. The proposed ranking surrogate performs ordinal regression on the solutions using a rank-based support vector machine (SVM) [5] which maximizes the margin between rank boundaries. Loshchilov et al. [6] implement rank-based SVM within CMA-ES in their aptly titled paper "Comparison-Based Optimizers Need Comparison-Based Surrogates". The proposed approach alleviates the problem of kernel choice using an adaptive kernel derived from the covariance matrix of CMA-ES.

The above studies were however tailored for single-objective optimization problems. Multi-objective optimization problems take the following mathematical form

$$\begin{array}{ll} \text{Minimize} & \mathbf{F}(\mathbf{x}) = \{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})\} \\ \text{Subject to} & \mathbf{x} \in S \end{array} \tag{1}$$

where $f_i : \mathbb{R}^n \to \mathbb{R}$ are $M(> 2)$ conflicting objectives that have to be simultaneously minimized and the variable vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ belongs to the non-empty feasible region $S \subset \mathbb{R}^n$. The feasible region is formed by the constraint functions and the variable bounds. A variable vector $\mathbf{x_1}$ is said to '*weakly* Pareto-dominate' $\mathbf{x_2}$ and denoted as $\mathbf{x_1} \preceq \mathbf{x_2}$ if

$$f_i(\mathbf{x_1}) \leq f_i(\mathbf{x_2}) \; \forall i \in \{1, 2, \ldots, M\}. \tag{2}$$

If in addition to the above there exists at least one $j \in \{1, 2, \ldots, M\}$ such that $f_j(\mathbf{x_1}) < f_j(\mathbf{x_2})$ then the corresponding Pareto-dominance relation is denoted as $\mathbf{x_1} \prec \mathbf{x_2}$, and read as $\mathbf{x_1}$ 'Pareto-dominates' $\mathbf{x_2}$ or $\mathbf{x_1}$ 'is better than' $\mathbf{x_2}$ or $\mathbf{x_1}$ 'is preferable to' $\mathbf{x_2}$. If neither $\mathbf{x_1} \preceq \mathbf{x_2}$ nor $\mathbf{x_2} \preceq \mathbf{x_1}$, then $\mathbf{x_1}$ and $\mathbf{x_2}$ are said to be 'non-dominated with respect to each other' or 'equivalent' or 'incomparable'. This dominance relation is denoted as $\mathbf{x_1}||\mathbf{x_2}$. A vector $\mathbf{x}^* \in S$ is said to be 'Pareto-optimal' if there does not exist any $\mathbf{x} \in S$ such that $\mathbf{x} \prec \mathbf{x}^*$. The set of all such $\mathbf{x}^*$ (which are, by definition, also non-dominated with respect to each other) is referred to as the 'Pareto-optimal set'. The projection of the Pareto-optimal set in the objective space, $\mathbf{F}(\mathbf{x}^*) \; \forall \mathbf{x}^*$ is called the 'Pareto-optimal front'. Most MOEAs work by dividing the population into 'non-dominated levels' or 'ranks' or 'fronts' and promoting high (numerically smaller) rank solutions.

Metamodels for multi-objective optimization problems are usually a straightforward extension of those used in single-

objective optimization, meaning that an independent meta-model is built for each objective $f_i(\mathbf{x})$. Model accuracy becomes especially important here, because the error involved in the Pareto-dominance comparisons increases exponentially with the number of objectives in the worst case [7]. Knowles and Nakayama [8] present a review of meta-modeling methods in multi-objective optimization and discuss several related issues. Common meta-modeling techniques are response surface methods, kriging, neural networks, radial basis functions and support vector regression (SVM-R) [9]. A general survey of meta-models can be found in [10].

On the other hand, a *mono-surrogate method* uses a single surrogate for all objectives. Since ideally the output of a mono-surrogate model should be same for all solutions at any given non-dominated level and different for solutions from different levels, classification algorithms have been a natural choice; and among them only SVM based methods have been pursued so far, probably due to their popularity. The first attempt at a mono-surrogate strategy by Yun et al. [11] uses One-Class SVM [12] to learn a decision boundary that envelops the visited part of the objective space. Solutions with lower decision function values are closer to the boundary, while those with negative values are considered as anomalies. These anomalies are emphasized during evolution, with the rationale that the Pareto-optimal front is outside the visited region. However, Loshchilov et al. [13] note that such an approach can be used to guide evolution only in specific problems and propose the Aggregated Surrogate Model (ASM) which combines ideas from SVM-R and One-Class SVM. ASM maps all current non-dominated variable vectors to a narrow interval $[\rho - \epsilon, \rho + \epsilon]$ and all current dominated solutions to its left ($< \rho - \epsilon$). The Pareto-optimal front expectedly lies in the 'negative region' of One-Class SVM, which here is the half-space $> \rho + \epsilon$. A Rank-based ASM (RASM) was proposed in [7] by the same authors. RASM uses the rank-based SVM referenced above to learn from Pareto-dominance relations of the kind $\mathbf{x_i} \prec \mathbf{x_j}$ but not from those of the kind $\mathbf{x_i} || \mathbf{x_j}$. The training set of ASM consists of solutions categorized as non-dominated and dominated, where as that of RASM contains pairwise *dominance relations* between solutions[1]. Thus while ASM does not learn to differentiate between different levels of dominated solutions, RASM does not learn to recognize non-dominated solutions at a given non-domination level. Unlike Yun et al.'s approach, which is defined in the objective space, both ASM and RASM work in the decision space. Moreover, they are both implemented in the filter-based meta-modeling framework [14], where meta-models are used to pre-screen solutions rather than to replace fitness evaluations altogether. Pareto rank learning (PRL) [15] goes a step further and uses the ranks obtained by non-dominated sorting [16] for training a rank-based SVM. PRL also uses the filtering approach to perform real (expensive) function evaluations on solutions classified by it as rank one. All the above methods update the SVM model periodically using recently evaluated solutions. The method described in [17] however, generates the SVM model only once as a representation of the Pareto-optimal front and uses it throughout the optimization process. An artificial training set is generated in the objective space by 'improving' a few known near-Pareto optimal solutions twice, first for obtaining the negative instances (dominated solutions) and once again for the positive instances (non-dominated solutions). It should however be pointed out that the purpose of the study in [17] was only to test whether a one-time SVM representation carries sufficient information to guide the search effectively.

[1]The consideration of *non-dominance relations* was deferred for a future study.

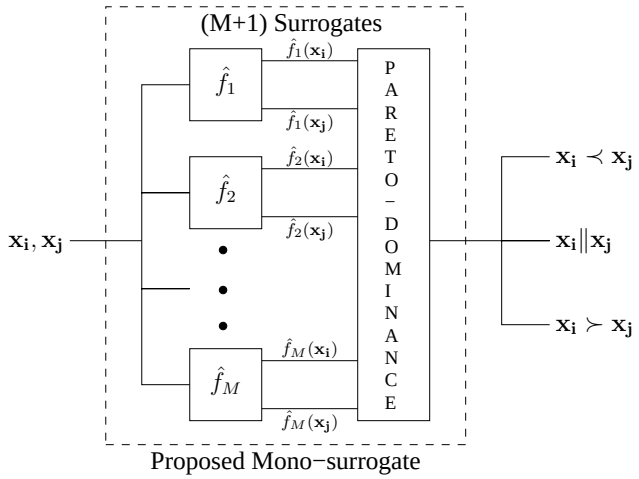

Fig. 1. Conceptualization of the proposed mono-surrogate using M objective surrogates $\hat{\mathbf{f}}$ and one Pareto-dominance surrogate.

In this paper, we propose a mono-surrogate strategy that uses multi-class classification. The paper is organized as follows. Section II describes the framework of the present approach. Multi-class classification algorithms used within this approach are briefly described in Section III. Next we present the experimental methodology in Section IV and finally discuss the results in Section V.

II. Classification based mono-surrogate

The present approach starts like any other surrogate method. The population is initialized randomly and actual (expensive) function evaluations are carried out. Selection and variation operators are used to produce offspring and individuals are ranked using Pareto-dominance principles. Thereafter, better ranked members are chosen to form the next generation population. The process continues until either an archive of pre-defined size is full or a pre-specified number of generations are executed, at which point the surrogate(s) is updated. In the proposed approach, the Pareto-dominance relations already established between different individuals of the current archive or the current population (as the case may be) serve as training instances. We use N to denote the size of this training set. For a pair of individuals $\mathbf{x_i}$ and $\mathbf{x_j}$, three possible Pareto-dominance relations exist (i) $\mathbf{x_i} \prec \mathbf{x_j}$, or (ii) $\mathbf{x_i} \succ \mathbf{x_j}$, or (iii) $\mathbf{x_i} || \mathbf{x_j}$. A multi-class classification algorithm can be trained using the N instances to classify new pairs of solutions into one of these three classes, thereby avoiding the need to evaluate objective functions and perform Pareto-dominance test. This *classification based mono-surrogate* approach can thus be thought of as a combination of $(M + 1)$ surrogates, M of which model the objective functions and a final surrogate which models the Pareto-dominance relation between two given individuals as shown in Figure 1. Like, ASM and RASM, the proposed approach also works in the decision space. However, it is

TABLE I. CHANGES IN CLASS SKEW (PERCENTAGE) WITH GENERATIONS FOR ZDT1 ($n = 5$)

gen	Class '$\prec$'	Class '$\succ$'	Class '$\|\|$'
2	1.26	24.99	73.75
20	0.27	0.28	99.45
40	0.19	0.17	99.64
60	0.16	0.18	99.67
80	0.15	0.16	99.68
100	0.11	0.13	99.76

important to understand that this method does not generate ranks (or a measure of rank) for individual solutions but simply predicts the Pareto-dominance relation between any two individuals from the population. As the population evolves, the spread of solutions in the decision space changes and the classification algorithm may need to be re-trained just like any other surrogate based methods.

In this preliminary study, our aim is only to compare various multi-class classification algorithms in terms of their accuracy and training times at various stages of optimization and for various problem sizes.

- **Optimization stage:** As optimization proceeds, the proportion of training instances belonging to the three classes changes. The class representing the non-dominance relation (i.e. $\mathbf{x_i}||\mathbf{x_j}$) grows in size with generations as the population moves towards the Pareto-optimal front. Table I shows an example of how the class proportions change with generations in ZDT1 for $n = 5$ variables. The fact that $O(MN^2)$ non-dominated sorting presented in [16] compares non-dominated solution pairs more often than other solution pairs contributes to this class skew. The classification algorithms should therefore be robust enough to handle class skewness in the training set.
- **Problem size:** The number of objectives and the number of variables also effect classification. It is well-known that with higher number of objectives a greater proportion of solutions are non-dominated. This further adds to the class skew discussed above. Moreover, higher number of variables means more features to be taken into account. Since the proposed mono-surrogate takes two solution vectors as input, doubling the number of variables quadruples the number of features. This results in increased training times.

While the integration of the mono-surrogate with optimization is left for a future study, the results of this study can guide the selection of an appropriate classifier for the mono-surrogate in specific cases. Next, we briefly describe ten off-the-shelf classification algorithms used with the above proposed mono-surrogate.

III. MULTI-CLASS CLASSIFICATION

In the field of supervised machine learning, classification refers to the task of training a computer program using a set of *instances* (training set) with known *class memberships*. The user chooses certain *features* of the instances that may effect its classification and the program 'learns' the mapping between these features and the classes. Once trained, the performance of the *classifier* is defined by its ability to correctly predict the class membership of a new instance (from test set) previously 'unseen' by the program during training. Many classifiers have been proposed in the machine learning literature [18] and the No Free Lunch theorem applies, meaning that for each application, a range of classifiers should be experimented with before choosing one.

Classifiers are often developed with binary classification in mind, i.e. when each instance can belong to one of two classes. Multi-class classification deals with problems involving $K > 2$ classes. Binary classifiers that output posterior probabilities can directly be extended for multi-class classification. The test instance is assigned to the class with the largest posterior probability. When the output of the binary classifier is not calibrated, voting mechanisms are used instead. In 'one-vs-all' voting, K binary classifiers are trained to distinguish each class from rest of the classes. On the other hand, the 'one-vs-one' approach builds $\binom{K}{2}$ binary classifiers to distinguish each class from every other class. The following sections briefly describe the ten multi-class classifiers used in this study.

A. Multinomial Logistic Regression

Logistic regression uses the logistic (or sigmoid) function to define the probability, in terms of a linear combination of D features $\mathbf{X}$, that an outcome Y is one of two classes $\{0, 1\}$, i.e.

$$Pr(Y = 1|\mathbf{X}) = \frac{1}{1 + e^{-(\beta_0 + \boldsymbol{\beta}^T\mathbf{X})}}. \tag{3}$$

Since $Pr(Y = 1|\mathbf{X}) + Pr(Y = 0|\mathbf{X}) = 1$, the above equation can be expressed as,

$$\ln\left(\frac{Pr(Y = 1|\mathbf{X})}{Pr(Y = 0|\mathbf{X})}\right) = \beta_0 + \boldsymbol{\beta}^T\mathbf{X}, \tag{4}$$

where $\beta_0 + \boldsymbol{\beta}^T\mathbf{X}$ represents the *decision boundary* between the two classes. The logit model in Eq. (4) can be extended to K classes using $K - 1$ independent decision boundaries as,

$$\ln\left(\frac{Pr(Y = k|\mathbf{X})}{Pr(Y = K|\mathbf{X})}\right) = \beta_{0k} + \boldsymbol{\beta}_\mathbf{k}^T\mathbf{X} \quad \forall\, k = \{1, \ldots, K - 1\}, \tag{5}$$

where the probability of the outcome being the K-th class is taken as reference in the denominator. The $K - 1$ coefficients β_{0k} and $(K - 1) \times D$ coefficients $\boldsymbol{\beta}_\mathbf{k}$ are estimated using maximum likelihood.

B. Support Vector Machines

Support vector machines are primarily binary classifiers that divide instances using a linear decision boundary (hyperplane) representing maximal separation or margin between their classes $Y = \{-1, 1\}$. Instances that are closest to the hyperplane on either side of it are called support vectors and they satisfy,

$$\left.\begin{aligned}\mathbf{w}^T\mathbf{X}_i - b = +1 \quad \forall\, Y_i = +1\\ \mathbf{w}^T\mathbf{X}_i - b = -1 \quad \forall\, Y_i = -1\end{aligned}\right\} \; Y_i(\mathbf{w}^T\mathbf{X}_i - b) = 1, \tag{6}$$

where $\mathbf{w}$ is the normal vector to the hyperplane and $b/\|\mathbf{w}\|$ is its distance from the origin. In general, all instances satisfy $Y_i(\mathbf{w}^T\mathbf{X}_i - b) \geq 1$. In *soft margin* SVMs, exceptions are allowed using non-negative slack variables ξ_i as follows,

$$Y_i(\mathbf{w}^T\mathbf{X}_i - b) \geq 1 - \xi_i \quad \forall\, i = \{1, \ldots, N\} \tag{7}$$

The maximal soft margin hyperplane is obtained by minimizing $\|\mathbf{w}\|$ while penalizing non-zero ξ_i with a penalty $C > 0$, i.e.

$$\begin{array}{ll} \min\limits_{\mathbf{w},b,\boldsymbol{\xi}} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} & Y_i(\mathbf{w}^T\mathbf{X}_i - b) \geq 1 - \xi_i \quad \forall\, i = \{1,\ldots,N\} \\ & \xi_i \geq 0 \quad \forall\, i = \{1,\ldots,N\} \end{array} \tag{8}$$

Since N is usually large, the dual form of the above optimization problem is solved. SVMs can 'act' as non-linear classifiers using the kernel trick which basically maps feature vectors X_i to a high-dimensional space where the decision boundary can be linear. LIBSVM's [19] implementation of one-vs-one multi-class SVM classification is used in this paper with $C = 1$ and $\sigma = 1$ for the Gaussian radial basis function kernel $\kappa(\mathbf{X_i}, \mathbf{X_j}) = \exp(-(\|\mathbf{X_i} - \mathbf{X_j}\|^2)/2\sigma^2)$.

C. Multi-layered Perceptron

Multi-layered perceptrons, or more commonly artificial neural networks, can be used for a variety of purposes, like function approximation, pattern recognition and clustering. A neural network consists of interconnected units or neurons arranged into different layers namely input, hidden and output layers. Each connection between neurons carries a weight. For classification, the input layer takes features from the training set. Each neuron in the hidden and output layers accepts a weighted sum of outputs from the previous layer and generates its own output using the *activation function*. The number of neurons in the output layer depends on the purpose of the neural network. Approximation of a scalar function require just one output neuron. For multi-class classification with C classes, typically C output neurons are used. Starting with randomly initialized values, the network weights are *learned* iteratively using the training set. In the standard *back-propagation algorithm*, each time a training instance is evaluated, the error at the output layer is back propagated to update the weight values. More sophisticated algorithms, like Levenberg-Marquardt and Scaled Conjugate Gradient, perform network learning from an optimization point of view, using search directions, line searches and error functions. In this paper we use a neural network model with D neurons in the input layer for the D features, 10 neurons in one hidden layer and three neurons in the output layer each representing a class. The sigmoid activation function is used at all hidden and output layer neurons.

D. Classification Trees

Classification trees are a variant of decision trees where the outcomes at the end nodes (leaves) belong to one of the C classes. A classification tree consists of a root node where all training instances are present. The root node splits the training set into two subsets represented by two child nodes. The process continues until all instances in a node belong to one class. Each node uses a *split criterion* to select a feature and a corresponding value, on the basis of which the two subsets are formed. Popular split criteria are the Gini impurity index, information gain and the twoing rule. In this work, we use the Gini impurity index is given by, $G(t) = 1 - \sum_{k=1}^{K}[f(k|t)]^2$, where $f(k|t)$ is the fraction of instances belonging to class k at the given node t. The split is based on the condition v (combination of a feature and a value) which maximizes the Gini gain Δ, between the parent node and its child nodes. It is given by,

$$\Delta = G(parent) - \sum_{v \in V} f_{parent,v} G(child|v), \tag{9}$$

where V is the set of all possible conditions obtained from the features and their sorted values, $f_{parent,v}$ is the fraction of instances in parent node that satisfy condition v and $G(child|v)$ is the Gini index of the child node satisfying v. We use Matlab's® `classregtree` function which also performs tree pruning to avoid over-fitting.

E. Naive Bayes Classifier

The naive Bayes classifier uses Bayes rule to calculate the probability that an instance $\mathbf{X}$ belongs to class k, i.e.,

$$Pr(Y = k|\mathbf{X}) = \frac{Pr(Y = k)Pr(\mathbf{X}|Y = k)}{\sum_{i=1}^{K} Pr(Y = i)Pr(\mathbf{X}|Y = i)}. \tag{10}$$

This classifier makes the 'naive' assumption that the features are *conditionally independent* of each other, which is almost always wrong. Thus, Eq. (10) can be written as,

$$Pr(Y = k|\mathbf{X}) = \frac{Pr(Y = k)\Pi_{j=1}^{D} Pr(X_j|Y = k)}{\sum_{i=1}^{K} Pr(Y = i)\Pi_{j=1}^{D} Pr(X_j|Y = i)}, \tag{11}$$

where $\mathbf{X} = [X_1, X_2, \ldots X_D]^T$. The probability distribution on the right hand side of Eq. (11) are estimated from the training data. For real-valued features, the probability distributions for each class i with each feature j, i.e. $Pr(X_j|Y = i)$, are often assumed to be Gaussian, which involves two parameters, namely the mean μ and the standard deviation σ. Thus, in all $2DK$ parameters are to be estimated from the training set. Maximum likelihood estimates are used commonly.

F. k Nearest Neighbor

k nearest neighbor or k-NN is the simplest of all the classifiers studied in this paper. Unlike other learning algorithms, k-NN defers all computations until a new instance is to be classified. Therefore, it is also known as lazy learner. Given a test instance, it approximates the mapping locally by considering k nearest neighbors of the instance in the feature space from the training set. The distance measure depends on the type of features. We use Euclidean distance since all features are real and continuous. Thereafter, a voting strategy is employed to predict the class of the instance. In majority voting, the predicted class is the class to which a majority of the of the k neighbors belong. In case of a tie between two or more classes, the predicted class is the one which contains the training instance closest to the test instance. In general however, majority voting is not recommended when class distribution is skewed. This study uses $k = D$ neighbors.

G. Linear Discriminant Analysis

Linear discriminant analysis (LDA), also sometimes known as Fisher's linear discriminant, attempts to find a linear transformation $y = \mathbf{w}^T\mathbf{X}$ that gives maximal separation between the projected instances of two classes. The probability distribution of each class ($i = 1$ and $i = 2$) is assumed to be normal,

characterized by mean $\boldsymbol{\mu}_i$ and covariance $\mathbf{S}_i$, which become $\mathbf{w}^T\boldsymbol{\mu}_i$ and $\mathbf{w}^T\mathbf{S}_i\mathbf{w}$ in the projected space. The separation in the projected space is defined by the ratio of variance *between* classes to the variance *within* classes, i.e.,

$$S(\mathbf{w}) = \frac{(\mathbf{w}^T\boldsymbol{\mu}_1 - \mathbf{w}^T\boldsymbol{\mu}_2)^2}{\mathbf{w}^T\mathbf{S}_1\mathbf{w} + \mathbf{w}^T\mathbf{S}_2\mathbf{w}} = \frac{\mathbf{w}^T\mathbf{S_B}\mathbf{w}}{\mathbf{w}^T\mathbf{S_W}\mathbf{w}}. \quad (12)$$

Here, $\mathbf{S_B} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ and $\mathbf{S_W} = \mathbf{S}_1 + \mathbf{S}_2$. It can be shown that $S(\mathbf{w})$ is maximized when $\mathbf{w} = {\mathbf{S_W}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$.

LDA can be generalized to K classes using $K-1$ projections, which can be calculated individually as above or more gracefully by arranging $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{K-1}$ as columns of the projection matrix $\mathbf{W}$. In this case, the separation is defined by,

$$S(\mathbf{W}) = \frac{|\mathbf{W}^T\mathbf{S_B}\mathbf{W}|}{|\mathbf{W}^T\mathbf{S_W}\mathbf{W}|}, \quad (13)$$

where, $\mathbf{S_B} = \sum_{i=1}^{K}(\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$, $\boldsymbol{\mu}$ is the mean of class means and $\mathbf{S_W} = \sum_{i=1}^{K}\mathbf{S}_i$. Again it can be shown that multi-class separation is maximized by the eigenvectors corresponding to the largest eigenvalue of ${\mathbf{S_W}}^{-1}\mathbf{S_B}$. A new instance $\mathbf{X}$ is assigned to the class whose projected mean $(\mathbf{W}^T\boldsymbol{\mu}_i)$ is closest to $\mathbf{W}^T\mathbf{X}$.

H. Quadratic Discriminant Analysis

Quadratic discriminant analysis (QDA) uses a generalized form of LDA in which the covariances of different classes are not equal. As a result more flexible decision boundaries between the classes can be obtained. However, since QDA estimates more parameters, the variances can be high.

I. Random Forests

A random forest is basically a collection of classification trees. Each classification tree uses the complete training set with the only difference that at each node the feature to be used in the split criterion is not chosen from the complete set of features but from a randomly selected subset of the feature vector. For classification problems the recommended size of this subset is $\sqrt{D}$. The predicted class for a given instance is the mode of class predictions of all classification trees in the forest. In this study, we choose the number of trees as 10.

J. Ensemble

The ensemble method used in this paper predicts the mode (most frequent) of class predictions of the above nine classifiers and a random classifier. Assuming the classifiers are run parallely, the training time is taken to be the maximum of all the classifiers.

IV. Experimental Methodology

As discussed before, the performance of the above described classification algorithms is studied with respect to (i) problem size, and (ii) class skewness. For varying the problem size, we use the ZDT test suite [20] with different number of variables, i.e. n = 5, 10, 15 and 20. In a future study, the number of objectives can also be varied using an appropriate test suite. Optimization is performed using NSGA-II [16] with the following parameter settings:

1) Population size = 100
2) Number of generations = 100
3) SBX crossover probability $p_c = 0.9$, distribution index $\eta_c = 15$
4) Polynomial mutation probability $p_m = 1/n$, distribution index $\eta_m = 20$

Since we are concerned with the performance of classification algorithms and not that of NSGA-II, we consider one particular NSGA-II run for some random seed. Each classification algorithm is trained and tested at different stages of optimization, i.e. at gen = 2, 20, 40, 60, 80 and 100, to study the effect of class skewness. At each of these stages, the Pareto-dominance relations obtained through pairwise comparisons performed by the non-dominated sorting routine are recorded for training and testing.

A. Cross-validation

Many different cross-validation methods are available in literature. We use the popular k-fold cross-validation to estimate the generalization performance of each classification algorithm. The pairwise comparisons recorded above are randomly divided into k nearly equally sized parts (or folds) with stratification, which ensures that the class proportions in each fold are roughly the same. The first fold is held-out and the remaining $k-1$ folds are used to train the algorithm in question. The performance of the trained algorithm is then evaluated for the first fold. This process is repeated k times, each time holding-out one fold for evaluation and using the other folds to train the classifier. The k performance metrics thus obtained are averaged to get the mean estimate of the performance for that particular classifier, for the test problem under consideration at a given problem size and optimization stage. We use $k = 10$ in this study.

B. Performance Criteria

The ten classification algorithms are compared with respect to two estimated performance metrics, the misclassification rate or error rate (ϵ) and the training time (τ). The former measures the accuracy, i.e. $\epsilon = \frac{\text{misclassified test instances}}{\text{total test instances}}$, while the latter measures the speed of the algorithm in seconds. It has been argued in machine learning literature, whether or not misclassification rate is a good accuracy measure, especially when dealing with datasets having considerable class skew [21]. However, other performance measures [22] are equally susceptible to class skew [21]. An alternate scalar measure that is unattenuated by skewed class distributions is the area under the ROC (Receiver Operating Characteristic) curve, often abbreviated as AUC (Area Under Curve) [23]. However, there are two reasons why we don't use AUC in this paper. Firstly, the generation of ROC curves requires a *class-discrimination threshold*, which is not available for all classification algorithms. Secondly, the generalization of ROC (and AUC) to multi-class classification is still debated. Hence, despite its shortcomings, we have chosen to use misclassification rate, accompanied by the misclassification rate of a random classifier to serve as baseline.

C. Feature Vector and Output

Note that in the proposed approach the feature vector $\mathbf{X}$ of length D is simply a juxtaposition of the two solution vectors,

TABLE II. LEGEND FOR PARETO CHARTS

Algorithm	Symbol
Random Classifier	○
Multinomial Logistic Regression	×
Support Vector Machine	+
Neural Networks	✳
Classification Tree	□
Naive Bayes Classifier	◇
k Nearest Neighbors	▽
Linear Discriminant Analysis	△
Quadratic Discriminant Analysis	◁
Random Forest	▷
Ensemble	☆

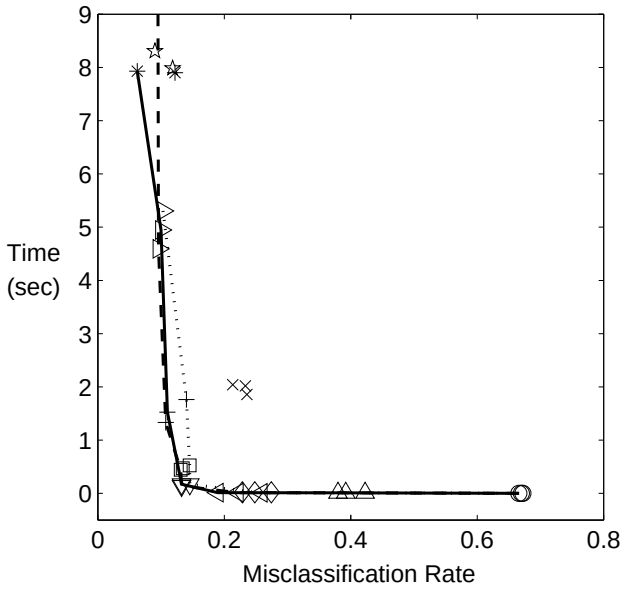

Fig. 2. Misclassification rate vs training time at $gen = 2$ for all algorithms with test problem size $n = 5$.

i.e. $\mathbf{X} = \begin{bmatrix} \mathbf{x_i} \\ \mathbf{x_j} \end{bmatrix}$, and therefore $D = 2n$. The outputs relations $\prec$, $\|$ and $\succ$ are assigned categorical labels.

V. RESULTS

The estimated mean misclassification rates and training times obtained after k-fold cross validation can be shown on Pareto charts. Figures 2-5 show the trade-off between the two performance criteria at the second generation of NSGA-II for ZDT1, ZDT2 and ZDT3 with n = 5, 10, 15 and 20 variables. Table II shows the legend used for the algorithms:

The Pareto-efficient classification algorithms in all cases are connected using (i) a continuous line for ZDT1, (ii) a dashed line for ZDT2 and (iii) a dotted line for ZDT3. The dominated algorithms clearly stand out as points to the right of the Pareto-efficient front. Some important observations from these figures are as follows:

1) The random classifier has the worst performance among all algorithm with respect to accuracy. This assures us that the learning algorithms are being trained properly. Even with significant class skew, as shown in Table I, the implemented algorithms perform better than the baseline random classifier, which is expected.

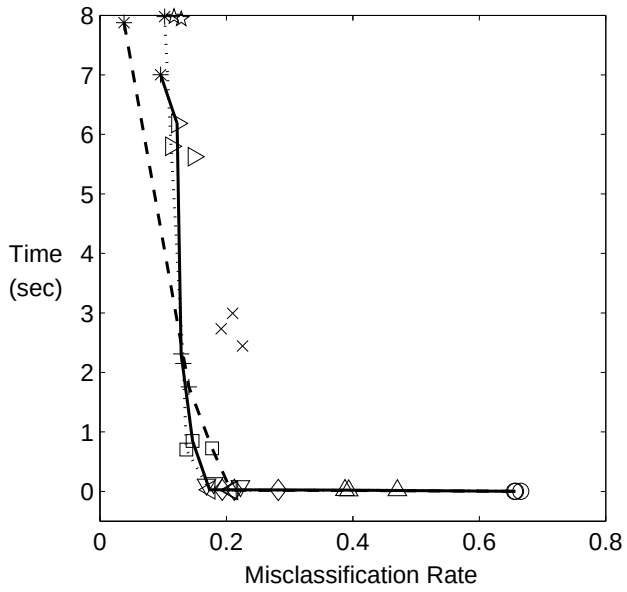

Fig. 3. Misclassification rate vs training time at $gen = 2$ for all algorithms with test problem size $n = 10$.

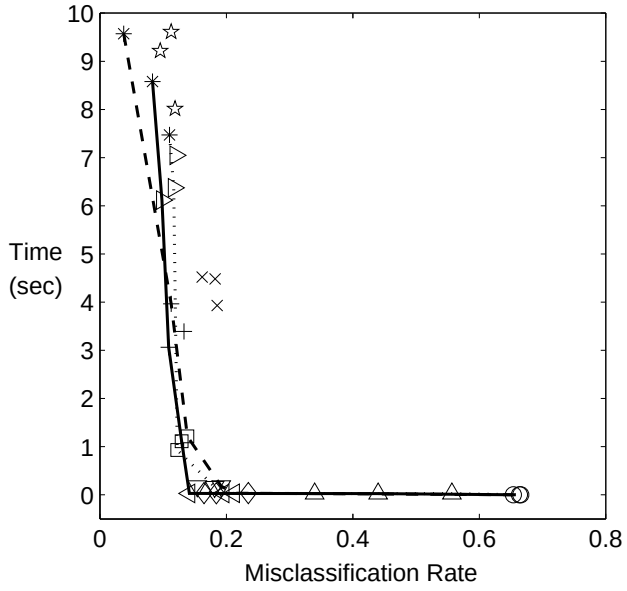

Fig. 4. Misclassification rate vs training time at $gen = 2$ for all algorithms with test problem size $n = 15$.

2) LDA is the worst performing algorithm among all learning algorithms. However, it is also the fastest.
3) Neural networks and the ensemble approach provide the best accuracy but are also the most time taking of all algorithms.
4) Random forest performs better than neural networks in terms of training times. They are second best when it comes to classification accuracy.
5) Multinomial logistic regression is a dominated algorithm for all three ZDT problems considered here. Thus, it should never be used with the mono-surrogate approach proposed in this paper.
6) The algorithms found on the knee region of the

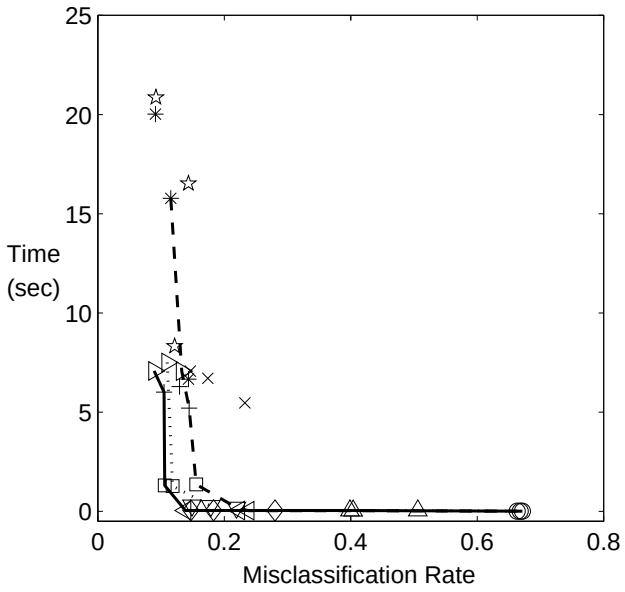

Fig. 5. Misclassification rate vs training time at $gen = 2$ for all algorithms with test problem size $n = 20$.

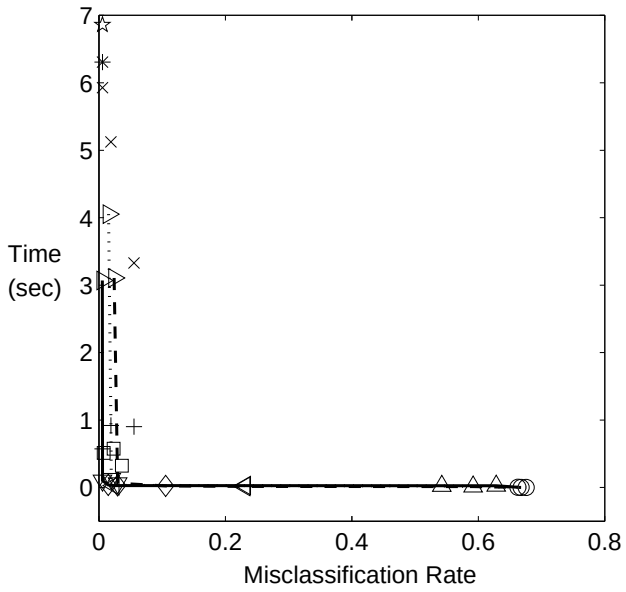

Fig. 6. Misclassification rate vs training time at $gen = 20$ for all algorithms with test problem size $n = 5$.

Pareto-efficient curve are SVM, classification trees, k-NN and to some extent QDA. For the purpose studied in this paper, these algorithms should be the most preferred.

The Pareto-efficient charts discussed above show sufficient trade-off between accuracy and training time in the initial generations. Thus, if more accurate predictions are required, neural networks and random forests may be used at the expense of additional training time. However, it is observed that this trade-off vanishes in subsequent generations. For example, consider the Pareto-efficient plot at $gen = 20$ as shown in Figure 6. Clearly, the choice of classification algorithm should be one of those at the knee region, since those at the extremities do not offer better accuracy or training times.

Next, we consider the individual performance of each algorithm with generations for different problem sizes. For illustration we choose SVM, since it is the only classification algorithm studied previously for surrogate modeling, as discussed in Section I. Moreover, from the discussion above SVM should be one of the preferred algorithms in the present mono-surrogate approach. Figure 7 shows the misclassification rate with generations for various problem sizes of ZDT3. Now consider a similar plot for training time as shown in Figure 8. From Figures 7 and 8 it is observed that problem size does not effect the accuracy of SVM as much as it effects the training time. Another interesting observation from Figure 8 is that a higher increase in SVM training time (with problem size) is required in the initial generations when the class proportions are not too skewed. Similar analysis can be performed on other algorithms to choose the best classification algorithm when training time is not an issue. For example, neural networks are highly parallelizable and, as observed in the Pareto-efficient charts, more accurate than other learning algorithms.

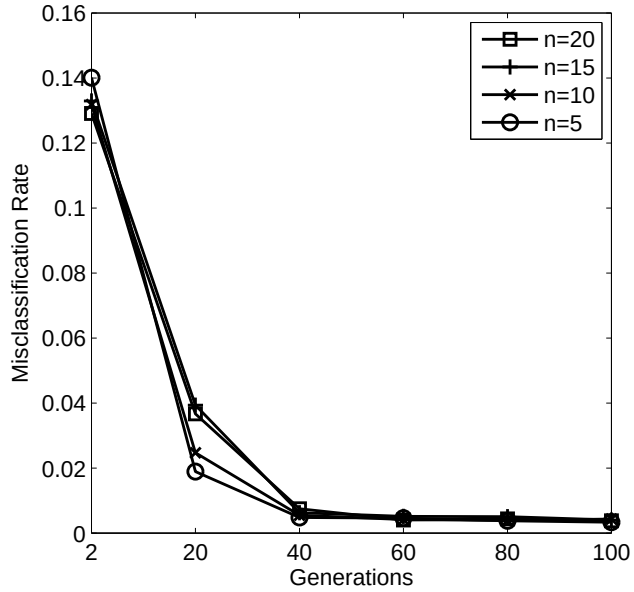

Fig. 7. SVM misclassification rate vs generations for various problem sizes of ZDT3.

VI. Conclusions

In this paper, we proposed a new mono-surrogate strategy which uses multi-class classification algorithm to establish Pareto-dominance relations between pairs of individuals from a population. The surrogate can be used to rank solutions without the need to evaluate expensive objective functions and perform Pareto-dominance tests. The misclassification rates and training times of ten popular classification algorithms were obtained though systematic experimentation, which involved scaling the number of variables for ZDT problems and training and testing the algorithms at various stages during the optimization with NSGA-II. The results of this study show that as far as modeling the optimization objectives and Pareto-

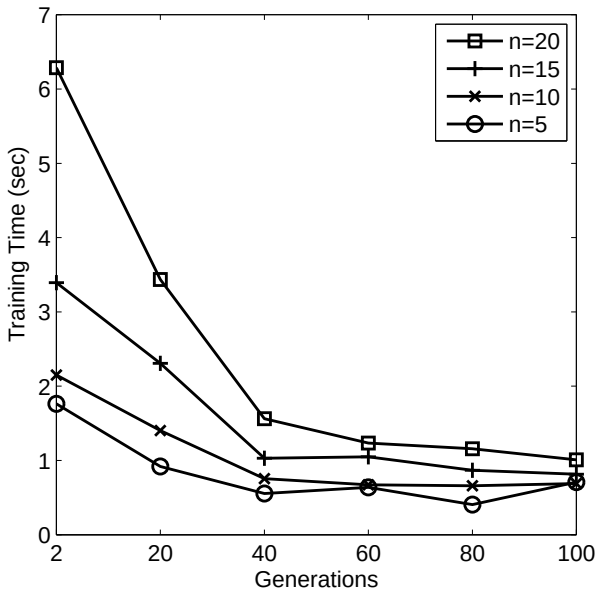

Fig. 8. SVM misclassification rate vs generations for various problem sizes of ZDT3.

dominance tests is concerned, some classification algorithm are clearly better than others in terms of both accuracy and speed. Among them are SVM, k-NN and classification trees. Random forests expectedly require more training time, however, the voting from multiple trees improves its prediction accuracy. The biggest revelation for the authors is that logistic regression is dominated in terms of both performance criteria. Through the use of Pareto charts we have quantitatively shown the relative performance of all algorithms.

The immediate extension to this study is the integration of the proposed mono-surrogate with an MOEA. Also, as discussed previously, scaling the number of objectives increases class skew significantly. Studying the performance of these algorithms under such conditions will be crucial for the proposed mono-surrogate approach to be used in many-objective optimization. In this study, we overcame the weakness of misclassification rate as a performance measure using a baseline random classifier. However, as and when a better performance indicator for algorithm accuracy is proposed, it should be used in place of the misclassification rate.

REFERENCES

[1] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, no. 1, pp. 3–12, Oct. 2003.

[2] ——, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, Jun. 2011.

[3] T. P. Runarsson, "Constrained Evolutionary Optimization by Approximate Ranking and Surrogate Models," *PPSN VIII*, pp. 401–410, 2004.

[4] T. Runarsson, "Ordinal regression in evolutionary computation," in *PPSN IX*, 2006, pp. 1048–1057.

[5] R. Herbrich, T. Graepel, and K. Obermayer, "Large Margin Rank Boundaries for Ordinal Regression," in *Advances in Large Margin Classifiers*, A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, Eds. Cambridge, MA: {MIT} Press, 2000, pp. 115–132.

[6] I. Loshchilov, M. Schoenauer, and M. Sebag, "Comparison-based optimizers need comparison-based surrogates," *PPSN XI*, pp. 364–373, 2010.

[7] ——, "Dominance-based pareto-surrogate for multi-objective optimization," in *Simulated Evolution and Learning*, 2010, pp. 230–239.

[8] J. Knowles and H. Nakayama, "Meta-modeling in multiobjective optimization," *Multiobjective Optimization*, vol. 5252, pp. 245–284, 2008.

[9] A. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, pp. 199–222, 2004.

[10] T. W. Simpson, J. D. Poplinski, P. N. Koch, and J. K. Allen, "Metamodels for computer-based engineering design: Survey and recommendations," *Engineering with Computers*, vol. 17, no. 2, pp. 129–150, 2001.

[11] Y. Yun, H. Nakayama, and M. Arakava, "Generation of Pareto frontiers using support vector machine," in *MCDM04*, 2004.

[12] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, a. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution." *Neural computation*, vol. 13, no. 7, pp. 1443–71, Jul. 2001.

[13] I. Loshchilov, M. Schoenauer, and M. Sebag, "A mono surrogate for multiobjective optimization," *GECCO '10 Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 471–478, 2010.

[14] M. Emmerich, K. Giannakoglou, and B. Naujoks, "Single- and multiobjective evolutionary optimization assisted by Gaussian random field metamodels," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 421–439, Aug. 2006.

[15] C.-W. Seah, Y.-S. Ong, I. W. Tsang, and S. Jiang, "Pareto Rank Learning in Multi-objective Evolutionary Algorithms," in *2012 IEEE Congress on Evolutionary Computation*. Ieee, Jun. 2012, pp. 1–8.

[16] K. Deb, S. Agarwal, A. Pratap, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: {NSGA-II}," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[17] H. Aytu and S. Sayn, "Using support vector machines to learn the efficient set in multiple objective discrete optimization," *European Journal of Operational Research*, vol. 193, no. 2, pp. 510–519, Mar. 2009.

[18] S. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, pp. 249–268, 2007.

[19] C. Chang and C. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, pp. 1–39, 2011.

[20] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: empirical results." *Evolutionary computation*, vol. 8, no. 2, pp. 173–95, Jan. 2000.

[21] L. Jeni, J. Cohn, and F. D. L. Torre, "Facing Imbalanced Data–Recommendations for the Use of Performance Metrics," *Affective Computing and ...*, 2013.

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

[23] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters*, vol. 27, pp. 861–874, 2006.