Towards Effective GP Multi-Class Classification Based on Dynamic Targets

Stefano Ruberto
European Commission Joint Research Centre (JRC), Ispra, Italy
stefano.ruberto@ec.europa.eu

Valerio Terragni
University of Auckland
Auckland, New Zealand
v.terragni@auckland.ac.nz

Jason H. Moore
University of Pennsylvania
Philadelphia, USA
jhmoore@upenn.edu

ABSTRACT

In the multi-class classification problem GP plays an important role when combined with other non-GP classifiers. However, when GP performs the actual classification (without relying on other classifiers) its classification accuracy is low. This is especially true when the number of classes is high. In this paper, we present DTC, a GP classifier that leverages the effectiveness of the dynamic target approach to evolve a set of discriminant functions (one for each class). Notably, DTC is the first GP classifier that defines the fitness of individuals by using the synergistic combination of linear scaling and the hinge-loss function (commonly used by SVM). Differently, most previous GP classifiers use the number of correct classifications to drive the evolution. We compare DTC with eight state-of-art multi-class classification techniques (e.g., RF, RS, MLP, and SVM) on eight popular datasets. The results show that DTC achieves competitive classification accuracy even with 15 classes, without relying on other classifiers.

CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**; **Supervised learning by classification**; **Machine learning**; *Artificial intelligence*;

KEYWORDS

Genetic Programming, Multiclass classifier, Semantic GP, Dynamic Targets, Evolutionary algorithms, Supervised learning

ACM Reference Format:
Stefano Ruberto, Valerio Terragni, and Jason H. Moore. 2021. Towards Effective GP Multi-Class Classification Based on Dynamic Targets. In *2021 Genetic and Evolutionary Computation Conference (GECCO '21), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3449639.3459324

1 INTRODUCTION

Multi-class classification is one of the most important research topics in machine learning [2], and plays a crucial role in many modern applications. When dealing with the multi-class classification problem, Genetic Programming (GP) is a fundamental building block for constructing effective classifiers. For instance, GP has a pivotal role in automating the architecture engineering activity of neural networks [10, 14, 19, 23, 53, 56], or to select/construct features for other non-GP classifiers [30, 37, 47, 67]. However, when GP alone is used to evolve a multi-class classifier (not combined with other classifiers) it has a low classification accuracy [20]. Indeed, several researchers reported poor performance of GP classifiers [11, 35, 37, 66, 73], especially when more than two classes are involved.

In this paper, we present ***D**ynamic **T**arget **C**lassifier* (**DTC**), a GP classifier that learns multiple discriminant functions, one for each class to predict. DTC is powered by the dynamic target approach [44, 45, 61] that divides the search problem into a sequence of GP runs. Each run returns a *partial model* that focuses on the portion of the problem that the previous runs did not capture well [61]. Ruberto et al. show that such an approach is more effective than evolving a single model [61], achieving interesting results in the domains of symbolic regression [61, 62] and image feature learning [59, 60].

We compare DTC with eight state-of-art classifiers (i.e., RF, RS, MLP, SVM, and three GP *wrapper approaches* [37, 66]) on eight popular datasets. The results show that DTC achieves competitive (and often even better) classification accuracy even with 15 classes. Notably, DTC employs GP to perform the actual classification and does not rely on other classifiers.

DTC presents a novel combination of well-known techniques that, to the best of our knowledge, have never been used in a GP classifier. First, before computing the fitness of an individual, DTC uses *linear scaling* [32] to optimize the individual with respect to the decision boundary of the discriminant function. Second, to compute the fitness, DTC relies on the *hinge-loss function* [58] (commonly used by SVM [13]) instead of canonical functions that maximize the number of correct classifications [20]. The hinge loss function has the advantage of penalizing those individuals that correctly classify instances whose prediction is close to the decision boundary [58].

Our proposed fitness function works well in synergy with the dynamic target approach. On one hand, the hinge-loss function focuses only on the portion of the problem in which the current classification model is uncertain [58]. On the other hand, each dynamic target focuses on the portions of the problem not optimized by previous GP runs [61].

The following section describes the background of multi-class classification with GP, and discusses the limitations of current approaches. Section 3 describes DTC in detail, highlighting its novel aspects. Section 4 describes the experiments that we conduct on eight popular datasets for the multi-class classification problem. Section 5 reports the results of the experiment comparing DTC with eight state-of-the-art classifiers. Section 6 concludes the paper highlighting promising future work, and discusses what our results entail for future studies on GP-based multi-class classification.

GECCO '21, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8350-9/21/07.
https://doi.org/10.1145/3449639.3459324

2 BACKGROUND AND RELATED WORK

Multi-class classification is one of the fundamental problems in Machine Learning [2, 20]. It is the problem of predicting the value of categorical attributes $C = \{c_1, c_2, \cdots c_k\}$ (the classes), based on the values of other p attributes [20]. In this paper, we assume classification problems with two or more classes ($k \geq 2$).

We consider *supervised learning*, in which a search algorithm induces a **classifier** from a set of correctly classified data instances $\mathcal{T} = \{(\vec{x}_i, y_i), i = 1, 2, \cdots n\}$ (training set), where $\vec{x}_i \in \mathbb{R}^p$ is the vector of p attributes.

Neural Networks (NNs) [28] have shown to be very effective in the multi-class classification problem, especially in the image classification domain [64]. Because developing NNs requires labor-intensive architecture engineering, researchers have investigated GP approaches to automate the architecture engineering activity of NNs. For example, the Neuroevolution of Augmenting Topologies (NEAT) techniques [3–5, 8, 10, 14, 43, 53, 56].

In this paper, we present a GP-based classifier that does not rely on other classifiers (e.g., NNs). Applying GP to the multi-class classification could bring some interesting advantages with respect to non-GP classifiers: (i) GP often does not need large training sets to learn competitive models [52], (ii) GP is generally robust to noisy data [42, 59, 60], and (iii) the intrinsic flexibility of GP could lead to classifiers that easily adapt to the specific problem at hand [20].

GP Classifiers. Previous GP classifiers investigated various strategies and model representations: decision trees [33, 70], classification rules [31, 46], and discriminant functions [12, 40, 41, 54]. Interested readers can refer to the survey of Espejo et al. for a literature review on this topic [20]. In this paper, we are targeting Genetic Programming (GP) classifiers represented as discriminant functions. A **discriminant function *f*** is a mathematical expression in which different kinds of operators and functions are applied to the attributes of the instance to be classified, i.e., $f : R^n \rightarrow C, f(\vec{x}) = c_j$.

We now discuss the relevant examples of GP classifiers that use discriminant functions. Paul and Iba propose a GP classifier for binary classification ($|C| = 2$) that learns multiple discriminant functions to classify the training set into two classes [54]. It then classifies instances with a majority voting. Differently, DTC targets multiclass classification with two or more classes ($|C| > 2$), and produces a single discriminant function for each class. Chen and Lu propose a GP classifier that produces a set of single-threshold discriminant functions [12]. Each of these functions classifies an instance in one or more classes. There could be multiple functions associated with a single class. To classify an instance, this approach considers the majority voting scheme. Lin et al. present the Layered Genetic Programming (LAGEP) classifier [40, 41], where each layer evolves a portion of the discriminant function, which is progressively constructed layer-by-layer. Similarly to LAGEP, DTC also progressively constructs the discriminant function. However, Lin et al. evaluated LAGEP mostly for the binary classification problem and using only two layers, while DTC shows competitive results even with 15 classes and using tens of partial models.

Early attempts of GP classifiers based on discriminant functions have shown poor performance [11, 35, 73] (when compared to other state-of-the-art classifiers). In fact, Krzysztof Krawiec remarks that *"in most real-world cases, it is rather unreasonable to expect the GP individuals to evolve into complete, well-performing classifiers, even for the two-class discrimination problem"* [35]. Castelli et al. and Silva and Tseng, reached a similar conclusion: the performance of GP classifiers is drastically reduced with the increasing number of classes [11, 66].

GP Feature selection/construction for multiclass classifier. Because of the poor performance of GP classifiers, GP-based constructive induction of features has been investigated as a good compromise to employ GP for multi-class classification [35]. More specifically, researchers have investigated the use of GP as a preprocessing step for a multi-class classifier (e.g., SVM, Random Forest, Bayesian approaches, and NN). The idea is to use GP to perform feature selection and construction to improve the discriminating power of the features, and thus improving the performance of a classifier. Following the taxonomy of Espejo et al. [20], such techniques can be divided into *filter* [48, 51] and *wrapper* approaches [1, 29, 30, 37, 47, 67, 69]. Both approaches use GP to explore the feature space by combining and selecting features. Filtering approaches rely on statistical properties (e.g., class scattering and information entropy) to compute the fitness, and do not use classification results to guide the evolution. Instead, wrapper approaches rely on the performance of the classifier to compute the fitness, and thus they are more related to DTC.

Wrapper approaches have gained popularity in recent years, showing promising results [37]. Miranda et al. propose a wrapper approach and evaluate it with five different classifiers (K Nearest Neighbors, Naive Bayes, SVM, Decision Tree, and Multilayer Perceptron) in the context of electroencephalogram analysis [47]. The results show better performance when compared to feature selection/construction using PCA. Similarly, Sotto et al. improve the performance of Random Forest classifiers by using GP to select and construct features [67]. The techniques of Howley et al. [29] and Sullivan et al. [69] use GP to build Kernel Functions for SVM classifiers. Agapitos et al. [1] present an evolutionary approach that optimizes the euclidean distance metric to improve the performance of a Nearest Neighborhood classifier.

Silva and La Cava et al. proposed a series of wrapper approaches for multi-class classification called MxGP: M2GP [30], M3GP [6, 49, 50, 66] and M4GP [36–38]. M4GP outperforms previous generations of MxGP, as well as highly ranked ML methods, like Multilayer Perceptron and Random Forests, for some multiclass classification problems [37]. The idea of MxGP is to employ GP to perform feature selection/construction in a way that the features become more suitable for a distance-based classifier (e.g., Nearest centroid classifier [22]). MxGP evolves a population of feature transformations calculating the fitness as follows: It creates one cluster per class on the hyper-feature space [6]. The predicted class of each observation is the class of the nearest centroid, according to the *Mahalanobis distance* [15]. The authors of MxGP show that, for this specific problem, such a distance performs much better than other distances (e.g., Euclidean Distance) [30].

Nevertheless, all wrapper approaches employ GP to improve the performance of other (non-GP) classifiers. Differently, DTC employs GP to perform the actual classification and does not rely on other classifiers.

3 DYNAMIC TARGET CLASSIFIER (DTC)

This paper presents DTC, a GP-based classifier for the multi-class classification problem. While previous approaches build functions that discriminate among two or more classes [73], DTC builds functions that discriminate one class each, i.e., each function discriminates one class against all the other classes. In particular, for each class $c_j \in C = \{c_1, c_2, \cdots c_k\}$, DTC considers a "masked" version of the training set, with only two classes c_j and $\overline{c_j} = \{c_w \in C : c_w \neq c_j\}$. DTC learns a set of $k = |C|$ functions (called F). Each function $f \in F$ uses one the masked training sets. Given an instance to classify $\vec{x}$, DTC returns the class c_j that corresponds to the $F[c_j](\vec{x})$ that returns the highest predicted value (given that c_j is set to 1).

Such a masking approach is commonly used by other non-GP classifiers (e.g., SVM), and has the advantage that it does not assume any distribution of the classes in the feature space. Indeed, previous GP classifiers assign an arbitrary value to each class (e.g., $c_1 = 1, c_2 = 2, \cdots c_n = n$) [12, 40, 41]. The choice of this value is purely arbitrary, but induces a total order on the classes that is unlikely to reflect the semantics of the problem. Instead, DTC avoids the issue altogether by considering each class individually. However, this approach introduces an imbalance issue that we address by adjusting the fitness scores with an imbalance ratio [7, 18, 39, 55].

Dynamic targets. DTC follows the *dynamic target approach* [44, 45, 61] that executes multiple GP runs driven by a dynamic "*target*" that changes at each run, based on the residual errors of previous runs. These GP runs produce multiple models, each focusing on the current target. Recent studies show that such an approach can be more effective than evolving a single model [44, 45, 61].

In particular, DTC follows the SGP-DT [61, 62] dynamic target framework, recently proposed by Ruberto et al. for the symbolic regression domain [61]. SGP-DT runs a GP algorithm multiple times to produce a sequence of GP models (called *partial models*), where each model focuses on a particular characteristic of the problem at hand. Then, it generates the *final model* with a linear combination of all the partial models. A key property of SGP-DT is that it does not fix the targets in advance, instead it dynamically discovers them during the GP evolution. At the first iteration, the target is the training instances. At the other iterations, SGP-DT defines the next target as the *residual errors* of the current iteration, i.e., the values predicted by the partial model minus the values of the (current) target. As such, the next iteration will focus on the problem characteristics that the previous iteration did not approximate well.

Evolving a discriminant function. Algorithm 1 describes how DTC evolves a discriminant function for each class $c_j \in C$. Function EVOLVE-DISCRIMINANT-FUNCTION has four inputs: (i) the class in input (c_j), (ii) the training set ($\mathcal{T}$), (iii) the number of external iterations (N_{ext}), which corresponds to the number of partial models, and (iv) the number of internal iterations (N_{int}), which is the number of generations that DTC uses to optimize each partial model. DTC outputs f, the discriminant function for class c_j.

Masking. An initial masking phase (Lines 2 to 10) transforms the training set $\mathcal{T}$ into $\mathcal{T}_{mask}$ by masking all the classes that are different from c_j. More specifically, it replaces the dependent variables $\vec{y}$ with either +1 (if $y_i = c_j$) or -1 (if $y_i \neq c_j$).

However, this likely introduces an imbalance problem [21], with c_j the *minority class* and $\overline{c_j}$ (all classes that are not c_j) the *majority class*. In this scenario, classifiers can have good accuracy on the majority class but very poor accuracy on the minority class [7]. One possible solution for imbalanced data in ML are *sampling* techniques [21] that artificially balance the training set. However, they suffer from overfitting and poor generalization issues [7].

Instead, we use an *imbalance ratio* to amplify the incorrect predictions of the minority class by a factor of the class imbalance [7]. For this reason, we count the number of instances in $\mathcal{T}_{mask}$ that belong to class c_j ($numc_j$). DTC needs this information to compute the imbalance ratio imb at line 10, i.e., the ratio of the instances that do not belong to class c_j ($n - numc_j$) to those that belong ($numc_j$). When computing the fitness function, DTC will rely on this ratio to adjust the hinge-loss scores of the instances belonging to c_j.

Algorithm 1: Dynamic Target Classifier (DTC)

input : c_j: class in input
$\mathcal{T} = \{(\vec{x}_i, y_i), i = 1, 2, \cdots n\}$: training set
N_{ext} : number of external iterations
N_{int} : number of internal iterations
output : f : discriminant function for class c_j

1 **Function** *EVOLVE-DISCRIMINANT-FUNCTION*
2 $\mathcal{T}_{mask} \leftarrow \emptyset$ **masking**
3 $numc_i \leftarrow 0$ // # of instances belonging to class c_j
4 **for** *each* $(\vec{x}_i, y_i) \in \mathcal{T}$ **do**
5 **if** $y_i = c_j$ **then**
6 add $(\vec{x}_i, +1)$ to $\mathcal{T}_{mask}$
7 $numc_j \leftarrow numc_j + 1$
8 **else**
9 add $(\vec{x}_i, -1)$ to $\mathcal{T}_{mask}$
10 $imb \leftarrow \frac{n - numc_j}{numc_j}$ // imbalance coefficient
11 $\vec{t} \leftarrow \vec{y} \in \mathcal{T}_{mask}$ // initialization of the target
12 $partialModels \leftarrow \emptyset$
13 **for** *ext-iter* $1 \ldots N_{ext}$ **do** // external iterations
14 $\mathcal{P} \leftarrow$ GET-RANDOM-INITIAL-POPULATION()
15 **for** *int-iter* $1 \ldots N_{int}$ **do** // internal iterations (GP runs)
16 **for** *each* $\mathcal{I} \in \mathcal{P}$ **do**
17 **fitness computation**
18 $\mathcal{I}_{\text{ls}} \leftarrow$ COMPUTE-LS$(\mathcal{I}, \vec{x}, \vec{t})$ // linear scaling
19 $\vec{\epsilon} \leftarrow \vec{t} - \mathcal{I}_{\text{ls}}(\vec{x})$ // error of the individual
20 $\vec{p} \leftarrow \vec{y} - \vec{\epsilon}$ // prediction
21 $\vec{hl} \leftarrow \ell(\vec{p}) : \max(0, 1 - \vec{t} \cdot \vec{p})$ // hinge-loss
22 $\textbf{fitness}(\mathcal{I}) = \sum_{i=1}^{n} \begin{cases} \vec{hl}_i^{\,2} \cdot imb & \text{if } \vec{y}_i = +1 \\ \vec{hl}_i^{\,2} & \text{otherwise} \end{cases}$
23
24 $\mathcal{I}^{\star}_{ls} \leftarrow$ GET-BEST-INDIVIDUAL$(\mathcal{P})$
25 **if** $fitness(\mathcal{I}^{\star}_{ls}(\vec{x})) = 0$ **then**
26 break loop line 13
27 $\mathcal{P}' \leftarrow \emptyset$
28 add ELITE$(\mathcal{P})$ to $\mathcal{P}'$
29 **while** $\mathcal{P}'$ *is not full* **do**
30 $\mathcal{I} \leftarrow$ TOURNAMENT-SELECTION$(\mathcal{P})$
31 add MUTATE$(\mathcal{I})$ to $\mathcal{P}'$
32 $\mathcal{P} \leftarrow \mathcal{P}'$
33 add $\mathcal{I}^{\star}_{ls}$ to $partialModels$
34 $\vec{t} \leftarrow \vec{t} - \mathcal{I}^{\star}_{ls}(\vec{x})$ // update the new target
35 $f \leftarrow \sum_{model \in partialModels} model$
36 **return** f

DTC then starts the external and internal iterations of the dynamic target approach [61]. It initializes the current target $\vec{t}$ with the dependent variables $\vec{y}$ of the masked training set (line 11), and starts the N_{ext} external iterations. At each external iteration (lines 13-34), DTC instantiates $\mathcal{P}$ (line 14) with a fresh randomly generated population using the *ramped-half-and-half* approach [34]. Then, it starts the N_{int} iterations to evolve $\mathcal{P}$.

3.1 Fitness Computation

DTC computes the fitness scores of the individuals in $\mathcal{P}$ by relying on the *linear scaling* [32] and the *hinge-loss function* [58].

Linear Scaling [32]. Function COMPUTE-LS calculates the linear scaling [32] for each individual $\mathcal{I} \in \mathcal{P}$. Linear scaling adapts instantly an individual to the best possible scale with respect to the current target ($\vec{t}$) [32]. This is particularly useful in our case because by using the hinge-loss function we introduce a boundary (0), and the individuals might not be optimized for such an arbitrary boundary. Note that we could have chosen any arbitrary boundary, we chose 0 following standard practices. In other words, linear scaling transforms the individuals so that their potential fit with the current target is immediately given; we do not need to wait for GP to produce an individual optimized for the boundary [32]. And thus, in a dynamic-target approach linear scaling reduces the number of both external and internal iterations [61]. Fewer iterations result in populations with simpler structural complexity and less computational cost [61].

We compute the linear scaling of an individual $\mathcal{I}$ following the equations of Keijzer [32]:

$$\textbf{Linear scaling} : \mathcal{I}_{ls} = a + b \cdot \mathcal{I} \tag{1}$$

$$\text{where} \quad a = \bar{t} - b \cdot \overline{\mathcal{I}(\vec{x})} \quad \text{and} \tag{2}$$

$$b = \frac{\sum_{i=1}^{n}[(t_i - \bar{t}) \cdot (\mathcal{I}(x_i) - \overline{\mathcal{I}(\vec{x})})]}{\sum_{i=1}^{n}[(\mathcal{I}(x_i) - \overline{\mathcal{I}(\vec{x})})^2]} \tag{3}$$

where n is the number of training cases, and $\bar{t}$ and $\overline{\mathcal{I}(\vec{x})}$ denote the average target and average output value, respectively. The cost of computing the linear scaling coefficients for each external iteration is $O(N_{int} \cdot n \cdot |\mathcal{P}|)$.

Hinge-loss function [58]. The hinge-loss is a loss function commonly used for training classifiers, most notably SVMs. At the best of our knowledge, we are the first to use it in a GP classifier.

Figure 1 depicts the intuition behind the hinge-loss function. The x-axis represents the distance from the boundary (0) of a given instance, and the y-axis represents the loss value (or penalty) that the function returns depending on the distance from the boundary. The function penalizes instances that are incorrectly classified proportionally to the distance from the boundary (<0 the red area in Figure 1). A positive distance from the boundary incurs a low hinge loss, or no hinge loss at all if the prediction is further away from the boundary (>= 1 the green area in Figure 1). If the distance from the boundary is 0 (meaning that the instance is exactly on the boundary), then we incur a loss size of 1.

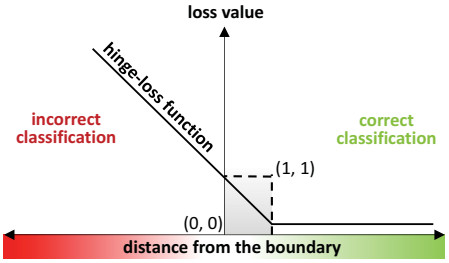

Figure 1: Hinge-loss function

The advantage of the hinge-loss function is that it captures the situation in which the instance is correctly classified but with low margin (the gray area in Figure 1), so that a search algorithm can further improve it. This is the main difference from the commonly used Root Mean Square Error (RMSE) and accuracy-based loss functions. RMSE penalizes instances at the increasing of the distance from the lines of the best fit (+1 and -1 in our case). Accuracy-based loss functions do not distinguish correctly classified instances that are very far (the green area in Figure 1) from or very near (the gray area in Figure 1) from the boundary.

$$\textbf{Hinge-Loss} : \ell(\vec{p}) = \max(0, 1 - \vec{t} \cdot \vec{p}) \tag{4}$$

where $\vec{t} \pm 1$ is the intended output (current target) and the vector $\vec{p}$ are the predicted values.

Lines 19 to 22 compute the fitness score of the individuals using the hinge-loss function. First, DTC computes the error of $\mathcal{I}_{ls}$ with respect to the current target $\vec{t}$, i.e., $\epsilon = \vec{t} - \mathcal{I}_{\text{ls}}(\vec{x})$. Then, DTC computes the prediction $\vec{p}$ as the difference between the actual prediction values ($\vec{y}$) and the residual errors $\vec{\epsilon}$, i.e., $\vec{p} = \vec{y} - \vec{\epsilon}$. Intuitively, because $\vec{t}$ captures the residual errors of all previous external iterations, ϵ represents the error that would be obtained if $\mathcal{I}_{ls}$ is used as the next partial model. And thus, $\vec{p} = \vec{y} - \vec{\epsilon}$ represents the hypothetical prediction of the model (if $\mathcal{I}_{ls}$ is used as the next partial model).

Line 21 computes the hinge-loss values, as described above. Then line 22 computes the fitness score of $\mathcal{I}$ by summing the hinge-loss values for each $i = 1 \cdots n$ as follows: If $y_i = +1$ (i.e, the actual class is c_j) we multiply the hinge-loss value hl_i with the imbalance ratio imb, otherwise we just sum the hinge-loss value hl_i as it is. This is to mitigate the class imbalance of the masked training set. In any case, we consider the square of the hinge-loss to (i) penalize more the high losses, and (ii) create a synergy with the linear scaling that also uses the least square methods [32].

Running example. We exemplify the fitness computation with a running example. Let assume that we have a masked training set $\mathcal{T}_{mask}$ with three instances ($n = 3$), actual prediction values $\vec{y} = \langle +1, -1, -1 \rangle$, and imbalance ratio $imb = 10$.

Let assume that at the first internal iteration DTC computes the fitness of a scaled individual $\mathcal{I}_{ls}^{a}$ that returns $\mathcal{I}_{ls}^{a}(\vec{x}) = \langle -0.7, -0.3, 0.0 \rangle$ when evaluated on the training set. Lines 19 and 20 compute the error $\vec{\epsilon}$ and prediction $\vec{p}$. In this case $\vec{p} = \mathcal{I}_{ls}^{a}(\vec{x})$ because at the first iteration there are not previous errors to approximate (i.e, $\vec{t} = \vec{y}$, line 11). The hinge-loss $\ell(\vec{p})$ is the following:

[*hinge-loss for i* = 1]: the actual value of this instance is +1 and the predicted value is -0.7, meaning that the point is on the wrong side of the boundary, thus incurring a large hinge loss of hl_1 = 1 - (+1 · -0.7) = +1.7 (because max{+1.7, 0} = +1.7).

[*hinge-loss for i* = 2]: the actual value of this instance is -1 and the predicted value is -0.3, which is smaller than 0 but greater than -1. The predicted class would be correct but the value is still too near to the boundary and get a moderate penalty of hl_2 = 1 - (-1 · -0.3) = +0.7 (because max{+0.7, 0} = +0.7).

[*hinge-loss for i* = 3]: the actual value of this instance is -1 and the predicted value is 0, which means that the point is on the boundary, thus incurring a cost of hl_3 = +1.

The fitness of $\mathcal{I}_{ls}^1$ is $[(1.7^2 \cdot (imb = 10)) + (0.7^2) + (1^2)]$ = 28.9 + 0.49 + 1 = 30.39.

After N_{int} internal iterations, before starting a new external iteration, DTC updates the new target $\vec{t}$ based on the residuals errors of the current target and the best model $\mathcal{I}_{ls}^\star$ (line 34). Let assume that the best model $\mathcal{I}_{ls}^\star$ is the individual discussed above ($\mathcal{I}_{ls}^a$). Thus, the new target is $\vec{t} = \vec{y} - \mathcal{I}_{ls}^1(\vec{x}) = \langle +1.7, -0.7, -1\rangle$.

At the second iteration, let us assume that DTC computes the fitness of a scaled individual $\mathcal{I}_{ls}^b$ that returns $\mathcal{I}_{ls}^b(\vec{x}) = \langle +1.2, -0.3, -1.0\rangle$ when evaluated on the training set. This time at line 20 we need to consider also the previous errors (the target $\vec{t}$) in order to get the prediction $\vec{p}$ and calculate the hinge loss.

The difference between the target $\vec{t}$ and the partial model $\mathcal{I}_{ls}^b(\vec{x})$ gives the error with respect to the training set $\vec{\epsilon} = \langle +0.5, -0.4, 0\rangle$. The values predicted by $\mathcal{I}_{ls}^b$ on the training set are $\vec{p} = \vec{y} - \vec{\epsilon} = \langle +1, -1, -1\rangle - \langle +0.5, -0.4, 0\rangle = \langle +0.5, -0.6, 0\rangle$. The corresponding hinge loss will be $\vec{hl} = \langle +0.5, +0.4, 0\rangle$. Then, DTC computes the fitness score of $\mathcal{I}_{ls}^b$ as explained above.

3.2 Evolution

After DTC computes all fitness scores for each individual in the current population $\mathcal{P}$, it gets the individual in $\mathcal{P}$ with the best (i.e., lowest) fitness score ($\mathcal{I}_{ls}^\star$). DTC then adds it to the list of partial models. It then checks if its fitness score is zero. If so, then the external loop at line 13 prematurely ends and DTC combines all partial models in a single discriminant function *f*. Otherwise (the common case), DTC evolves $\mathcal{P}$ into $\mathcal{P}'$.

Following the dynamic-target framework [61], DTC uses a variant of the classical GP algorithm that does not use any form of crossover. Ruberto et al. show that such a variant is effective for the dynamic-target approach [61]. More specifically, DTC adopts the standard elite operator (line 28), uses the tournament selection [57] and the canonical mutation operators for tree-like individuals [61]. The tournament selection selects one individual $\mathcal{I}_{ls} \in \mathcal{P}$ (line 30), mutates it with a certain probability (line 31) and adds it to $\mathcal{P}'$.

After DTC completes all the N_{ext} external iterations, it returns the discriminant function f as a linear combination of all the *partial models* produced during the external iterations (using the linear scaling coefficient already computed at line 18). That is the sum of the partial models with their *linear scaling* [32] coefficients. Intuitively, by combining all partial models we are summing all the estimates of the residuals, and thus obtaining a function f that well approximates the training set [60].

Algorithm 2: The DTC classifier

```
input  : T = {(x_i, y_i), i = 1, 2, ··· n} : training set
         C = {c_1, c_2, ··· c_k} : classes to predict
         N_ext : number of external iterations
         N_int : number of internal iterations
output : F: k discriminant functions for each class in C
37 Function TRAIN-CLASSIFIER
38     F[] ← ∅
39     for each class c_j ∈ C do
40         F[c_j] ← EVOLVE-DISCRIMINANT-FUNCTION(c_j, T, N_ext, N_int)
41     return F

input  : x: new instance to classify
         C = {c_1, c_2, ··· c_k} : classes to predict
         F: k discriminant functions for each class in C
output : predicted_class: predicted class of x (c ∈ C)
42 Function GP-CLASSIFIER
43     predicted_value[] ← ∅
44     for each class c_j ∈ C do
45         predicted_value[c_j] ← F[c_j](x)
46     predicted_class ← argmax_{c_j ∈ C} {predicted_value[c_j]}
47     return predicted_class
```

3.3 Classification

Algorithm 2 describes how DTC creates the discriminant functions F (TRAIN-CLASSIFIER) and classifies new instances (GP-CLASSIFIER).

Function **TRAIN-CLASSIFIER** has four inputs: (i) the training set ($\mathcal{T}$), (ii) the set of classes to predict $C = \{c_1, c_2, \cdots c_k\}$, (iii) the number of external iterations (N_{ext}), and (iv) the number of internal iterations (N_{int}). It constructs the set of discriminant functions by invoking Function EVOLVE-DISCRIMINANT-FUNCTION described in Algorithm 1 for each class $c_j \in C$.

Function **GP-CLASSIFIER** has three inputs: (i) the new instance to classify ($\vec{x}$), (ii) the set of possible classes $C = \{c_1, c_2, \cdots c_k\}$, (iii) the set of discriminant functions (F) returned by Function TRAIN-CLASSIFIER. It returns the set of discriminant functions by invoking Function TRAIN-CLASSIFIER. To classify a new instance $\vec{x}$, DTC computes the *predicted_value* of every discriminant function in $F[c_j](\vec{x})$, where $j = 1 \cdots k$ (line 45). DTC returns the predicted class as the one that corresponds to the function with the highest predicted value (line 46). Ideally, only one function in the set returns a value greater than zero. However, if multiple functions in F return values greater than zero, DTC considers the highest one. This is because functions with higher predicted values are likely to have more confidence in their predictions.

4 EXPERIMENT

We now describe an experiment that we conducted to evaluate the classification accuracy of DTC on eight popular datasets. Moreover, we compare DTC with eight state-of-the-art classifiers.

4.1 Datasets

Table 1 shows the characteristics of the eight datasets that we consider in our experiments. The number of classes ranges from 2 (*heart*) and 15 (*movl*), and the number of instances from 270 (*heart*) to 5,000 (*wav*). These are popular datasets for the multi-class classification problem. Six of these datasets are from the UCI data

Table 1: The characteristics of the eight datasets used in our experiment

	heart	im-3	wav	segm	im-10	yeast	vowel	movl
# classes	2	3	3	7	10	10	11	15
# instances	270	322	5,000	2,310	6,798	1,484	990	360
# attributes	13	6	40	19	6	8	13	90

repository [17], and the other two, *im-3* and *im-10*, are satellite datasets from the United States Geological Survey (USGS) [71]. We choose them because they are the datasets used by LaCava and Silva to evaluate M2GP, M3GP, and M4GP.

Heart Statlog [17] (**heart**) has 270 instances and 13 attributes that includes age, sex, cholesterol, type of pain, electrocardiogram characteristics. This classification problem has two classes representing the presence or absence of a heart disease.

Waveform_40 [9, 17] (**wav**) has three classes of waves generated by a combination of 2 or 3 "base" waves. It includes 5,000 instances generated by adding noise (mean 0, variance 1) in each of the 40 continuous attributes.

Image Segmentation [17] (**segm**) has 2,310 instances with 19 attributes. The attributes represent the characteristics of a hand-segmented region of 3x3 pixels randomly chosen from seven outdoor images (i.e., the seven classes to predict).

Yeast [17] (**yeast**) comprises 1,484 instances. Each instance is a set of eight binary attributes, representing yeast protein traits. The 10 classes to predict are the possible 10 cellular localization sites of these proteins. This dataset is very imbalanced [17].

Vowel [72] (**vowel**) includes 11 different vowel classes of the Japanese language. Each of these 11 vowels is uttered 6 times by 15 different speakers for a total of 990 instances [72].

Movement Libras [16] (**movl**) has 15 classes of 24 instances each (360 instances in total), represented as 90 continuous attributes. Each class represents a type of hand gesture. This problem has the highest number of classes (15) among the eight considered datasets.

The datasets **im-10** and **im-3** consist of satellite images from the USGS [71]. The dataset *im-10* includes 6,798 images, while *im-3* 322 images, which belong to ten and three classes, respectively.

4.2 Classifiers

We compare our approach with eight different classifiers. We consider four standard non-GP classifiers: Random Forest (**RF**) [26], Random Subspace (**RS**) [27]. Multi Layer Perceptron (**MLP**) [63], and Support Vector Machines (**SVM**) [13]. We also consider the following four state-of-the-art *wrapper approaches*, which employ GP to select/construct features for a *nearest centroid classifier* [22].

M2GP [30] uses multiple trees to transform the original features in a new n-dimensional space (where n is decided in advance). M2GP computes the class centroids and classifies instances according to the nearest centroid. M2GP and all of its successor employ the covariance matrix and the Mahalanobis distance to compute the distance between instances and centroids [30].

Table 2: Configuration parameters of DTC

Parameter description	value	Parameter description	value
population size ($\|\mathcal{P}\|$)	100	size elitism	1
prob. of mutation	100%	depth of tree (initialization)	[1;4]
prob. mutation leaf level	70%	max. depth of tree (mutation)	5
Ephemeral Random Constants (ERC)	[-1;1]	max. depth of tree (all phases)	15
tournament size	2	internal iterations (N_{int})	50

M3GP [66] addresses the limitation of M2GP that needs to know in advance the number of dimensions (n). M3GP proposes genetic operators to dynamically explore the number of dimensions to optimize the classification.

eM3GP [66] is a variant of M3GP that uses ensembles. In particular, it identifies the best feature transformations and uses different strategies to build ensembles of such transformations.

M4GP [36–38] follows the same approach as M3GP with the main difference of using a stack-based representation of individuals (while both M2GP and M3GP use tree-like individuals). Such a representation gives the advantage that an individual returns multiple outputs, and thus reduces the exploration cost [37]. There are three variants of M4GP, each adopting a different parent selection criterion: (i) tournament selection [57], (ii) lexicase [25, 68], and (iii) Age-Fitness Pareto survival (AFP) [65], which exploits an archive of best individuals. For a fair and meaningful comparison, we consider the variant of M4GP that uses tournament selection, because it is the same selection criterion used by DTC.

We chose to compare with wrapper approaches because they are known to drastically outperform previous GP classifiers that evolve discriminant functions [37, 66]. Indeed, several researchers reported poor classification accuracy for GP classifiers, especially with classification problems with more than two classes [11, 35, 73].

4.3 Experimental Setup

Table 2 shows the configuration parameter values of DTC, which are the same used by the dynamic target approach SGP-DT [61]. Note that the probability of mutation is 100% because DTC does not use the crossover operator. While we kept the number of internal iterations fixed for all experiments ($N_{\text{int}} = 50$), for the number of external iterations (N_{ext}) we explored four different values ("low": 20, "medium": 40, "high": 80, "very high": 160) on some sample data. We noticed that DTC needs a "high" or "very high" N_{ext} with datasets with a high number of instances or classes. *yeast* is the only dataset that does not follow this trend (because it is very imbalanced). As such, we choose the following values of N_{ext} and we kept them the same for every trial: "low" (*yeast*, *heart*), "medium" (*movl*, *im-3*), "high" (*wav*, *vowel*), and "very high" (*segm*, *im-10*).

Table 3: Median classification accuracy (%) of the 30 trials on the TEST set (statistical significance is marked in *underlined bold*)

dataset	RF	RS	MLP	SVM	M2GP	M3GP	eM3GP	M4GP	DTC
heart	80.25	81.48	80.25	55.56	80.25	79.01	80.86	**87.65**	81.48
im-3	94.85	92.78	95.88	93.81	93.81	95.36	8.45	**97.94**	93.81
wav	81.50	82.20	83.33	**86.30**	84.87	84.33	81.23	**86.03**	**85.90**
segm	**97.26**	95.96	96.32	55.84	95.60	95.61	94.73	95.09	96.54
im-10	**96.86**	93.92	90.22	90.36	90.19	90.96	90.31	89.56	**93.46**
yeast	**57.53**	**56.63**	**57.98**	41.12	53.82	56.19	56.19	56.84	**59.35**
vowel	89.39	82.83	82.49	81.82	85.86	93.77	78.62	**95.96**	92.09
movl	71.76	65.74	75.93	14.35	62.96	57.07	65.15	**76.85**	**76.39**

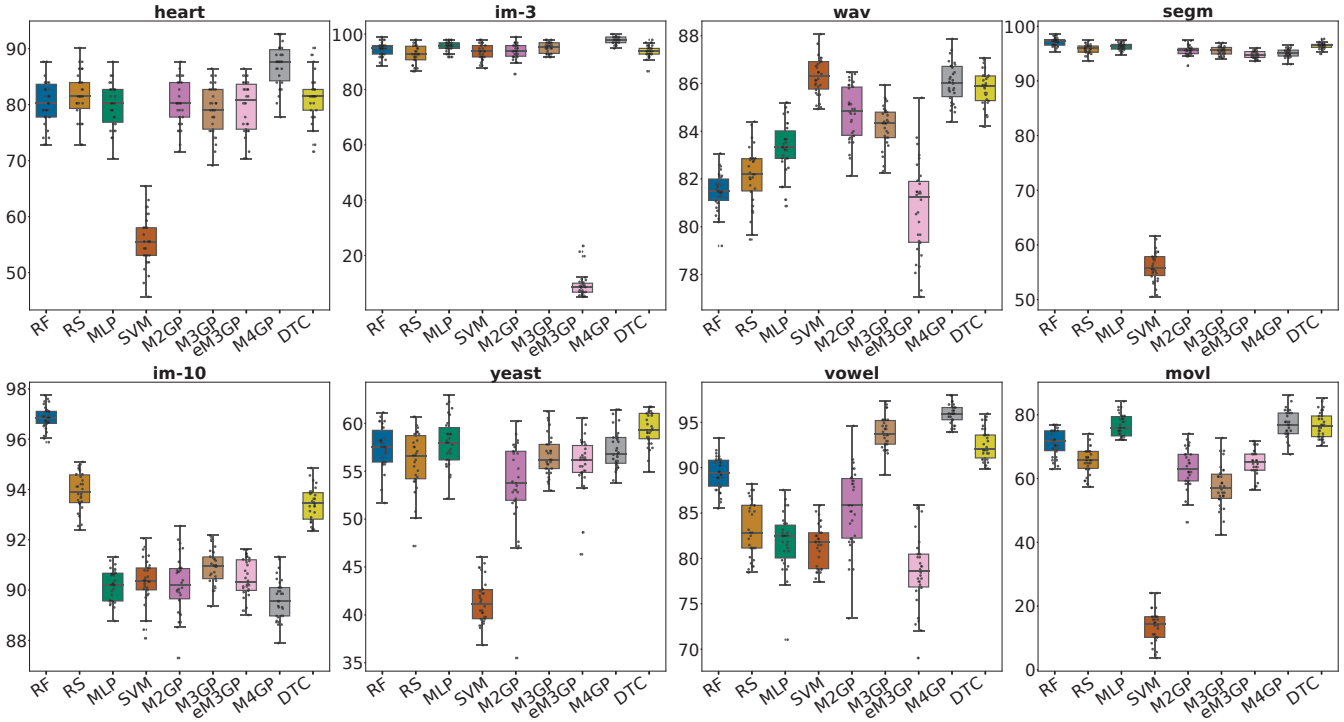

Figure 2: Distributions of the classification accuracy (%) of the 30 trials on the TEST set.

We contacted La Cava and Silva (the authors of MxGP), which provided us the datasets and the classification accuracy results of the eight classifiers (see Section 4.2). They also helped us to replicate as much as possible their experimental setup, so that we could run DTC on the same setup. Each classifier is run for 30 trials, and for each trial, the dataset is randomly partitioned into 70% training and 30% testing. The performance of the classifiers is evaluated with the median classification accuracy % of the 30 trials ($\frac{\#\ correct\ pred.}{\#\ total\ pred.} \cdot 100$).

To check for statistical significance when comparing the accuracy of DTC and the other eight classifiers, we performed the following statistical tests. First, we performed a *Friedman test* [22] that indicates significant differences between methods across all datasets. Then, we computed the p-values with the post-hoc analysis based on the non-parametric pairwise *Wilcoxon rank-sum test* [24] ($\alpha = 0.05$).

5 RESULTS

Table 3 shows, for each dataset and classifier, the median classification accuracy % (on the test set) of the 30 trails, For each dataset (row in the table), the medians in **underlined bold** represent the distributions that are statistically significantly better than the ones not in bold. If a dataset has multiple medians in underlined bold, it means that there is no statistical difference among them. Figure 2 shows the box plots of the distributions of the classification accuracy of the 30 trials. The results show that DTC achieves competitive results on all datasets. In particular, we highlight four interesting findings.

I. DTC has high accuracy even with many classes (≥ 7)**.** The datasets with a higher number of classes (≥ 7) are *segm*, *im-10*, *yeast*, *vowel*, and *movl*. If for each of these datasets we sort the classifiers based on median accuracy, DTC is always in the TOP 3 classifiers. This is an unprecedented result for GP classifiers based on discriminant functions [11, 35, 66, 73].

Table 4: Median classification accuracy (%) of the 30 trials on the TRAINING set

dataset	RF	RS	MLP	SVM	M2GP	M3GP	eM3GP	M4GP	DTC
heart	98.41	88.89	98.41	100.00	89.42	94.71	86.77	93.65	99.47
im-3	100.00	97.11	98.67	100.00	98.22	99.56	93.30	99.11	100.00
wav	99.47	92.01	98.49	100.00	87.40	90.69	81.86	88.81	91.77
segm	99.88	98.42	97.56	100.00	96.82	98.08	96.16	95.86	99.81
im-10	99.81	96.30	91.05	100.00	91.44	92.96	92.00	90.46	96.02
yeast	98.27	71.08	64.58	100.00	62.56	68.49	61.06	59.68	65.89
vowel	99.86	97.76	91.92	100.00	95.89	100.00	87.88	100.00	100.00
movl	99.21	92.26	91.27	100.00	100.00	100.00	100.00	100.00	100.00

II. DTC outperforms off-the-shelf classifiers for most of the datasets. DTC has a higher median classification accuracy than Random Forest (RF) for seven datasets (except *im-3*), Random Subspace (RS) for six datasets (except *im-10* and *heart*), Multi Layer Perception (MLP) for seven datasets (except *im-3*), and Support Vector Machines (SVM) for six datasets (except *im-3* and *wav*). Notably, SVM and DTC both use the hinge-loss function and the one-versus-all classification strategy.

III. DTC outperforms early versions of MxGP. When considering the median classification accuracy, DTC outperforms M2-GP and M3-GP for seven datasets (except *im-3*) and eM3-GP for all datasets.

IV. DTC and M4GP achieve similar and complementary performance. DTC outperforms M4GP for three datasets (*segm*, *im-10*, *yeast*) with statistical significance in two of them. In the other five datasets, they both achieve comparable accuracy. Interestingly, DTC and M4GP exhibit complementary performance. Table 3 shows that DTC and M4GP are the classifiers that collectively achieve the best results overall (see the underlined bold medians).

Remarkably, La Cava recently experimented with a variant of M4GP, called M4GP-float [37], that evolves discriminant functions to directly classify instances (without relying on the *nearest centroid classifier*). The results of La Cava show that the original M4GP with tournament selection drastically outperforms M4GP-float, especially on the datasets with seven or more classes [37]. This is also true if we compare DTC with M4GP-float. For example, for the datasets *segm*, *vowel*, and *movl* ($|C| \geq 7$), DTC has a median accuracy on the test set more than double that the one of M4GP-float [37].

6 DISCUSSION AND CONCLUSION

In this paper, we presented DTC, a GP classifier for the multi-class classification problem. At the best of our knowledge, DTC is the first attempt to combine the synergy of the GP dynamic target approach, hinge-loss function, and linear scaling. Our results on eight popular datasets show that DTC achieves a classification accuracy that is competitive with – and sometimes even better than – popular ML approaches (e.g., RF, RS, SVM, and MLP) and the state-of-the-art wrapper approaches for GP feature selection/construction (e.g., M2GP, M3GP, eM3GP, and M4GP). Notably, differently from DTC, wrapper approaches employ GP to improve the performance of other (non-GP) classifiers. DTC employs GP to perform the actual classification and does not rely on other classifiers.

The promising results of DTC spark interesting future work. Above all, there are three research directions that could further improve the effectiveness of DTC.

Parameter tuning. Table 4 shows the median classification accuracy % of the eight classifiers on the training set. In some cases, a high classification accuracy of DTC on the training set (Table 4) corresponds to a significantly lower classification accuracy on the test set (Table 3). For instance, on the *movl* dataset the median classification accuracy of DTC is 100% on the training set and 76.39% on the test set. Even if 76.39% is the best result on the test set (together with the one of M4GP), there might still be room for improvement. In fact, this situation suggests over-fitting. A systematic exploration of the parameters of DTC could help better understand this issue and further improve the classification accuracy on the test set.

Alternative parent selection. Another interesting future work is to study alternative parent selection techniques (DTC currently uses tournament selection). In particular, the selection technique *lexicase* [68] could work in synergy with the dynamic target approach. In fact, the dynamic target approach focuses on some characteristics of the problem at every external iteration, while *lexicase* could focus on a different portion of the training cases at each internal iteration. This would further promote the division of the problem in smaller, hopefully, easier parts.

Reduce the computational effort. Population-based approaches are generally computationally expensive. Indeed, DTC performs many GP runs for learning one discriminant function, and needs to learn one discriminant function for each class. To give an idea of the typical resource consumed by DTC, we report the computational cost for the *yeast* dataset, which has ten classes ($|C| = 10$). The median size of a partial model is 81 nodes (recall that DTC evolves tree-like individuals). DTC computed 20 external iterations ($N_{ext} = 20$) for *yeast*, and thus it computed 1,640 nodes to evolve ten discriminant functions. According to the results reported by La Cava et al. [37], the cost of M4GP is on the same order of magnitude.

However, preliminary experiments show that by drastically reducing the depth of the trees (15 in our experiments) the computational cost decreases significantly, while the classification accuracy slightly decreases. Investigating the trade-off between classification accuracy and computational cost is an important future work. Also, one could both reduce the computational cost and improve the performance of DTC by changing the way it represents individuals: from trees to more compact or expressive structures (such as, the stack-based representation used by M4GP [37]).

REFERENCES

[1] Alexandros Agapitos, Michael O'Neill, and Anthony Brabazon. 2013. Adaptive distance metrics for nearest neighbour classification based on genetic programming. In *European Conference on Genetic Programming*. Springer, 1–12.

[2] Mohamed Aly. 2005. Survey on multiclass classification methods. *Neural Netw* 19 (2005), 1–9.

[3] Alejandro Baldominos, Yago Saez, and Pedro Isasi. 2018. Evolutionary Convolutional Neural Networks: An Application to Handwriting Recognition. *Neurocomputing* 283 (2018), 38–52. https://doi.org/10.1016/j.neucom.2017.12.049

[4] Alejandro Baldominos, Yago Saez, and Pedro Isasi. 2018. Model Selection in Committees of Evolved Convolutional Neural Networks Using Genetic Algorithms. In *Intelligent Data Engineering and Automated Learning (IDEAL '18)*. 364–373. https://doi.org/10.1007/978-3-030-03493-1_39

[5] Alejandro Baldominos, Yago Saez, and Pedro Isasi. 2019. Hybridizing Evolutionary Computation and Deep Neural Networks: An Approach to Handwriting Recognition Using Committees and Transfer Learning. *Complexity* (2019). https://doi.org/10.1155/2019/2952304

[6] João E Batista and Sara Silva. 2020. Improving the Detection of Burnt Areas in Remote Sensing using Hyper-features Evolved by M3GP. In *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[7] U. Bhowan, M. Johnston, and M. Zhang. 2012. Developing New Fitness Functions in Genetic Programming for Classification With Unbalanced Data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42, 2 (April 2012), 406–421. https://doi.org/10.1109/TSMCB.2011.2167144

[8] Erik Bochinski, Tobias Senst, and Thomas Sikora. 2017. Hyper-Parameter Optimization for Convolutional Neural Network Committees Based on Evolutionary Algorithms. In *Proceedings International Conference on Image Processing (ICIP '17)*. 3924–3928. https://doi.org/10.1109/ICIP.2017.8297018

[9] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.

[10] James Butterworth, Rahul Savani, and Karl Tuyls. 2019. Evolving Indoor Navigational Strategies using Gated Recurrent Units in NEAT. In *Proceedings of the Companion of Genetic and Evolutionary Computation Conference (GECCO '19)*. 111–112. https://doi.org/10.1145/3319619.3321995

[11] Mauro Castelli, Sara Silva, Leonardo Vanneschi, Ana Cabral, Maria J. Vasconcelos, Luís Catarino, and João M. B. Carreiras. 2013. Land Cover/Land Use Multiclass Classification Using GP with Geometric Semantic Operators. In *Applications of Evolutionary Computation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 334–343.

[12] Zheng Chen and Siwei Lu. 2007. A genetic programming approach for classification of textures based on wavelet analysis. In *2007 IEEE International Symposium on Intelligent Signal Processing*. IEEE, 1–6.

[13] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. In *Machine Learning*. 273–297.

[14] Joe Davison. 2020. DEvol: Automated Deep Neural Network Design via Genetic Programming. https://github.com/joeddav/devol

[15] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. 2000. The mahalanobis distance. *Chemometrics and intelligent laboratory systems* 50, 1 (2000), 1–18.

[16] D. B. Dias, R. C. B. Madeo, T. Rocha, H. H. Biscaro, and S. M. Peres. 2009. Hand movement recognition for Brazilian Sign Language: A study using distance-based neural networks. In *2009 International Joint Conference on Neural Networks*. 697–704. https://doi.org/10.1109/IJCNN.2009.5178917 ISSN: 2161-4407.

[17] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[18] Charles Elkan. 2001. The Foundations of Cost-Sensitive Learning. In *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*. 973–978.

[19] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *J. Mach. Learn. Res.* 20 (2019), 55:1–55:21.

[20] Pedro G. Espejo, Sebastián Ventura, and Francisco Herrera. 2009. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40, 2 (2009), 121–144.

[21] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. 2004. A multiple resampling method for learning from imbalanced data sets. *Computational intelligence* 20, 1 (2004), 18–36.

[22] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. 2001. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York.

[23] Edgar Galván and Peter Mooney. 2021. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence* (2021).

[24] Winston Haynes. 2013. Wilcoxon Rank Sum Test. *Encyclopedia of systems biology* (2013), 2354–2355.

[25] Thomas Helmuth, Lee Spector, and James Matheson. 2015. Solving Uncompromising Problems With Lexicase Selection. *IEEE Trans. Evol. Comput.* 19, 5 (2015), 630–643. https://doi.org/10.1109/TEVC.2014.2362729

[26] Tin Kam Ho. 1995. Random decision forests. In *Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I*. IEEE Computer Society, 278–282. https://doi.org/10.1109/ICDAR.1995.598994

[27] Tin Kam Ho. 1998. The Random Subspace Method for Constructing Decision Forests. *IEEE Trans. Pattern Anal. Mach. Intell.* 20, 8 (1998), 832–844. https://doi.org/10.1109/34.709601

[28] John J Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* 79, 8 (1982), 2554–2558.

[29] Tom Howley and Michael G. Madden. 2005. The genetic kernel support vector machine: Description and evaluation. *Artificial intelligence review* 24, 3-4 (2005), 379–395.

[30] Vijay Ingalalli, Sara Silva, Mauro Castelli, and Leonardo Vanneschi. 2014. A multi-dimensional genetic programming approach for multi-class classification problems. In *European Conference on Genetic Programming*. Springer, 48–60.

[31] Emiliano Carreño Jara, Guillermo Leguizamón, and Neal Wagner. 2007. Evolution of classification rules for comprehensible knowledge discovery. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 September 2007, Singapore*. IEEE, 1261–1268. https://doi.org/10.1109/CEC.2007.4424615

[32] Maarten Keijzer. 2003. Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. In *Proceedings of the European Conference on Genetic Programming (EuroGP '03)*. 70–82. https://doi.org/10.1007/3-540-36599-0_7

[33] John R Koza. 1990. Concept formation and decision tree induction using the genetic programming paradigm. In *International Conference on Parallel Problem Solving from Nature*. Springer, 124–128.

[34] John R. Koza. 1993. *Genetic programming - on the programming of computers by means of natural selection*. MIT Press.

[35] Krzysztof Krawiec. [n.d.]. Genetic Programming Based Construction of Features for Machine Learning and Knowledge Discovery Tasks. ([n. d.]).

[36] William La Cava, Sara Silva, Kourosh Danai, Lee Spector, Leonardo Vanneschi, and Jason H Moore. 2018. A multidimensional genetic programming approach for identifying epsistatic gene interactions. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 23–24.

[37] William La Cava, Sara Silva, Kourosh Danai, Lee Spector, Leonardo Vanneschi, and Jason H. Moore. 2019. Multidimensional genetic programming for multiclass classification. *Swarm and evolutionary computation* 44 (2019), 260–272.

[38] William La Cava, Sara Silva, Leonardo Vanneschi, Lee Spector, and Jason Moore. 2017. Genetic programming representations for multi-dimensional feature learning in biomedical classification. In *European Conference on the Applications of Evolutionary Computation*. Springer, 158–173.

[39] Jin Li, Xiaoli Li, and Xin Yao. 2005. Cost-sensitive classification with genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*. IEEE, 2114–2121. https://doi.org/10.1109/CEC.2005.1554956

[40] Jung-Yi Lin, Hao-Ren Ke, Been-Chian Chien, and Wei-Pang Yang. 2007. Designing a classifier by a layered multi-population genetic programming approach. *Pattern Recognition* 40, 8 (2007), 2211–2225.

[41] Jung-Yi Lin, Hao-Ren Ke, Been-Chian Chien, and Wei-Pang Yang. 2008. Classifier design with feature selection and feature extraction using layered genetic programming. *Expert Systems with Applications* 34, 2 (2008), 1384–1393.

[42] Uriel López, Leonardo Trujillo, Yuliana Martinez, Pierrick Legrand, Enrique Naredo, and Sara Silva. 2017. RANSAC-GP: Dealing with Outliers in Symbolic Regression with Genetic Programming. In *Proceedings of the European Conference on Genetic Programming (EuroGP '17)*. 114–130.

[43] Tyler McDonnell, Sari Andoni, Elmira Bonab, Sheila Cheng, Jun-Hwan Choi, Jimmie Goode, Keith Moore, Gavin Sellers, and Jacob Schrum. 2018. Divide and conquer: neuroevolution for multiclass classification. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 474–481.

[44] David Medernach, Jeannie Fitzgerald, R. Muhammad Atif Azad, and Conor Ryan. 2015. Wave: A Genetic Programming Approach to Divide and Conquer. In *Proceedings of the Companion of Genetic and Evolutionary Computation Conference* (Madrid, Spain) *(GECCO '15)*. 1435–1436. https://doi.org/10.1145/2739482.2764659

[45] David Medernach, Jeannie Fitzgerald, R. Muhammad Atif Azad, and Conor Ryan. 2016. A New Wave: A Dynamic Approach to Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '16)*. 757–764. https://doi.org/10.1145/2908812.2908857

[46] Roberto R. F. Mendes, Fabricio de B. Voznika, Alex Alves Freitas, and Júlio C. Nievola. 2001. In *Principles of Data Mining and Knowledge Discovery, 5th European Conference, PKDD 2001, Freiburg, Germany, September 3-5, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2168)*. Springer, 314–325. https://doi.org/10.1007/3-540-44794-6_26

[47] Ícaro Marcelino Miranda, Claus Aranha, and Marcelo Ladeira. 2019. Classification of EEG signals using genetic programming for feature construction. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1275–1283.

[48] Mohammed Ahmed Muharram and George D. Smith. 2005. Evolutionary Constructive Induction. *IEEE Trans. Knowl. Data Eng.* 17, 11 (2005), 1518–1528. https://doi.org/10.1109/TKDE.2005.182

[49] Luis Muñoz, Sara Silva, and Leonardo Trujillo. 2015. M3gp–multiclass classification with gp. In *European Conference on Genetic Programming*. Springer, 78–91.
[50] Luis Muñoz, Leonardo Trujillo, Sara Silva, Mauro Castelli, and Leonardo Vanneschi. 2019. Evolving multidimensional transformations for symbolic regression with M3GP. *Memetic Computing* 11, 2 (2019), 111–126.
[51] Kourosh Neshatian and Mengjie Zhang. 2008. Genetic Programming and Class-Wise Orthogonal Transformation for Dimension Reduction in Classification Problems. In *Genetic Programming, 11th European Conference, EuroGP 2008, Naples, Italy, March 26-28, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4971)*. Springer, 242–253. https://doi.org/10.1007/978-3-540-78671-9_21
[52] Patryk Orzechowski, William La Cava, and Jason H. Moore. 2018. Where Are We Now?: A Large Benchmark Study of Recent Symbolic Regression Methods. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. 1183–1190. https://doi.org/10.1145/3205455.3205539
[53] Evgenia Papavasileiou and Bart Jansen. 2017. An Investigation of Topological Choices in FS-NEAT and FD-NEAT on XOR-Based Problems of Increased Complexity. In *Proceedings of the Companion of Genetic and Evolutionary Computation Conference (GECCO '17)*. 1431–1434. https://doi.org/10.1145/3067695.3082497
[54] Topon Kumar Paul and Hitoshi Iba. 2006. Classification of scleroderma and normal biopsy data and identification of possible biomarkers of the disease. In *2006 IEEE Symposium on Computational Intelligence and Bioinformatics and Computational Biology*. IEEE, 1–6.
[55] Wenbin Pei, Bing Xue, Lin Shang, and Mengjie Zhang. 2019. Reuse of program trees in genetic programming with a new fitness function in high-dimensional unbalanced classification. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. ACM, 187–188. https://doi.org/10.1145/3319619.3321958
[56] Yiming Peng, Gang Chen, Harman Singh, and Mengjie Zhang. 2018. NEAT for Large-Scale Reinforcement Learning Through Evolutionary Feature Learning and Policy Gradient Search. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. 490–497. https://doi.org/10.1145/3205455.3205536
[57] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming*. lulu.com. http://www.gp-field-guide.org.uk/
[58] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. 2004. Are Loss Functions All the Same? *Neural Comput.* 16, 5 (2004), 1063–107. https://doi.org/10.1162/089976604773135104
[59] Stefano Ruberto, Valerio Terragni, and Jason H. Moore. 2020. Image feature learning with a genetic programming autoencoder. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. ACM. https://doi.org/10.1145/3377929.3389981
[60] Stefano Ruberto, Valerio Terragni, and Jason H. Moore. 2020. Image Feature Learning with Genetic Programming. In *Parallel Problem Solving from Nature – PPSN XVI (Lecture Notes in Computer Science)*. Springer International Publishing, Cham, 63–78. https://doi.org/10.1007/978-3-030-58115-2_5
[61] Stefano Ruberto, Valerio Terragni, and Jason H. Moore. 2020. SGP-DT: Semantic Genetic Programming Based on Dynamic Targets. In *Genetic Programming (Lecture Notes in Computer Science)*. Springer International Publishing, Cham, 167–183. https://doi.org/10.1007/978-3-030-44094-7_11
[62] Stefano Ruberto, Valerio Terragni, and Jason H. Moore. 2020. SGP-DT: Towards Effective Symbolic Regression with a Semantic GP Approach Based on Dynamic Targets. In *Proceedings of the Genetic and Evolutionary Computation Conference (Hot Off the Press track), GECCO 2020, Cancun, Mexico, July 8-12, 2020*. 25–26. https://doi.org/10.1145/3377929.3397486
[63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation*. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.
[64] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117. https://doi.org/10.1016/j.neunet.2014.09.003
[65] Michael D. Schmidt and Hod Lipson. 2010. Age-fitness pareto optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010*. ACM, 543–544. https://doi.org/10.1145/1830483.1830584
[66] Sara Silva, Luis Munoz, Leonardo Trujillo, Vijay Ingalalli, Mauro Castelli, and Leonardo Vanneschi. 2016. Multiclass classification through multidimensional clustering. In *Genetic Programming Theory and Practice XIII*. Springer, 219–239.
[67] Léo FDP Sotto, Regina C. Coelho, and Vinícius V. de Melo. 2016. Classification of cardiac arrhythmia by random forests with features constructed by kaizen programming with linear genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. 813–820.
[68] Lee Spector. 2012. Assessment of problem modality by differential performance of lexicase selection in genetic programming: a preliminary report. In *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings*. ACM, 401–408. https://doi.org/10.1145/2330784.2330846
[69] Keith M. Sullivan and Sean Luke. 2007. Evolving kernels for support vector machine classification. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. 1702–1707.
[70] Toru Tanigawa and Qiangfu Zhao. 2000. A Study on Efficient Generation of Decision Trees Using Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00), Las Vegas, Nevada, USA, July 8-12, 2000*. Morgan Kaufmann, 1047–1052.
[71] USGS. [n.d.]. U.S. geological survey (USGS) earth resources observation systems (EROS) data center (EDC). http://glovis.usgs.gov/
[72] Xuechuan Wang and Kuldip K. Paliwal. 2003. Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition. *Pattern Recognition* 36, 10 (Oct. 2003), 2429–2439. https://doi.org/10.1016/S0031-3203(03)00044-X
[73] Mengjie Zhang and William Smart. 2004. Multiclass Object Classification Using Genetic Programming. In *Workshops on Applications of Evolutionary Computation*. 369–378. https://doi.org/10.1007/978-3-540-24653-4_38