NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Layered Genetic Programming for Feature Extraction in Classification Problems

Justina Padolskaitė

Dissertation presented as the partial requirement for obtaining a Master's degree in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

LAYERED GENETIC PROGRAMMING FOR FEATURE EXTRACTION IN CLASSIFICATION PROBLEMS

by

Justina Padolskaitė

Dissertation presented as the partial requirement for obtaining a Master's degree in Data Science and Advanced Analytics

Advisor: Leonardo Vanneschi

Co-Advisor: Illya Bakurov

August 2020

ABSTRACT

Genetic programming has been proven to be a successful technique for feature extraction in various applications. In this thesis, we present a Layered Genetic Programming system which implements genetic programming-based feature extraction mechanism. The proposed system uses a layered structure where instead of evolving just one population of individuals, several populations are evolved sequentially. Each such population transforms the input data received from the previous population into a lower dimensional space with the aim of improving classification performance.

The performance of the proposed system was experimentally tested on 5 real-world problems using different dimensionality reduction step sizes and different classifiers. The proposed method was able to outperform a simple classifier applied directly on the original data on two problems. On the remaining problems, the classifier performed better using the original data. The best solutions were often obtained in the first few layers which implied that increasing the size of the system, i.e. adding more layers was not useful. However, the layered structure allowed control of the size of individuals.

KEYWORDS

Genetic Programming; Feature Extraction; Dimensionality Reduction; Classification

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

GP	**Genetic Programming**
LGP	**Layered Genetic Programming System**
DT	**Decision Tree**
KNN	**K-Nearest Neighbors**
LR	**Logistic Regression**
GA	**Genetic Algorithm**
ADFs	**Automatically Defined Functions**
SVM	**Support Vector Machine**
MLP	**Multi-Layer Perceptron**

1. INTRODUCTION

Many real-world problems can be modeled as classification problems, which make the classification perhaps one of the most widely studied problems in the data mining and machine learning communities, with numerous applications that cover different domains, such as medical diagnosis [1, 2, 3, 4, 5], e-mail spam detection [6, 7], fraud detection [8, 9], and customer churn prediction [10, 11], to name a few. The set of input features plays an important role for the success of a good classification model. In many classification tasks, the representation of the original feature set is rarely optimal and thus can be improved. Very often, data contain irrelevant or redundant features, and the features have complex interactions between them which should be taken into consideration. Some systems have an intrinsic part of the learning process which attempts to select relevant features or construct new possibly more insightful features based on the original ones. For instance, decision trees (DTs) use feature selection in every step when constructing the tree [12]; hidden neurons of artificial neural networks implicitly create new features [13]. Feature selection and extraction methods are often applied to enhance the quality of the feature set, reduce the dimensionality and thus to improve the performance of the algorithm in terms of learning time and predictive accuracy [14].

While feature selection methods aim to choose a subset of most relevant and discriminant original features [13], the goal of feature extraction is to map original features to a new feature space with improved separability of different classes [15]. There are various techniques that have been used for feature extraction over the years that range from the principal component analysis [16] and support vector machines (SVM) [17] to genetic algorithms (GAs) [18]. Indeed, the application of genetic programming (GP) in order to construct/extract new features has become increasingly popular due to the capability of GP to automatically build mathematical expressions based on some objective function. To-date, GP has been used in different scenarios showing promising results.

In this thesis, we contribute to the study of the application of GP for feature extraction. The novelty of this thesis stems from the implementation of a layered structure. That is, in each layer a separate GP population is evolved with the aim of mapping the original feature set to a new feature space with reduced dimensionality, thereby making the classification task easier and also improving the performance of a classifier. The rationale for stacking the GP populations into several layers is based on the architecture of the deep artificial neural networks where only the first layer receives as input the original data and in all other layers, the input is the output of the previous layer. The primary question of this work is whether GP can benefit from the layered structure.

This thesis is organized as follows. Section 2 provides a theoretical background necessary for this work. That is, Section 2 introduces the reader to the machine learning concepts, presents the basic aspects of the GP as well as the previous work on the application of GP to feature extraction. Section 3 introduces the proposed system by describing its design and functioning in more detail. Section 4 presents the experimental study conducted to evaluate the performance of the proposed system. Finally, the concluding remarks and future research are presented in Section 5.

2. THEORETICAL BACKGROUND

2.1. MACHINE LEARNING

Machine learning is the subfield of Artificial Intelligence which studies the algorithms that automatically improve with experience, i.e. *learn* [19]. Data are at the core of machine learning, as they serve as examples used for learning. When the machine learning algorithm is provided with the training data, it outputs a trained *model*. Having created a predictive algorithm, it will generate a predictive model which, when a new data instance is given as input to it, outputs a prediction based on the data that was used for training. This basic machine learning process is illustrated in Figure 1.

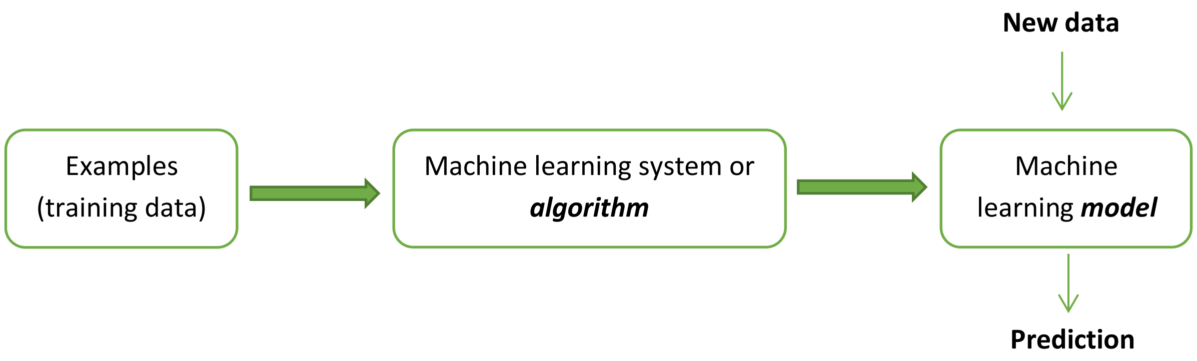
New data
Examples (training data)
Machine learning system or **algorithm**
Machine learning **model**
Prediction

Figure 1. Machine learning process.

Machine learning techniques are typically classified into three broad categories: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. In supervised learning, the training data are labelled, i.e. there are known target values, and the aim is to learn a mapping function from the input to the output. In unsupervised learning, unlabeled data are used, i.e. desired output is unknown, and the aim is to discover the hidden structures and relationships in these data. In reinforcement learning, the aim is to maximize a reward signal. It involves exploration of an adaptive sequence of actions in a given environment with the motivation to maximize the cumulative reward. In addition, a type of learning, which has some unknown target values, and thus is between supervised and unsupervised learning is called semi-supervised learning.

In supervised learning, the problem can either be a regression problem or a classification problem. With respect to regression problems, the output variable is a real value. In the classification problems, the output variable is a category, and the goal is to find a model that accurately predicts a class label for a given instance of input data.

2.2. Genetic programming

In the early 1990s, GP [20] became a popular search technique. GP is the subfield of evolutionary algorithms which applies the principles of Darwin's theory of evolution and natural selection [21] to search for the fittest computer program (*individual*) in the space of all possible computer programs (*individuals*). The individuals reproduce, adapt, and compete amongst themselves.

GP starts with a randomly initialized population of individuals. Consequently, GP evaluates each individual's performance using a fitness function and assigns a fitness value to each of them. Based on the fitness values, it then chooses some of the individuals and produces a new population for the next generation from these chosen individuals by means of genetic operators. The search repeats until an optimal or acceptable solution is found, or a certain other stopping criterion is met. A general GP algorithm can be summarized by the following pseudo-code:

```
1. Generate an initial population of random individuals, P.
2. Evaluate the fitness of each individual in the population P.
3. Until an acceptable solution is found or some other termination criterion (e.g., the prefixed
   maximum number of generations has been executed) is met, repeat:
     a. Create an empty population, P'.
     b. Until the number of individuals in P' is equal to the number of individuals in P, repeat:
          i. With a probability based on fitness, select one or two individual(s) from P.
         ii. With specified probabilities, create new individual(s) by applying genetic
             operators.
        iii. Insert the individuals created in the previous step into P'.
     c. Evaluate the individuals in P'.
     d. Replace P with P'.
4. Return the individual with best fitness.
```

In the following subsections, representation of individuals, initialization of GP population, fitness, selection and genetic operators are explained in more detail.

2.2.1. Representation of individuals

In the GP population, an *individual* is a computer *program* that is a solution to a problem that is intended to be solved. The way GP program is encoded varies among different GP systems. Based on the representation of individual, some categories of GP can be mentioned such as linear GP [22], where

GP programs are linear sequences of instructions, Cartesian GP [23], in which a program is represented as an indexed graph, encoded in the form of a linear string of integers, and grammar-based GP [24, 25, 26] which uses a context-free grammar to define the initial GP structures. However, perhaps the most common representation is a tree structure, as shown in Figure 2, and it was used in this work.

Based on the example given in Figure 2, variables (X_1, X_2, X_3) and constant (4) in the program are leaves of the tree, which in GP are called *terminals*, while the arithmetic operations (+, -, /) are internal nodes called *functions*. *Terminal set* and *function set* define a set of elements which are available to GP to create computer programs and are defined on the problem domain. Commonly, the terminal set consists of the program's external inputs (variables), functions with no arguments and constants, while the function set includes arithmetic operations, mathematical functions, Boolean operations, conditional operators, iterative operations and any other domain-specific functions that may be defined [22].

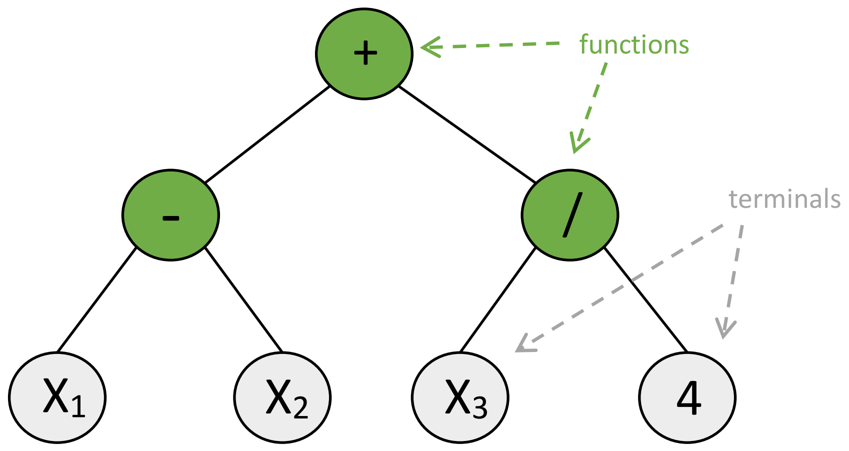

Figure 2. Example of a tree-based representation of a GP individual.

In the tree structure, each node has a depth value associated to it which is the number of edges from the tree's root node to that node. The depth of a tree is the depth of its deepest leaf.

2.2.2. Population initialization

The initialization of the population consists of the creation of the programs that will later be evolved. The individuals in the initial population are typically randomly generated. Three earliest methods described by Koza [20] are *Full*, *Grow*, and their combination known as *Ramped Half-and-Half*. In all these methods, the initial individuals do not exceed the maximum depth specified by the user.

Full method. While the maximum depth is not reached, nodes are taken at random from the function set. Once the maximum depth is reached, only terminals are randomly chosen. This way, all the leaves of a generated tree are at the same depth and the depth of a tree is always equal to the predefined maximum depth.

Grow method. Nodes at depths less than the predefined maximum are randomly chosen from the combined set consisting of the union of the function set and the terminal set. Once a branch contains a terminal node, that branch is terminated. The random selection of nodes at the maximum depth is restricted to the terminal set. Since the branch may be terminated before the specified maximum depth has been reached, trees created using *Grow* method are likely to have irregular shapes.

Ramped Half-and-Half method. Let *d* be the predefined maximum tree depth. The population is divided evenly into *d* groups, and, for each such group, a distinct maximum depth value is set from a range between 1 and *d*. Then, for each depth group, half of the individuals in the group are created using the *Full* method and another half of the individuals are created using the *Grow* method. This initialization method creates trees having various sizes and shapes and, this way, enhances the diversity of initial population. For this reason, it is the most commonly used initialization method.

2.2.3. Fitness evaluation

In order to determine how well a GP individual performs, i.e. how well-suited an individual is to solve a given problem, the individual needs to be evaluated. A function to evaluate individuals is called *fitness function,* and the result of the evaluation (a numeric value) is *fitness*. The fitness function plays an important role in guiding GP to obtain the best solutions within a large search space, and hence, its design is critical. A good fitness function guarantees a more effective and efficient exploration of search space while an inappropriate fitness function can make GP trapped in a local optimum [27].

2.2.4. Selection

The selection methods are used to select individuals to which genetic operators will be applied. A key property of selection mechanism is *selection pressure*, which is defined as the degree to which better performing individuals are favored [28]. Good genetic material in the chosen parents is expected to be propagated along evolution in order to speed up population convergence. A method with high selection pressure highly favors the individuals having better fitness while a method with a weak selection pressure is less discriminating.

The most commonly used method to select individuals in GP is *tournament selection* [22], which is based on competition within a subset of the population. A number of individuals are randomly sampled with replacement from the current population into a tournament, and then the one which has the best fitness from the tournament is chosen to be a parent for the next generation. By using different tournament sizes, the selection pressure can be adjusted: the larger the tournament size, the higher the selection pressure.

2.2.5. Genetic operators

Genetic operators are applied to the individuals selected in the selection step to produce offspring for the next generation. The *crossover* operator generates new offspring by combining the genetic information of two existing individuals. Given two individuals as parents, the *standard GP crossover* [20], also known as *swap crossover*, begins by selecting one random point (a node) in each parent. The selected point is called *crossover point* for that parent. Two offspring individuals are then produced by exchanging the subtrees rooted at the crossover points, as illustrated in Figure 3.

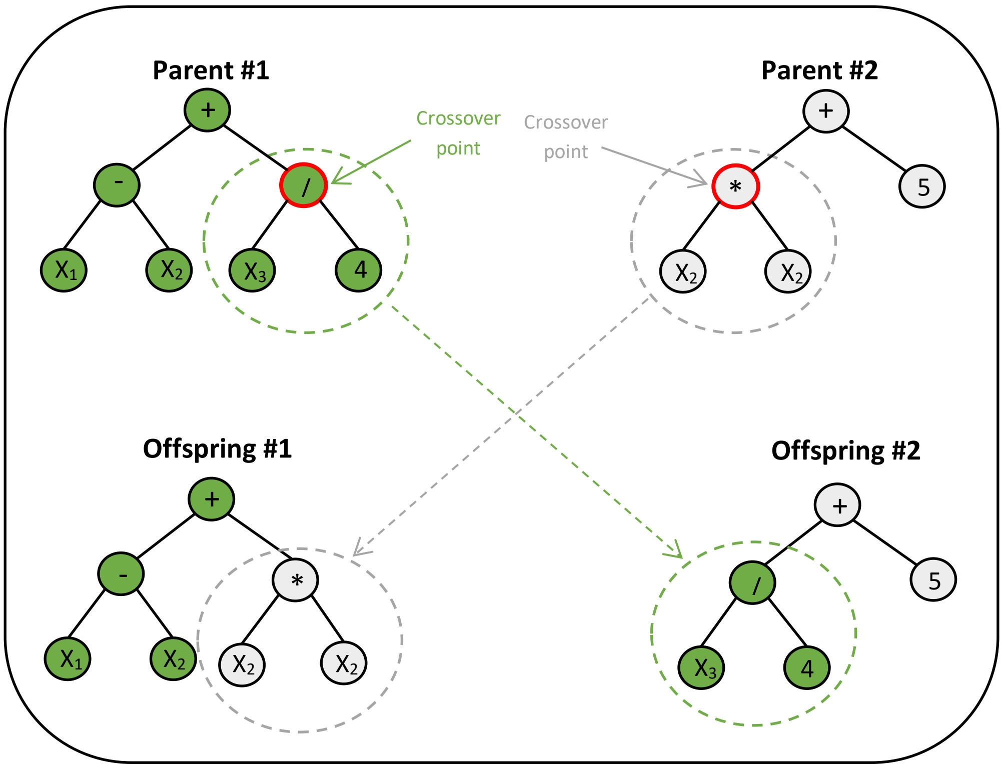

Figure 3. An example of standard crossover.

Mutation operator operates on only one parental program. The *standard GP mutation* [20], called *subtree mutation*, begins by randomly selecting a point, called *mutation point*, within the selected individual and then substitutes the subtree rooted there with a randomly generated subtree as illustrated in Figure 4. This operator allows to better explore potentially new genetic information, i.e. areas of the search-space, as it introduces previously unseen structures in the individuals.

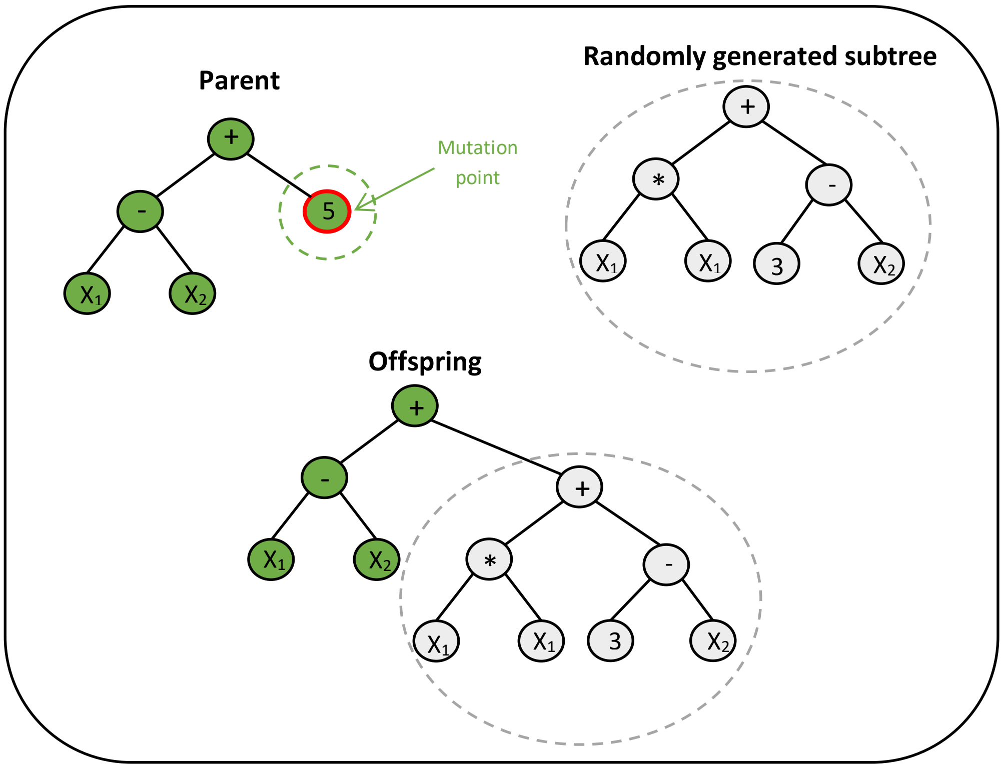

Figure 4. An example of subtree mutation.

The genetic operators are applied with a given probability, and in GP usually they are mutually exclusive. When the rates of crossover and mutation (probability of applying crossover and probability of applying mutation) add up to a value which is less than 1, a *reproduction* operator is used, which simply inserts a copy of a selected individual into the new population.

2.3. Genetic programming for feature extraction

Due to its capability to automatically generate solutions and detect the underlying relationship that exists in the data as well as its flexibility, GP is a good choice for generating features. It has been widely applied to different scenarios showing promising results. Considering the GP's interaction with the final learner, two major categories can be noted based on previous research. In one category, which can be seen as a wrapper approach, the final learner is incorporated in the evolutionary process and its performance on generated features determines the fitness of GP individuals; in another category, feature construction or extraction is done as a preprocessing phase, and no particular classifier is involved in the evaluation of the new features, expecting to obtain more general results. However, regarding the latter approach, a problem independent indicator is needed to determine the appropriateness of generated features, thus the design of a fitness function might be more challenging [29]. Below, we briefly describe the main aspects of other researchers' work.

Raymer et al. [30] applied GP to modify the original feature space to improve the performance of K-nearest neighbors (KNN) classifier. In their approach, for problems with *n* features individuals consisted of *n* automatically defined functions (ADFs), where each ADF was evolved for a particular feature using that feature and zero or more constants. Fitness of individuals was measured based on the classifier performance on extracted features. The system was applied to a biochemistry problem and showed slightly better results compared to a similar system with genetic algorithm.

Similarly to [30], in [31], Bot proposed a GP framework for automatic feature extraction to improve the accuracy of KNN classifier on the new features, however, it also aimed to reduce the dimensionality of the dataset. Proposed approach was a greedy algorithm, where new features were added one-at-a-time, and only if the relative increase in the accuracy on training set was greater than the predefined constant. The system was applied to 16 different datasets, where the number of features varied from 4 to 60. It was shown that the system was able to reduce the dimensionality of most datasets to one or two features while improving or at least not worsening the classification performance of KNN classifier.

Sherrah [32] created a feature extraction system based on GP where the individuals were multi-trees, each of which encoded one new feature, and fitness was the estimated misclassification rate of a particular classifier trained on the pre-processed data. A set of classifiers used by the system included minimum distance to means classifier, parallelepiped classifier, and Gaussian maximum likelihood classifier. According to Sherrah, the classifiers used by the system should not be too powerful, otherwise they would do all the classification work by themselves and there would be no pressure for the system. The experiments carried out using synthetic and real-world datasets from different domains, such as medical diagnosis, social sciences and image processing, showed that although the simple classifiers performed poorly on their own, the system was able to improve their classification performance by evolving appropriate features. For the real-world problems, however, the proposed method did not result in a significantly better classification performance than the multi-layer perceptron (MLP). Overall, the effectiveness of the system broke down on high-dimensional problems.

Kotani et al. [33] proposed a feature extraction method using GP to improve the classification accuracy in the pattern recognition tasks. They assumed that extracted features were the polynomial expressions of the original features and searched for them using GP, they then fed the generated features into a KNN classifier to evaluate the classification performance and evaluate the fitness of individuals. Experiments performed on two artificial tasks and the acoustic diagnosis for compressors as a real-world task confirmed the effectiveness of the proposed method.

Krawiec [34] used GP for changing the representation of the input data with the objective to improve the performance of a particular classifier, namely C4.5 DT. Under the standard approach, individuals in the evolving population were encoded as a fixed-length vector of expression trees and during the evolutionary search random modifications by means of genetic operators were applied to all features. However, the preliminary experiments of Krawiec showed that while some parts of the solution (some features) were improved due to genotype changes, at the same time some other parts might have undergone deteriorating modifications. Because of that, in addition to the general framework for GP-based feature extraction, an extended approach, in which useful features were preserved during an evolutionary process, was proposed. Under the extended approach, the genotype of each individual was split into two disjoint parts, where the former part was a subject to evolutionary search, and the latter part was treated as a repository where most valuable features of the individual were stored. This way, two evaluation mechanisms were implemented: one that was used to evaluate an individual as a whole (the whole feature set) and another one that was used to evaluate a particular feature. The utility of a feature was evaluated using a scalar utility measure based on feature usage statistics in the DT construction. Performed experiments showed that for all considered problems (3 real-world and 3 artificial problems), one of the two presented approaches provided better classification accuracy than the one obtained on original dataset; the solution obtained under the extended approach was better than the solution obtained under the standard approach at least for two problems. However, the results did not allow to state that the proposed extended method outperformed the approach under which the raw input data were used in a systematic and statistically significant manner. Moreover, overfitting was noticed in many of the experiments.

In [35], Bhanu and Krawiec proposed another version of standard approach to use GP for feature extraction presented in [34], which was based on the paradigm of cooperative coevolution. In [34], features were evolved autonomously and then co-adapted to the remaining part of the derived dataset. More precisely, each species in cooperative coevolution algorithm was responsible for developing one feature for the derived dataset, and each GP individual representing a particular species implemented a single feature. In this way, the entire solution was obtained by *n* cooperating GP individuals, where *n* is the dimensionality of the derived dataset. The experimental results showed that, in terms of classification accuracy of C4.5 DT obtained on the derived dataset, features found by means of coevolutionary search were better than features extracted under the standard approach. The obtained results also indicated the presence of overfitting. However, the experiments were performed only on one benchmark dataset and it is not clear how the presented method would work on different problems.

Guo et al. [36] applied GP-based feature extraction method to bearing conditions monitoring task. Instead of using the classification results to determine the fitness value of GP individuals, they adapted the Fisher criterion based on maximization of between-class scatter over the within-class scatter and sought to maximize the degree of difference between the classes, this way reducing the computational demands. Experimental results by Guo et al. demonstrated that artificial neural networks and SVM were able to obtain better classification accuracy when features extracted by GP were used, compared with the classification accuracy obtained using features generated by other classical feature extraction methods.

Similarly to [36], in [37], Guo and Nandi proposed to evaluate fitness based on the Fisher criterion, but to overcome the drawback of the Fisher criterion when it is used as a measure of class separation (it can result in a large value not only due to well-separated clusters but also due to the overlapping classes with small variances), they developed a modified Fisher criterion where the within-class scatter was based on the distance instead of the variance between any two patterns belonging to the same class. The experiments carried out with a simple minimum distance classifier on breast cancer diagnosis problem demonstrated that a proposed method could significantly reduce the dimensionality required to describe the problem and made the classification effective in one-dimensional feature space, improving the classification performance and outperforming more sophisticated classifiers like MLP and SVM used with a set of all original features.

Firpi et al. [38] applied a general-purpose algorithm consisting of GP module and KNN classifier to epileptic seizure prediction task. There GP generated artificial features directly from the reconstructed state-space trajectory of the EEG signals, attempting to find the best discrimination between the non-seizure data and pre-seizure data by minimizing the error-risk objective function. The extracted features were then fed into a KNN classifier. Performed experiments showed that features generated by GP matched or exceeded the performance of traditional conventional features.

Similarly to [30], [31], [33] and [38], in the GP-based feature extraction system proposed by Guo et al. in [15], a GP module was used in conjunction with a KNN classifier. The novel part of the system was a new primitive function introduced in a function set that determined the number of extracted features automatically during GP evolution, with no need to predefine that number beforehand. More precisely, a new feature was obtained using the subtree under that function node, and in this way, the number of times that function appeared in the tree structure determined the number of generated new features. The experiments carried out on the epileptic EEG classification problems showed that proposed method was successful in increasing the classification accuracy and significantly reducing the dimensionality.

Neshatian et al. [29] used GP for feature extraction in classification tasks aiming to improve the classification performance as well as reduce the dimensionality and learn a smaller DT. The number of generated new features was equal to the number of classes in a dataset and each of those features was created by a separate GP run. Rather than using the classification performance as fitness, they designed GP fitness function based on class dispersion and entropy. The approach was examined on 12 benchmark classification problems and results showed that this approach outperformed the standard way of using DTs on original features in terms of classification performance, dimensionality and learned DT size. Moreover, compared with classification using only new features, the classification results using a combined feature set (new features and original features) were worse for most of the datasets.

Smith and Bull [39] examined the use of GP and GA to improve the classification performance of, initially, C4.5 DT. In the proposed system, GP was used to construct new features from the original data and GA was applied to find the most predictive ones which then were fed to the final classifier. There GP individuals consisted of multiple trees, where the number of trees was equal to the number of numeric attributes in the dataset. However, the minimum number of compound trees was introduced to ensure that, for datasets with a small number of features, there would be a sufficient number of new features as subsequent feature selection was performed using a combined set of GP-based features and original features. Fitness of individuals was based on the classification performance of the same classification method as used to create a final classifier, evaluated on constructed new features. Experimental results showed that the proposed hybrid system was able to improve the classification accuracy of C4.5 DT on most of the datasets.

In [40], Otero et al. used GP to construct a single generally-useful feature out of original real-valued attributes which then was appended to the set of original features for use by C4.5 algorithm. Under their approach, the feature construction was independent from the classification algorithm. The information gain ratio was used as fitness function, making the evaluation of individuals relatively more efficient compared to the evaluation of individuals when fitness is based on the classification performance, since it did not require the execution of classification algorithm. However, such fitness criterion is only applicable to a single feature and could not be considered for a set of features [39]. Experiments performed on 4 public-domain datasets showed that GP-based feature was useful for 2 datasets as the error rate of C4.5 DT decreased when such feature was used, but for other 2 datasets there was no significant difference between the error rate of the classifier applied to a set of original features with and without the GP-based feature.

Afzali et al. [41] applied GP to salient object detection. In the proposed method, GP was used to automatically select and combine complementary saliency features to produce the final saliency map. The goodness of each individual was evaluated using fitness function based on the Kullback-Leibler divergence. The experimental results proved the ability of the proposed method to tackle a wide range of saliency features from different segmentation levels and effectively choose and combine them. The GP-based method either significantly outperformed or achieved a comparable performance to the other methods on benchmark datasets.

Bi et al. [42] proposed a multi-layer tree GP approach to feature extraction and image classification which could benefit from the prior designed image-related operators and descriptors. They designed a new tree-based GP program structure composed of 5 layers to achieve automatic region detection, high-level feature extraction, feature construction, and binary classification simultaneously. A new terminal set including 4 types of terminals representing the input image as well as a new function set, including image operators and region descriptors to detect more informative high-level features, were proposed. The experiments carried out on 6 different image datasets of varying difficulty showed that proposed method was able to achieve either a significantly better or comparable classification performance compared with the baseline methods (5 other GP methods for image classification and 42 non-GP methods based on 7 commonly used classification algorithms and 6 image feature extraction methods).

In [43], Tran et al. investigated the use of GP for feature construction and selection on high-dimensional classification problems. A GP individual was represented using tree-based structure; however, such individual not only generated a new high-level feature, but also worked as a binary classifier whose balanced accuracy was used as a fitness measure to guide the search. A single-tree generated by best individual of GP run was proposed to be used to create 6 different feature sets, namely 1) a single constructed feature; 2) a combined feature set of the original features and a constructed feature; 3) a set composed of original features used in terminal nodes; 4) a combined set of 1) and 3); a set containing new features created using all possible subtrees of the tree obtained by best GP individual; 5) a combined set of 3) and 5). The sets *1)*, *2)*, *3)*, and *5)* had been used by other researchers, for instance *1)* can be found in [44], *2)* in [40], [44], and sets *3)* and *5)* were used in [45]; however, sets *4)* and *6)* were proposed for the first time. The performed experiments showed that GP could choose informative features and construct new features that had better discriminatory ability than original features.

Instead of executing multiple GP runs or using multi-tree representation of individual to construct multiple features, Ahmed et al. [45] proposed to evolve a single tree and generate new features using all subtrees of that tree. Moreover, they proposed a fitness function which combined the Fisher criterion and *p*-value, where the Fisher criterion worked by maximizing the between-class scatter and minimizing the within-class scatter while the *p*-value ensured the separation between the different classes is significantly large. The proposed method was examined on a number of mass spectrometry datasets using 7 different classification methods and showed good performance results in terms of dimensionality reduction, classification performance and biomarker identification.

In [46], Tan et al. used GP to evolve composite operators which generated feature vectors from the original orientation field in fingerprint classification problem. The primitive operators were separated into computation and feature generation operators, and feature vector that represented a fingerprint image and was used for classification was formed by features computed whenever feature generation operators were used in the evolved binary tree structure. The experimental results showed that GP could try unconventional ways of combining primitive image processing operations and find good composite operators to extract useful features. Compared with the results of other previously published research, the proposed approach was efficient and promising.

Huertas et al. [47] proposed to use GP for automatic feature extraction for cloud classification. Under the proposed approach, GP was used to evolve a function that aimed to transform an image pixel by pixel, then the mean and standard deviation of the transformed image was computed and used as input features for a linear SVM classifier. The performance of proposed method was tested on whole-sky cloud images and compared with the results obtained using a set of expert-defined features proposed by Heinle et al. [48], which are widely used in the cloud classification problems. The experimental results showed that the proposed method was able to achieve a similar classification accuracy to the 4 most important expert-defined features, but it was unable to reach the highest accuracy obtained using 12 standard features.

In [49], Aslam et al. proposed a three-stage method, composed of feature selection, GP-based feature extraction and classification, for diabetes classification problem. In this method, different subsets of original features were formed based on the order of feature importance obtained by averaging the results of several feature selection methods, and for each such subset GP was trained generating one new feature, which was then used as input for a classifier. The performed experiments showed that GP-based features helped KNN and SVM classifiers to achieve a significant improvement in performance compared to the performance achieved using original diabetes features.

To summarize, the aforementioned studies indicate that GP can be successfully used for extracting new and potentially more discriminant features.

3. LAYERED GENETIC PROGRAMMING SYSTEM

3.1. SYSTEM DESIGN

In this work, we present a Layered Genetic Programming system (LGP) which consists of two parts, namely GP, which is used to transform the input dataset into a new dataset with a lower dimensionality, and a simple classifier that is trained on the transformed dataset during the GP training stage and is used to perform the final classification. Given the input dataset *D*, the objective of the GP population is to transform this dataset into a new dataset *D’* that has lower dimensionality, in such a way that using the transformed dataset *D’* classification performance would be improved. In fact, the work of the GP population can be interpreted as feature extraction as it derives new features from the original ones.

As it has already been discussed in the theoretical part in 2.3, GP has been widely used for feature extraction. However, there are several aspects that make the system design proposed in this study different than in previous work. To be more specific, the current system employs a layered architecture as shown in Figure 5, where in each such layer a GP population is evolved and used to transform a dataset outputted by another GP population in the previous layer. Using this type of structure, only the GP population in the first layer receives as input the original dataset that characterizes the problem. Moreover, in each subsequent layer, the dimensionality of the dataset is gradually reduced. As a result, the final classification is performed using fewer new features than the number of original features.

In the following subsections, the main aspects of system design and functioning, namely representation of GP individuals, the method used to initialize the GP population in each layer, fitness evaluation of the GP individuals, and the application of genetic operators are described in more detail.

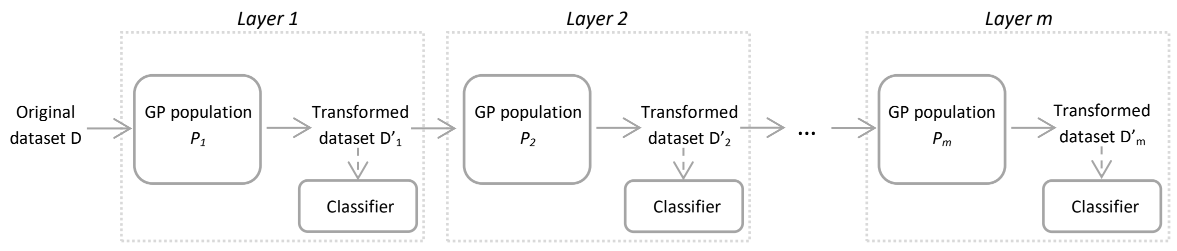

Figure 5. System structure.

3.1.1. Representation of individuals

The output vector of a traditional GP individual (a single tree) can be considered as one new feature derived from the original features. However, when transforming the input dataset, not only do the features themselves matter but also their synergy. In order to allow two scales of search – a search over the features themselves and a search over combinations of features – an individual consists of multiple trees, like in [34, 39]. Thus, each GP individual is *n*-dimensional vector of trees, where each tree is responsible for creating one distinct feature in the new dataset. When all trees are evaluated on the input dataset, a complete dataset characterized by *n* new features is obtained. An example showing a representation of a GP individual consisting of 4 trees that can be used to create a new dataset consisting of 4 features is depicted in Figure 6.

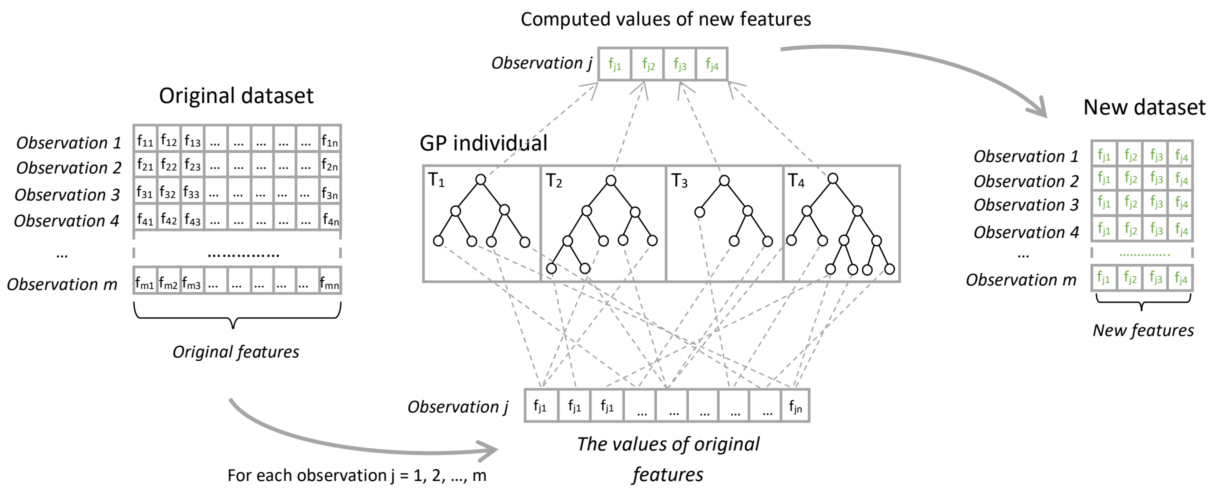

Figure 6. An example of a GP individual composed of multiple trees.

3.1.2. Population initialization

In each layer, an initial GP population consists of randomly generated GP individuals. As each tree in the GP individual is responsible for creating one new feature, it is important to ensure the diversity not only between the individuals in the population but also within each individual, i.e. between the trees that the individual is composed of. If the trees of the GP individual output the same semantics, such an individual is not useful. To address this issue and ensure enough diversity in the initial trees in each layer, an initialization method, which creates each tree in the GP individual using a different input feature subset, is used. The way how the feature subsets are generated depends on the dimensionality of the input dataset in a particular layer. If the dimensionality is relatively large, it is reasonable to use disjoint feature subsets to increase the possibility of each input feature to be used in the initial trees at least once. However, if the number of features in the input dataset is not that large, the full feature set can be used.

The implemented initialization method starts by performing a simple random sampling without replacement to generate a distinct subset of input features for each tree in the individual. If the number of features in the full feature set divided by the number of the required feature subsets (number of trees in the individual) is greater than 2, disjoint subsets of the size equal to this calculated number are created. Otherwise, the subsets with intersection of the size equal to the greater value out of 30% of the full feature set size and the number of features in the input dataset divided by the number of required feature subsets are created if the input dataset does not have less than 10 features, or each of the trees is created using the full feature set if there are less than 10 features in the input dataset. Once the feature subset for a particular tree is formed, the tree is created using *full* or *grow* method, where each of these two methods has an equal probability of being used. The depth for the tree is randomly chosen from the predefined range.

The initialization of GP population can be summarized as follows:

```
1. Define feature subset size ds for disjoint subsets.
2. For each tree i = 1, 2, …, n in the GP individual:
   2.1. If ds > 2 then
            Create disjoint feature subset S_i of size ds.
        Else if the number of features in the full feature set is greater or equal to 10 then
            Create feature subsets with intersection where subset size is equal to max(a, b), where
            a = 0.3 * feature set size, b = feature set size / number of feature subsets.
        Else
            Use the full feature set.
   2.2. Select a method to create the tree from {full, grow}.
   2.3. Select the tree depth from the predefined range.
   2.4. Create the tree.
```

3.1.3. Evaluation of individuals

Different measures to evaluate the fitness of GP individuals that return as output a new feature or feature set have been proposed in the literature and tested experimentally, such as classification accuracy [34, 31, 35], misclassification value [15], information gain ratio [40], the Fisher criterion [49, 37], entropy [29]. If a classification performance metric is used, such as classification accuracy or misclassification rate, the utility of the entire new feature set is evaluated, however, it requires a classifier to be trained using new features every time an individual needs to be evaluated, which makes the process computationally demanding.

Assuming that time is not a constraint in the training process, we use a fitness function which is based on the classifier's performance on the dataset outputted by a GP individual. More precisely, fitness of a GP individual is the F1 score computed on the transformed dataset returned as output by that individual. F1 score is chosen because the classification accuracy may be a misleading metric in case of imbalanced datasets. F1 score combines *precision* (what proportion of positive identifications was actually correct) and *recall* (what proportion of actual *positives* was identified correctly) into one metric giving equal relative contribution to both of them [50]. The range of F1 score is [0; 1], where the greater the value, the better is the performance of a model. Thus, this is a maximization problem.

The fitness evaluation procedure for an $individual_i$ is as follows:

1. Given training data, use $individual_i$ to create a new training set.
2. Train a classifier on the training set obtained in step *1)*.
3. Predict the class of each instance in the training set obtained in step *1)* using the trained classifier obtained in step *2)*.
4. Compute F1 score and assign this value as fitness of $individual_i$.

To obtain fitness on test set, the same procedure is repeated using the test data, except the step 2) which is then omitted.

3.1.4. Classification

The classification method applied on the output of GP individual is either DT, or KNN, or logistic regression (LR). These three widely used methods perform conceptually different classification, are relatively simple and easy to implement. Below, a brief description of each method is given, including their main advantages and disadvantages.

Decision tree. To model the relationships among the features and the target classes, the DT use a tree structure that can be represented as a set of the if-then rules. The DT is built using a recursive partitioning also known as *divide and conquer* where the training instances are recursively partitioned into disjoint subsets until some stopping criterion is met. The algorithm for constructing DT usually works top-down, at each step choosing the feature that is best for splitting the data. In order to determine which feature is the best one, various measures can be used, for instance, entropy, Gini, or misclassification error [51]. Generally, such measures define how pure or impure the obtained subsets are with respect to the target variable. The purer the obtained subsets, the better the split. In the obtained tree, the root node and internal nodes of the tree contain attribute test conditions, the edges correspond to the outcome of a test, and each of the leaf nodes (also known as terminal or decision nodes) is labeled with one class representing the most appropriate target value or contains a probability vector representing the probability of the target variable being of a particular class. An instance is classified by traversing from the root node to the leaf node according to the outcome of the attribute tests.

Among the advantages of the DTs is the interpretability – the obtained classification rules are the sequences of simple rules given in a tree form which is easy to assimilate. Moreover, DTs can be easily used with a mixture of numeric and categorical features, they do not require the feature scaling, and perform internal feature selection [52]. However, there is a high risk of overfitting with DT which happens when the tree grows large and becomes too specific for the training set. Furthermore, DTs can be non-robust, i.e. small changes in the training data can lead to large changes in obtained classification rules [53].

K-nearest neighbors. KNN is an instance-based learning technique that does not have a specialized training phase but performs a classification at runtime directly based on the training data which is stored in memory. The idea of the KNN is to label new instances based on their similarity to already labelled instances in the training data. The similarity is determined using a distance measure, where the Euclidean distance is among the most frequently used ones. In general, for the new data point that needs to be classified, the *k* closest neighbors (training data points) are found and using a majority vote the class which appears most frequently in the defined neighborhood is assigned to that new data point. This way, instead of trying to draw decision boundaries across the whole space, the KNN classifier makes a decision based on the local information.

The main advantages of the KNN method include its simplicity, interpretability, and adaptability to irregular feature spaces [54, 55]. However, the KNN classifier is sensitive to the curse of dimensionality [19, 56] as in high dimensional data, the data points are relatively distant from each other considering all different dimensions. Furthermore, this method can have a poor run-time performance if the training set is large.

Logistic regression. LR is a parametric classification method which models the probability of class-membership based on the values of a set of given features. In its basic form, the LR uses a logistic function, also called sigmoid function which constrains the probability estimates to between 0 and 1 and ensures that they sum up to one. This way the obtained probabilities are sensible for all values of the explanatory variables. In fact, LR model is a linear combination of the inputs, but this linear combination relates to the logarithm of odds *(log-odds*) where the odds is the ratio of the probability of the predicted event occurring to the probability of that event not occurring. Differently from the DT and KNN, LR model has parameters that are learnt from the training data using a learning algorithm.

The LR models are widely used due to their simplicity and possibility to interpret the model's coefficients easily in terms of odds ratios [57, 58]. LR works well on the linearly separable problems as it is a linear classifier which produces linear decision boundaries (if nonlinear versions of the explanatory variables are not included in the model, for instance, the squared values of explanatory variables) [57]. However, overfitting problem may arise in LR models, especially with high dimensional and/or sparse data [59].

Regarding how the classification methods are used in the system, two approaches are considered:

1) One classification method is randomly chosen and used for a specified number of GP generations, after which the random choice is repeated.
2) One classification method is used during the system training process.

In case of 1), the system does not depend on one specific classification method but tries to take advantage of several different methods in order to achieve better classification performance. In case of 2), the work of GP population is wholly directed to changing the data representation to make it more appropriate to a particular classification method.

When solving a given problem, the performance of a classifier is dependent on the specified hyper-parameter values. Each dataset transformation done by GP individuals implies a problem change and thus requires hyper-parameters of the classifier to be optimized. However, doing this for every dataset returned by GP individuals highly increases the time required to train a system. For this reason, the hyper-parameters are tuned only when a new GP population is initialized, i.e. once in each layer. For each individual in the initial generation, on a training set returned as output by that individual, the specified hyper-parameters of a classifier are exhaustively explored using a cross-validated grid search over a specified parameter grid. Those hyper-parameters that result in highest F1 score on the left-out data are selected and used when evaluating fitness of the individual. Once all individuals in the initial generation are evaluated, the hyper-parameter values that were selected on the dataset returned as output by the best-of-generation individual are set in the next generations.

3.1.5. Genetic operators

As the GP individual consists of multiple trees, the traditional genetic operators (subtree mutation and standard crossover that were described in 2.2.5) are applied, with a given probability, to all its trees. In other words, for each *i = 1, 2, 3, …, n*, where *n* is the number of trees in the individual, the tree T_i is mutated with a given probability p_m, where p_m is a parameter of the algorithm. In case of crossover, two individuals *parent1* and *parent2* are selected, each tree in the *parent1* is paired with a randomly selected tree from the *parent2*, and then for each *i = 1, 2, 3, …, n*, where *n* is the number of trees in the *parent1*, the pair of trees $Pair_i$ is crossed over with a given probability p_c, where p_c is a parameter of the algorithm. This way, using the crossover operator one offspring is obtained which consists of the same number of trees as the first individual, yet it carries some genetic information from both parents.

It is important to highlight that mutation and crossover operators are applied independently to each tree or pair of trees. This implies that, although all trees in the individual to which mutation is applied have the same probability of being mutated and all pairs of trees have the same probability of being crossed over, some of the trees would be mutated whereas others would not, and respectively, some of the pairs of trees would be crossed over while others would not (in such case, a respective tree from parent one is replicated).

Because of its structure, the GP multi-tree individual can be treated as an individual of GA [60]. There each tree in GP individual corresponds to one gene of GA individual. This allows the usage of some traditional genetic operators of GA, for instance, one-point crossover. Given two GP individuals *parent1* and *parent2* as parents, a random crossover point is chosen, and the offspring is generated by taking all the trees that are in the positions from the crossover point to the left from *parent1* and all the trees that are in the positions from the crossover point to the right from *parent2* as shown in Figure 7. This way the obtained offspring outputs a new dataset that contains some features generated by *parent1* and some by *parent2*.

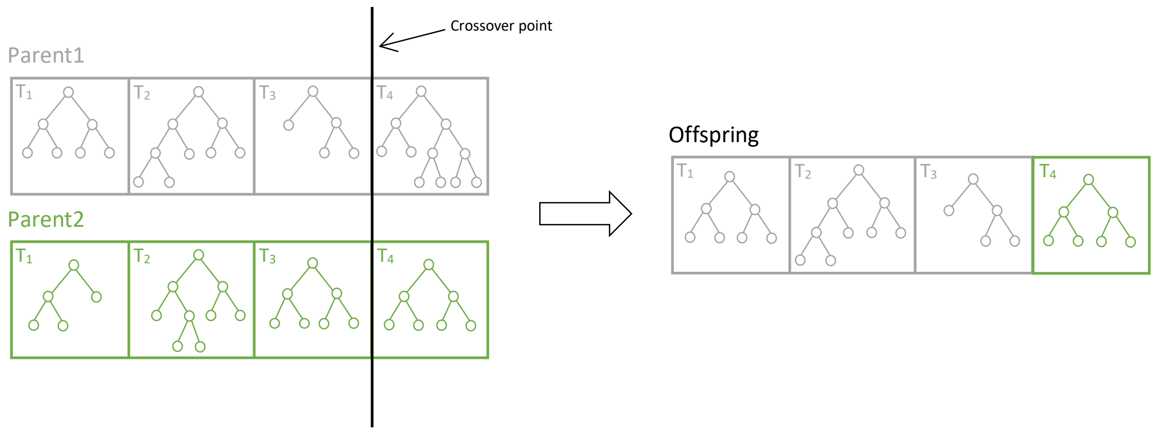

Figure 7. One-point crossover.

3.1.6. Elitism

A common approach to ensure that the quality of the obtained solution will not decrease from generation to generation is to use elitism strategy. Using this strategy, some of the best individuals obtained in the current generation are carried over to the next generation unaltered thus preserving the best genetic material and making it available for further improvement.

In LGP, elitism is used in two ways. First, the usual elitism strategy is used within each layer by copying the best individual in the current generation to the next generation and making it available for further evolution. Secondly, the elitism strategy is used between the GP layers. As in each layer a new GP population is initialized, there is a risk that the best individual obtained in the next layer will be worse than the best individual in the current layer. To reduce this risk, the best individual obtained in the current layer is always retained. When the best individual in the last generation in the new layer is obtained, it is compared to the best individual of the previous layer and if it is fitter, it becomes the best so far obtained solution, otherwise the best individual from the previous layer stays as the best of best individual. Differently from the elitism strategy within the layers, the best-of-layer individual is only retained but not inserted into the new layer.

4. EXPERIMENTAL STUDY

The objective of experimental study is to test the performance of the proposed LGP system. The key question is whether the GP that uses a layered structure can reduce the dimensionality of the dataset while improving the classification performance of simple classifiers. In the following subsections, test problems, design of experiments, and experimental results are presented.

4.1. DATASETS

In the performed experiments, 5 real-world binary classification problems were used. In each of these problems, the objective is to discriminate between the two classes: ready and not ready biodegradable chemicals (QSAR Biodegradation dataset [61] that hereinafter will we referred to as BIODEG); sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock (Sonar dataset [62] that hereinafter will we referred to as SONAR); benign and malignant tumors (Wisconsin Diagnostic Breast Cancer dataset [63] that hereinafter will we referred to as WDBC); *good* and *bad* radar system returns (Ionosphere dataset [64] that hereinafter will we referred to as IONO); approved and rejected credit card applications (Credit Approval dataset [65] that hereinafter will we referred to as CREDIT).

The datasets differ in terms of the number of features and instances, the features/instances ratios, and types of the features. The number of features varies from 30 to 60, the number of instances varies from 208 to 1055, and the features/instances ratio varies from 0.039 to 0.288. The WDBC dataset contains only numeric features. In BIODEG dataset, there are both numerical and categorical features, with the latter already provided in a numeric form of 1/0 corresponding to the presence/absence of a certain feature. In SONAR dataset, all features are numerical in a range [0.0, 1.0]. In IONO dataset, the features are numerical, continuous. The CREDIT dataset originally contains both numerical and categorical features, 15 in total, but the categorical ones were converted into dummy variables for each category after that obtaining 37 features in total. All datasets, except for CREDIT, have no missing values. In the CREDIT dataset, 7 features have missing values (67 missing values in total), which were replaced with mode for categorical features and median for numeric features. Table 1 summarizes the main characteristics of the datasets. All these datasets are available at UCI Machine Learning Repository [66].

Table 1. Description of datasets.

Dataset	# Classes	# Features	# Instances	Features/instances ratio	Target class proportion (0/1)
QSAR Biodegradation (*BIODEG*)	2	41	1055	0.039	0.66/0.34
Sonar (*SONAR*)	2	60	208	0.288	0.53/0.47
Wisconsin Diagnostic Breast Cancer (*WDBC*)	2	30	569	0.053	0.63/0.37
Ionosphere (*IONO*)	2	34	351	0.097	0.64/0.36
Credit approval (*CREDIT*)	2	37	690	0.054	0.56/0.44

4.2. EXPERIMENTAL SETUP

The performed experiments were divided into two phases. In the first phase, for each test problem, different configurations of the LGP were tested. In the second phase, the performance of the LGP with the best configuration found in the first phase was compared with the performance of the classification methods applied directly on the original datasets. The classification methods applied on the original datasets were the same ones that were used in the LGP system, i.e. if the best performance was achieved by LGP where the classifier was LR, then its performance was compared with the LR trained and tested on the original data.

Considering the stochastic nature of GP and the volatility of the results depending on the data partition, for each test problem the experiments were run 30 times using different seeds of the pseudo-random number generator, and then the results were summarized using the medians. Each time, the original dataset was randomly partitioned into training and test sets where the former contained 80% of samples and the latter contained the remaining 20% of samples. In the first experimental phase, only the training set was used. In each run, this set was further partitioned into sub-training set containing 80% of the full training set samples and development set containing the remaining 20% of the full training set samples. Then different LGP configurations were tested. The sub-training set was used by GP individuals for learning while the decision which configuration to choose was made based on the median fitness value (median F1 score) on the development set achieved by the best-of-run individuals. Once the best configurations of LGP were found for all test problems, these configurations were used in the next experimental phase. In the second phase, the full training set was used by GP individuals for learning and the test set which contained previously unseen data was used for the final performance evaluation. To evaluate the performance of classification method applied directly on the original dataset, the method was applied for 30 times on each test problem, using the same set of 30 different data partitions as with LGP. The representation of how the dataset was split and which sets where used in each experimental phase are showed in Figure 8.

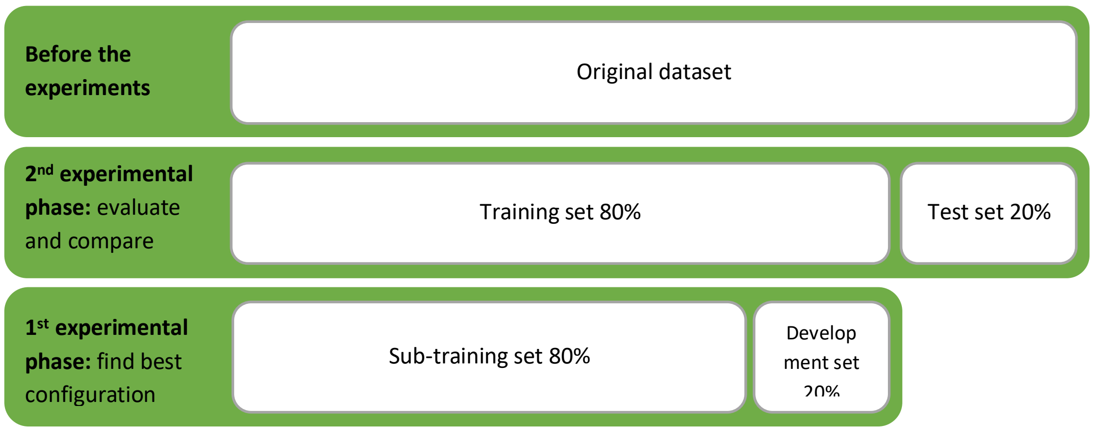
Before the experiments
Original dataset
2nd experimental phase: evaluate and compare
Training set 80%
Test set 20%
1st experimental phase: find best configuration
Sub-training set 80%
Development set 20%

Figure 8. Dataset split representation.

In addition to the usual GP parameters, there are some new parameters that are specific to the LGP system. That is, the number of layers which defines the size of the system and thus has impact on the time required for the training phase (the more layers there are, the longer the training lasts as more GP populations have to be evolved), and also the dataset dimensionality reduction step size, which defines by what number of features to reduce the dimensionality of the dataset from layer to layer, and the classifier used for evaluating the fitness of individuals. Regarding the dimensionality reduction step and the size of the system, the following variants were considered:

1. In each new layer, the number of features in the transformed dataset is equal to the largest integer that is not greater than the number of features in the input dataset in that layer divided by 2. This way, in each layer, the dimensionality is reduced by half, approximately. New GP layers are added until the original data are transformed into a one-dimensional space. For WDBC dataset this required 4 GP layers, and for all other test problems – 5 layers.
2. In each new layer, the number of features is reduced by the square root of the number of features in the input dataset in that layer. Similarly to the previously described variant, the dimensionality is reduced gradually, however, slower, and in order to transform the input data into a one-dimensional space it would require a larger LGP system, i.e. more GP layers. However, for each test problem, we used the same number of layers as in the first variant so

as to ensure that when comparing different variants, the final solutions would be obtained using the same total number of generations.

3. The GP population contains individuals of different sizes and so, the dimensionality reduction step depends on a particular individual. Let *n* be the number of features in the input dataset in a particular layer. Then in that layer, each individual *i* in the initial population is composed of m_i trees, where m_i is an integer number randomly chosen from a range [1, n-1]. This allows to automatically determine the dimensionality of the transformed dataset during the evolution of population instead of having to specify a dimensionality reduction step beforehand and ensures that the transformed dataset has fewer features than the input dataset. However, in each layer, the search space for the GP population is larger compared to the previously described variants. For each test problem, the same number of layers was used as in the first variant.
4. Although the idea of the proposed system is to reduce the dimensionality of the input dataset, a variant where the dimensionality is kept the same from layer to layer is also considered in order to observe the behavior of the system in such case. For each test problem, the same number of layers was used as in the first variant.

Regarding the classification methods applied on the transformed dataset, the following variants were considered:

1. In each generation, a classification method is randomly chosen with a uniform probability from {DT, KNN, LR}. This way, it may happen that in each generation a different classifier is used to evaluate individuals.
2. A classification method is randomly chosen with a uniform probability from {DT, KNN, LR} and is used for 10 generations, after which the selection is repeated. This ensures some stability in fitness evaluation as the same classifier is used for assessing individuals for several generations.
3. In the first generation in each layer, a classification method is randomly chosen with a uniform probability from {DT, KNN, LR}, and subsequently this method is used to evaluate fitness of individuals in all generations in that layer, i.e. the classifiers may differ only between the layers.
4. In all layers in all generations the same classification method is used which is DT or KNN or LR.

For the first variant of dimensionality reduction step, all variants of classification method choice were tested. Other three variants of dimensionality reduction step were tested in combination with the variant where the classification method was randomly chosen in each generation. Thus, in total 9 different combinations of these parameters tested. Furthermore, three genetic operators, namely subtree mutation, standard crossover, and one-point crossover were tested with different probabilities which are given in Table 2. In fact, these are the probabilities of applying a respective genetic operator to an individual. If either the subtree mutation or standard crossover was applied, the probability of mutating each tree or crossing over each pair of trees was 0.5.

Table 2. Tested probabilities of applying genetic operators to GP individual.

No.	Prob. of subtree mutation	Prob. of standard crossover	Prob. of one-point crossover
1	1.0	0	0
2	0.8	0.2	0
3	0.5	0.5	0
4	0.2	0.8	0
5	0.8	0	0.2
6	0.5	0	0.5
7	0.2	0	0.8

Other parameter settings that were common in all experiments are listed in Table 3. In each layer, GP population of 100 individuals was evolved for 30 generations and then the dataset returned as output by the best individual in the last generation was transferred to a subsequent layer. The number of generations per layer was chosen such that it is a multiple of 3 in order to ensure that all three classification methods may be used for an equal number of generations in the cases when the classification method is randomly chosen in each generation or when the same classification method is used for 10 subsequent generations. Moreover, a relatively small number of generations per layer was chosen considering the overall time required to evolve all populations. However, this number of generations should be enough to obtain a good solution.

Table 3. Parameter settings that were common in all experiments.

Parameter	Value
Runs	30
Population size	100
Number of generations in each GP layer	30
Selection	Tournament selection of size 5
Initialization	Initialization using subsets of features as described in 3.1.2, with the initial tree depth in range [1, 4]
Function set	{+, -, *, /[1]}
Terminal set	Variables of considered problem, and constants that are randomly generated from a range [-1, 1]
Elitism	Keep best-of-generation individual and best-of-layer individual
Termination criterion in each layer	Maximum number of generations is reached

The classification methods (DT, KNN, LR) used in the system were the ones that are available in Scikit Learn [67]. Their hyper-parameters were tested and selected using a parameter grid shown in Table 4 in each initial generation as described in Section 3.1.4. For KNN, the Euclidean distance was used as a distance metric. The values tested as the number of neighbors to use (*k*) were chosen such that they would include smaller and larger values. Although large values of *k* make the method computationally expensive, they decrease the chance that the decision will be influenced by the noise in training data [68]. Moreover, an often-used rule of thumb is to choose *k* which equals the square root of the number of instances in the training set [69, 70, 71] and so, such value was included in the parameter grid for each considered dataset. DT has many more hyper-parameters that could be potentially tested, but that would substantially slow down the computation. For this reason, the hyper-parameters that control the size of the tree and thus can help to prevent the tree from overfitting, namely the maximum tree depth and the minimum number of samples required to be at a leaf node, were chosen to be optimized, and other hyper-parameters of DT were used as default. For LR, the *liblinear* solver was used. *L1* and *L2* norms used in the penalization were tested with different values of parameter *C*, which is the inverse of regularization strength. All other parameters were used as default.

[1] Protected as that can help to prevent in [20].

Table 4. Parameter grid.

Parameter	Values
DT: maximum depth of the tree	1, 2, 3, 4
DT: minimum number of samples required to be at a leaf node	10, 25, 50, 100
KNN: number of neighbors to use	5, 7, 9, 25, $\sqrt{\#\text{ training instances}}$
LR: the norm used in the penalization	L1, L2
LR: C	1 x 10^{-2}, 1 x 10^{-1}, 1 x 10^{0}, 1 x 10^{1}, 1 x 10^{2}

The hyper-parameters of the classifiers that were applied on the original datasets were tuned too. For that, the same parameter grid given in Table 4 was used. To analyze the statistical significance of the results, the Wilcoxon signed-rank test was used considering a significance level of 0.05. The project was developed using Python programming language. As a base for LGP system, *gplearn* package [72] was used, and then it was further developed based on the needs of this project.

4.3. EXPERIMENTAL RESULTS

In this sub-section, the obtained results are presented. Table 5, Table 6, Table 7, Table 8, and Table 9 summarize the main results for BIODEG, WDBC, SONAR, IONO, and CREDIT problems respectively obtained in the first experimental phase in 30 runs where the objective was to test different LGP configurations and choose the best one in terms of median F1 score achieved on the development set. In these tables, the following information is provided:

- *LGP config.* column refers to the tested combination of dimensionality reduction step and classification method choice, where:
  1. DIV2GEN1 – in each subsequent layer, the dimensionality of the input dataset was reduced by half, and the classifier applied on the transformed dataset was randomly chosen in each generation.
  2. DIV2GEN10 – in each subsequent layer, the dimensionality of the input dataset was reduced by half, and the classifier applied on the transformed dataset was randomly chosen for every 10 generations.

3. DIV2GEN30 – in each subsequent layer, the dimensionality of the input dataset was reduced by half, and the classifier applied on the transformed dataset was randomly chosen for every 30 generations.
4. DIV2DT – in each subsequent layer, the dimensionality of the input dataset was reduced by half, and the classifier applied on the transformed dataset was DT.
5. DIV2KNN – in each subsequent layer, the dimensionality of the input dataset was reduced by half, and the classifier applied on the transformed dataset was KNN.
6. DIV2LR – in each subsequent layer, the dimensionality of the input dataset was reduced by half, and the classifier applied on the transformed dataset was LR.
7. SQRTGEN1 – in each subsequent layer, the dimensionality of the input dataset was reduced by the square root of the number of features in the input dataset in that layer, and the classifier applied on the transformed dataset was randomly chosen in each generation.
8. NOREDUCGEN1 – in the subsequent layers, the dataset was transformed without changing its dimensionality, and the classifier applied on the transformed dataset was randomly chosen in each generation.
9. RANDGEN1 – the initial populations contained GP individuals of different sizes as described in 4.2, and the classifier applied on the transformed dataset was randomly chosen in each generation.

- *Mutation/Crossover* column shows with which probability of applying mutation or crossover operator to an individual the highest median F1 score of 30 runs was achieved on the development set. There *P_MUT* refers to the probability of applying subtree mutation operator, *P_XO* – standard crossover operator, and *P_XO1P* – one-point crossover operator.
- *Fitness on training set* column shows the median fitness of best individual achieved on the training set, with the standard deviation provided in the parentheses.
- *Fitness on development set* column shows the median fitness of best individual achieved on the development set, with the standard deviation provided in the parentheses.
- The column *# trees* shows the median number of trees that the best individual was composed of, with the minimum and maximum number of trees given in the parentheses. In other words, this is the dimensionality of the obtained transformed dataset on which the best classification performance was noticed.
- *Avg. tree depth* column gives the median of average tree depth in the best individual.
- *Max. tree depth* column gives the median of maximum tree depth in the best individual.
- *Avg. tree size* column shows the median of the average number of nodes in the tree of the best individual.

- *Max. tree size* column shows the median of the maximum number of nodes in the tree of the best individual.
- In *Classif. method* column, the classification method which was applied on the transformed dataset outputted by the best individual, is given. If the classification method was randomly chosen and used for a certain number of generations, it might happen that in different runs a different classification method was used on the final transformed dataset. In such cases, the classification method that was used in the largest number of runs is given.
- *Layer of best individual* column shows in how many runs the best individual was obtained in a certain layer. There the percentage is out of the total number of runs.

Looking at the results obtained on different datasets and in different variants, it can be noticed that most often, the best individual was found in the first few layers. This implies that from a certain point LGP was often not able to improve the previously obtained solutions anymore. There is the largest percentage of runs where the best individual was obtained in the first layer, following by the second and third layers. A slightly different behavior was observed for DIV2DT where for all considered problems, the best individual was more often obtained in the second layer rather than in the first layer.

When considering the RANDGEN1 variant, where instead of a having a predefined dimensionality reduction step size the algorithm automatically determined the dimensionality of the transformed dataset during the evolution of population, it can be noted that for all considered problems, it reduced the dimensionality of the dataset slower. The result of this was that the final transformed dataset contained more features compared to the number of features in the final transformed datasets obtained in other LGP variants where the dimensionality reduction step size was specified in advanced. Moreover, it can be also noticed that keeping the same number of features in the next layers was not useful as regarding the F1 score achieved on the development set, the results of NOREDUCGEN1 variant were worse than those of other tested variants.

The LGP showed an ability to control the sizes of the trees that the individuals were composed of. As several layers were used and in each such layer a new population was initialized with the individuals composed of the trees with the depth not exceeding the predefined maximum depth, this did not allowed the trees to grow very large. As a result, for all considered problems, in all tested variants the median of maximum tree depth of best individual was less than 10 and the median of maximum tree size was less than 40 nodes.

Table 5. Results of different LGP configurations on BIODEG dataset. The configuration which showed best results in terms of median F1 score on the development set is highlighted in grey.

No.	LGP config.	Mutation/ Crossover	Fitness on training set	Fitness on development set	# trees	Avg. tree depth	Max. tree depth	Avg. tree size	Max. tree size	Classif. method	Layer of best individual
1	DIV2GEN1	P_MUT=0.5 P_XO1P=0.5	0.873 (0.010)	0.775 (0.041)	20 (5-20)	2.1	4	8	29	KNN	L1: 17 (56.7%) L2: 6 (20.0%) L3: 7 (23.3%)
2	DIV2GEN10	P_MUT=0.2 P_XO1P=0.8	0.867 (0.017)	0.767 (0.047)	20 (5-20)	2.2	4	9.3	31	KNN	L1: 19 (63.3%) L2: 7 (23.3%) L3: 4 (13.3%)
3	DIV2GEN30	P_MUT=0.2 P_XO=0.8	0.875 (0.015)	0.773 (0.038)	20 (2-20)	2.8	7	10	30	KNN	L1: 16 (53.3%) L2: 7 (23.3%) L3: 5 (16.7%) L4: 2 (6.7%)
4	DIV2DT	P_MUT=0.8 P_XO=0.2	0.862 (0.010)	0.763 (0.050)	10 (5-20)	2.4	5	9.4	31	DT	L1: 7 (23.3%) L2: 17 (56.7%) L3: 6 (20.0%)
5	DIV2KNN	P_MUT=0.5 P_XO1P=0.5	0.887 (0.009)	0.770 (0.039)	20 (10-20)	2	4	8.2	31	KNN	L1: 22 (73.3%) L2: 8 (26.7%)
6	DIV2LR	P_MUT=0.2 P_XO=0.8	0.849 (0.011)	0.802 (0.046)	20 (5-20)	2.6	6	8.4	28	LR	L1: 20 (66.7%) L2: 8 (26.7%) L3: 2 (6.7%)
7	SQRTGEN1	P_MUT=0.2 P_XO1P=0.8	0.881 (0.009)	0.770 (0.041)	29 (15-35)	2	4	8.2	31	KNN	L1: 11 (36.7%) L2: 7 (23.3%) L3: 8 (26.7%) L4: 1 (3.3%) L5: 3 (10.0%)
8	NOREDUCGEN1	P_MUT=0.5 P_XO=0.5	0.873 (0.011)	0.771 (0.042)	41 (41-41)	2.2	6	8	35	KNN	L1: 12 (40.0%) L2: 5 (16.7%) L3: 8 (26.7%) L4: 2 (6.7%) L5: 3 (10.0%)
9	RANDGEN1	P_MUT=1	0.871 (0.010)	0.761 (0.060)	37 (17-40)	2.1	5	8.1	33	KNN	L1: 16 (53.3%) L2: 7 (23.3%) L3: 3 (10.0%) L4: 2 (6.7%) L5: 2 (6.7%)

Table 6. Results of different LGP configurations on WDBC dataset. The configuration which showed best results in terms of median F1 score on the development set is highlighted in grey.

No.	LGP config.	Mutation/ Crossover	Fitness on training set	Fitness on development set	# trees	Avg. tree depth	Max. tree depth	Avg. tree size	Max. tree size	Classif. method	Layer of best individual
1	DIV2GEN1	P_MUT=0.5 P_XO1P=0.5	0.996 (0.004)	0.956 (0.026)	15 (3-15)	2.2	4	9.7	31	LR	L1: 26 (86.7%) L2: 3 (10.0%) L3: 1 (3.3%)
2	DIV2GEN10	P_MUT=0.2 P_XO1P=0.8	0.994 (0.005)	0.955 (0.024)	15 (7-15)	2.2	4	9.5	31	LR	L1: 25 (83.3%) L2: 5 (16.7%)
3	DIV2GEN30	P_MUT=0.8 P_XO1P=0.2	0.993 (0.006)	0.955 (0.026)	15 (3-15)	2.3	4	9.3	30	LR	L1: 19 (63.3%) L2: 8 (26.7%) L3: 3 (10.0%)
4	DIV2DT	P_MUT=0.5 P_XO=0.5	0.989 (0.006)	0.949 (0.036)	7 (1-15)	2.7	5	10.7	24	DT	L1: 6 (20.0%) L2: 19 (63.3%) L3: 3 (10.0%) L4: 2 (6.7%)
5	DIV2KNN	P_MUT=0.5 P_XO=0.5	0.993 (0.004)	0.954 (0.029)	7 (3-15)	2.7	5	9.9	29	KNN	L1: 13 (43.3%) L2: 15 (50.0%) L3: 2 (6.7%)
6	DIV2LR	P_MUT=0.8 P_XO1P=0.2	1.000 (0.003)	0.957 (0.031)	15 (7-15)	2.2	4	9.6	31	LR	L1: 27 (90.0%) L2: 3 (10.0%)
7	SQRTGEN1	P_MUT=0.5 P_XO1P=0.5	1.000 (0.002)	0.955 (0.031)	20 (12-25)	2.2	4	9.1	31	LR	L1: 14 (46.7%) L2: 10 (33.3%) L3: 4 (13.3%) L4: 2 (6.7%)
8	NOREDUCGEN1	P_MUT=1	1.000 (0.003)	0.944 (0.028)	30 (30-30)	2.2	4	9.5	31	LR	L1: 15 (50.0%) L2: 12 (40.0%) L3: 1 (3.3%) L4: 2 (6.7%)
9	RANDGEN1	P_MUT=0.5 P_XO=0.5	1.000 (0.004)	0.949 (0.026)	25 (4-29)	2.3	5	9	31	LR	L1: 16 (53.3%) L2: 8 (26.7%) L3: 3 (10.0%) L4: 3 (10.0%)

Table 7. Results of different LGP configurations on SONAR dataset. The configuration which showed best results in terms of median F1 score on the development set is highlighted in grey.

No.	LGP config.	Mutation/ Crossover	Fitness on training set	Fitness on development set	# trees	Avg. tree depth	Max. tree depth	Avg. tree size	Max. tree size	Classif. method	Layer of best individual
1	DIV2GEN1	P_MUT=0.5 P_XO1P=0.5	0.992 (0.023)	0.758 (0.090)	30 (7-30)	2.1	4	8.7	31	LR	L1: 26 (86.7%) L2: 2 (6.7%) L3: 2 (6.7%)
2	DIV2GEN10	P_MUT=0.5 P_XO1P=0.5	0.980 (0.023)	0.725 (0.088)	30 (15-30)	2	4.5	8.8	31	LR	L1: 28 (93.3%) L2: 2 (6.7%)
3	DIV2GEN30	P_MUT=0.2 P_XO=0.8	0.975 (0.020)	0.746 (0.091)	30 (7-30)	2.4	6	9	32	LR	L1: 20 (66.7%) L2: 6 (20.0%) L3: 4 (13.3%)
4	DIV2DT	P_MUT=0.2 P_XO=0.8	0.960 (0.013)	0.750 (0.090)	15 (3-30)	2.1	6	7.1	20	DT	L1: 5 (16.7%) L2: 12 (40.0%) L3: 10 (33.3%) L4: 3 (10.0%)
5	DIV2KNN	P_MUT=0.2 P_XO=0.8	0.975 (0.011)	0.759 (0.087)	15 (7-30)	2	7	6.8	25	KNN	L1: 14 (46.7%) L2: 13 (43.3%) L3: 3 (10.0%)
6	DIV2LR	P_MUT=0.8 P_XO=0.2	1.000 (0.005)	0.736 (0.093)	30 (15-30)	2	5	8.4	31	LR	L1: 29 (96.7%) L2: 1 (3.3%)
7	SQRTGEN1	P_MUT=0.2 P_XO1P=0.8	1.000 (0.007)	0.721 (0.101)	52 (26-52)	1.9	4	8.1	31	LR	L1: 20 (66.7%) L2: 9 (30.0%) L5: 1 (3.3%)
8	NOREDUCGEN1	P_MUT=0.5 P_XO=0.5	1.000 (0.001)	0.699 (0.103)	60 (60-60)	2	4	8.4	31	LR	L1: 16 (53.3%) L2: 7 (23.3%) L3: 5 (16.7%) L4: 1 (3.3%) L5: 1 (3.3%)
9	RANDGEN1	P_MUT=0.8 P_XO=0.2	1.000 (0.018)	0.705 (0.097)	54 (28-59)	2	5	8.8	31	LR	L1: 16 (53.3%) L2: 10 (33.3%) L3: 2 (6.7%) L4: 2 (6.7%)

Table 8. Results of different LGP configurations on IONO dataset. The configuration which showed best results in terms of median F1 score on the development set is highlighted in grey.

No.	LGP config.	Mutation/ Crossover	Fitness on training set	Fitness on development set	# trees	Avg. tree depth	Max. tree depth	Avg. tree size	Max. tree size	Classif. method	Layer of best individual
1	DIV2GEN1	P_MUT=0.2 P_XO=0.8	0.956 (0.014)	0.838 (0.062)	8 (1-17)	2.2	5	7.2	21	DT	L1: 13 (43.3%) L2: 8 (26.7%) L3: 6 (20.0%) L4: 2 (6.7%) L5: 1 (3.3%)
2	DIV2GEN10	P_MUT=0.2 P_XO=0.8	0.957 (0.014)	0.833 (0.066)	8 (2-17)	2.3	5	8.3	23	DT	L1: 12 (40.0%) L2: 8 (26.7%) L3: 7 (23.3%) L4: 3 (10.0%)
3	DIV2GEN30	P_MUT=0.5 P_XO=0.5	0.962 (0.015)	0.857 (0.066)	12 (4-17)	2	4.5	6.9	20	DT	L1: 15 (50.0%) L2: 12 (40.0%) L3: 3 (10.0%)
4	DIV2DT	P_MUT=0.5 P_XO=0.5	0.969 (0.010)	0.861 (0.068)	8 (2-17)	2.1	5	7.6	19	DT	L1: 8 (26.7%) L2: 10 (33.3%) L3: 9 (30.0%) L4: 3 (10.0%)
5	DIV2KNN	P_MUT=0.5 P_XO=0.5	0.960 (0.012)	0.846 (0.063)	8 (4-17)	2	4.5	6.8	16	KNN	L1: 3 (10.0%) L2: 14 (46.7%) L3: 13 (43.3%)
6	DIV2LR	P_MUT=0.5 P_XO=0.5	0.969 (0.017)	0.868 (0.067)	17 (4-17)	2.1	5	6.5	17	LR	L1: 17 (56.7%) L2: 7 (23.3%) L3: 6 (20.0%)
7	SQRTGEN1	P_MUT=0.2 P_XO=0.8	0.988 (0.013)	0.842 (0.074)	23 (10-28)	2.3	6.5	8.1	29	LR	L1: 13 (43.3%) L2: 5 (16.7%) L3: 2 (6.7%) L4: 6 (20.0%) L5: 4 (13.3%)
8	NOREDUCGEN1	P_MUT=1	0.981 (0.019)	0.811 (0.070)	34 (34-34)	2.1	5	7.8	31	LR	L1: 12 (40.0%) L2: 9 (30.0%) L3: 5 (16.7%) L4: 2 (6.7%) L5: 2 (6.7%)
9	RANDGEN1	P_MUT=0.8 P_XO1P=0.2	0.972 (0.017)	0.835 (0.069)	28 (6-33)	2	5	7.6	31	LR	L1: 11 (36.7%) L2: 7 (23.3%) L3: 7 (23.3%) L4: 4 (13.3%) L5: 1 (3.3%)

Table 9. Results of different LGP configurations on CREDIT dataset. The configuration which showed best results in terms of median F1 score on the development set is highlighted in grey.

No.	LGP config.	Mutation/ Crossover	Fitness on training set	Fitness on development set	# trees	Avg. tree depth	Max. tree depth	Avg. tree size	Max. tree size	Classif. method	Layer of best individual
1	DIV2GEN1	P_MUT=0.2 P_XO1P=0.8	0.913 (0.011)	0.833 (0.042)	9 (1-18)	2.2	4	10.2	31	KNN	L1: 10 (33.3%) L2: 12 (40.0%) L3: 4 (13.3%) L4: 3 (10.0%) L5: 1 (3.3%)
2	DIV2GEN10	P_MUT=0.2 P_XO1P=0.8	0.906 (0.011)	0.832 (0.029)	9 (4-18)	2.2	4	9.6	31	KNN	L1: 12 (40.0%) L2: 9 (30.0%) L3: 9 (30.0%)
3	DIV2GEN30	P_MUT=0.8 P_XO=0.2	0.906 (0.011)	0.836 (0.035)	9 (1-18)	3.1	6	11.3	31	KNN	L1: 10 (33.3%) L2: 6 (20.0%) L3: 7 (23.3%) L4: 6 (20.0%) L5: 1 (3.3%)
4	DIV2DT	P_MUT=1	0.911 (0.013)	0.835 (0.036)	4 (2-18)	2.5	4	8.8	17	DT	L1: 1 (3.3%) L2: 11 (36.7%) L3: 16 (53.3%) L4: 2 (6.7%)
5	DIV2KNN	P_MUT=0.5 P_XO1P=0.5	0.920 (0.009)	0.833 (0.041)	9 (1-18)	2.3	4	10.1	31	KNN	L1: 10 (33.3%) L2: 12 (40.0%) L3: 6 (20.0%) L4: 1 (3.3%) L5: 1 (3.3%)
6	DIV2LR	P_MUT=1	0.893 (0.012)	0.844 (0.034)	9 (2-18)	2.4	5	9.7	28	LR	L1: 13 (43.3%) L2: 15 (50.0%) L3: 1 (3.3%) L4: 1 (3.3%)
7	SQRTGEN1	P_MUT=0.2 P_XO1P=0.8	0.920 (0.009)	0.831 (0.039)	18 (12-31)	2.1	4	9	31	KNN	L1: 4 (13.3%) L2: 6 (20.0%) L3: 5 (16.7%) L4: 10 (33.3%) L5: 5 (16.7%)
8	NOREDUCGEN1	P_MUT=0.8 P_XO=0.2	0.916 (0.011)	0.825 (0.036)	37 (37-37)	2.2	6.5	8.4	36	LR	L1: 2 (6.7%) L2: 3 (10.0%) L3: 7 (23.3%) L4: 8 (26.7%) L5: 10 (33.3%)
9	RANDGEN1	P_MUT=0.8 P_XO=0.2	0.910 (0.007)	0.833 (0.052)	24 (11-36)	2.2	5.5	7.7	31	DT	L1: 5 (16.7%) L2: 4 (13.3%) L3: 9 (30.0%) L4: 6 (20.0%) L5: 6 (20.0%)

In the discussed results tables, for each considered problem, the LGP configuration that showed the best result in terms of F1 score on the development set is highlighted in grey. On 4 out of 5 considered problems, namely BIODEG, WDBC, IONO, and CREDIT the largest F1 score on the development set was achieved by reducing the dimensionality of the input dataset by half in the subsequent layers and using LR as the classifier on the transformed dataset. Similarly to that, on SONAR dataset, the best result in terms of F1 score on the development set was achieved by also reducing the number of features in the transformed dataset by half in the subsequent layers, however using KNN as the classifier on the outputted dataset instead of LR. When the same classification method is used in all generations, there is more stability in the fitness evaluation of the individuals, which seems to be useful for the algorithm. Regarding which genetic operator – mutation or crossover – is more useful in LGP, there is no clear trend as on some datasets better results were achieved when the mutation operator was applied with a higher probability while on other datasets application of a standard or one-point crossover with a higher probability worked better. For instance, for CREDIT, the best combination was DIV2LR and mutation operator applied with probability of 1, which implies mutating every selected individual. For IONO dataset, DIV2LR showed better results when either the mutation operator or standard crossover operator was applied with probability of 0.5. For BIODEG problem, DIV2LR worked better when the mutation was applied with probability 0.2 and standard crossover with probability of 0.8. Similarly, for SONAR dataset, DIV2KNN also showed better results when the mutation was applied with probability of 0.2 and standard crossover with probability of 0.8. For WDBC, LGP showed better results when DIV2LR was used in a combination with mutation operator applied with probability of 0.8 and one-point crossover with probability of 0.2.

In the second experimental phase, for each test problem, the best configuration of LGP was run using the full training set for learning and test set containing previously unseen data for final evaluation. Then the performance of LGP was compared with the performance of the simple classifiers applied directly on the original datasets. The results of LGP obtained on the full training set and test set are given in Table 10. This table summarizes the results of 30 runs. *Fitness on training set* column shows the median fitness of best individual achieved on the full training set, with the standard deviation provided in the parentheses. *Fitness on test set* column shows the median fitness of best individual achieved on the test set, with the standard deviation provided in the parentheses. All other columns, namely *# trees, Avg. tree depth, Max. tree depth, Avg. tree size*, *Max. tree size, Classif. method, Layer of best individual* provide the same kind of information as the respective columns in the previously discussed tables of results.

For all considered problems, the best individual was most often obtained in the first layer. However, there were runs when the LGP kept improving the obtained solutions in the subsequent layers. For BIODEG problem, there were 12 runs (40% of total number of runs) when the best solution was obtained in the second or further layer, for WDBC problem – 3 such runs (10% of total number of runs), for SONAR problem – 14 such runs (46.7% of total number of runs), for IONO problem – 10 such runs (33.3% of total number of runs), for CREDIT problem – 10 such runs (33.3% of total number of runs). The lowest dimensional space to which the LGP transformed the original dataset was 2-dimensional space for BIODEG, IONO and CREDIT problems, 3-dimensional space for WDBC problem and 7-dimensionl space for SONAR dataset (note that SONAR dataset had the largest number of original features out of all considered datasets).

Table 10. Results of LGP on full training and test sets of different datasets.

Dataset	Fitness on training set	Fitness on test set	# trees	Avg. tree depth	Max. tree depth	Avg. tree size	Max. tree size	Classif. method	Layer of best individual
BIODEG	0.842 (0.010)	0.791 (0.035)	20 (2-20)	2.6	6	8.1	24	LR	L1: 18 (60.0%) L2: 8 (26.7%) L3: 2 (6.7%) L4: 2 (6.7%)
WDBC	0.997 (0.003)	0.952 (0.022)	15 (3-15)	2.3	5	9.4	31	LR	L1: 27 (90.0%) L2: 2 (6.7%) L3: 1 (3.3%)
SONAR	0.968 (0.008)	0.775 (0.068)	30 (7-30)	2.2	6	7.1	25	KNN	L1: 16 (53.3%) L2: 12 (40.0%) L3: 2 (6.7%)
IONO	0.960 (0.011)	0.854 (0.064)	17 (2-17)	2.2	5	7.6	23	LR	L1: 20 (66.7%) L2: 4 (13.3%) L3: 5 (16.7%) L4: 1 (3.3%)
CREDIT	0.888 (0.008)	0.851 (0.031)	18 (2-18)	2.2	5	8.3	31	LR	L1: 20 (66.7%) L2: 8 (26.7%) L4: 2 (6.7%)

In Table 11, for each test problem, the median F1 scores achieved on training and test sets by LGP and the simple classifiers applied directly on the original datasets are provided. The values in the parentheses are the standard deviations. The results on the test set for which there is a statistically significant difference are given in bold. In addition, Figure 9 shows the evolution of the median F1 score on training and test sets as the number of generations increases and the layers change. For comparison, in the plots, the dashed line shows the median F1 scores achieved by simple classifiers on the respective set of the original datasets.

For all considered problems, the LGP increased the F1 score on the training set. However, the performance on the test set varied depending on the dataset. For two problems, namely SONAR and IONO, LGP improved the F1 score on the test set and outperformed the simple classifiers applied directly on the original datasets, which was a statistically significant result. However, on BIODEG and WDBC problems, LGP was outperformed by simple classifiers. On CREDIT dataset, LGP achieved a lower F1 score on the test set than the simple classifier, but the difference was not statistically significant.

Table 11. Median F1 scores on training and test sets obtained using LGP and simple classifiers.

Dataset	Median F1 score on training set		Median F1 score on test set	
	LGP	Classifier on original dataset	LGP	Classifier on original dataset
BIODEG	0.842 (0.010)	0.830 (0.011)	0.791 (0.035)	**0.799 (0.030)**
WDBC	0.997 (0.003)	0.979 (0.003)	0.952 (0.022)	**0.976 (0.018)**
SONAR	0.968 (0.008)	0.869 (0.022)	**0.775 (0.068)**	0.763 (0.067)
IONO	0.960 (0.011)	0.935 (0.023)	**0.854 (0.064)**	0.800 (0.059)
CREDIT	0.888 (0.008)	0.849 (0.007)	0.851 (0.031)	0.856 (0.030)

Overall, the LGP showed quite different results on training and test sets which implies the existence of the overfitting. In Figure 9, it can be noticed that fitness on the training set continuously improved over generations and layers until it stabilized. However, on the test set, the fitness tended to improve for some generations, then at some point it decreased, and later stabilized at some value.

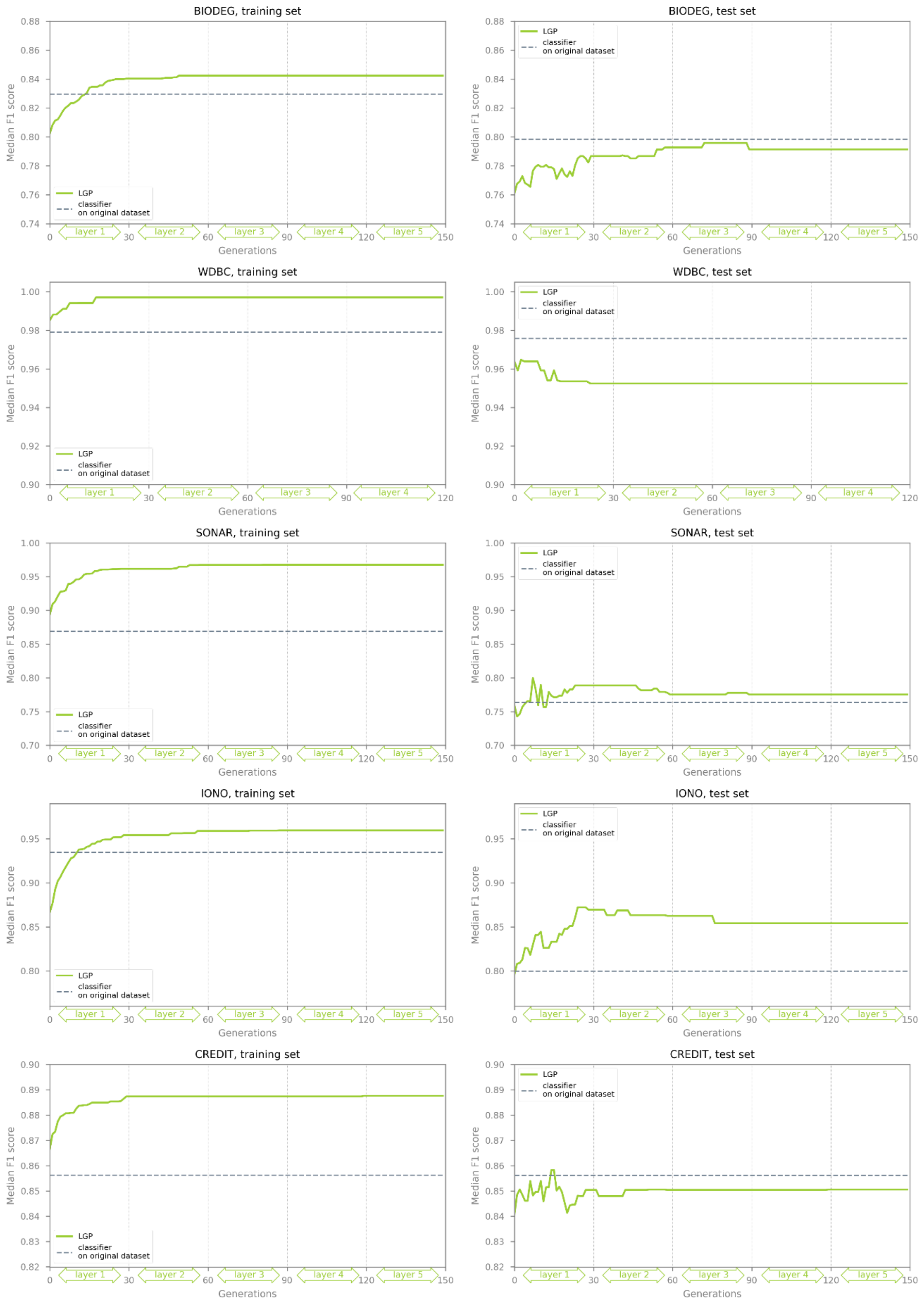

Figure 9. The evolution of F1 score on training and test sets as the number of generations increase and the layers change. For comparison, F1 scores achieved using simple classifiers on the original data are shown by dashed lines.

5. CONCLUSIONS AND FUTURE WORK

In this work, the LGP system was presented. The LGP implemented a layered structure where several GP populations were evolved sequentially and used to transform the original data into a lower dimensional space where the regularities in the data should be more easily detected by the classification algorithm. The performance of a classifier applied on the transformed dataset defined the fitness of GP individuals and thus, classification performance was a key factor leading the GP evolutionary process.

The performance of the proposed system was experimentally tested on 5 different binary classification problems. The performed experiments showed that LGP could reduce the dimensionality of the data but the classification performance on such transformed data sometimes could be worse than on the original dataset. In terms of the median F1 score achieved on the test set, on 2 considered problems, LGP outperformed the classifier applied directly on the original dataset in a statistically significant manner, on other 2 datasets it was outperformed, and on 1 dataset it achieved a slightly lower median F1 score on the test set but the difference was not statistically significant. Thus, the performance of the LGP is problem dependent – on some problems it might extract features that are fewer in number compared to the original features but are more useful for the classification task and allows to improve the performance of the classifier, on some other problems the layered GP-based feature extraction mechanism does not work well.

In the ideal scenario, we would expect that the final solution was obtained in the last layer, where the data were transformed into the smallest possible dimension and the performance of the classifier using such data was improved. However, for the later populations the difficulty of the task increased, and the chosen number of generations was not enough to improve the solution obtained by the first few populations. As a result, often the best solution was obtained in the first few layers. Thus, larger system size, i.e. more layers were not useful. On the other hand, the use of several layers allowed to control the size of the trees that the individuals were composed of. As in each layer a new population was initialized and it was evolved for a relatively small number of generations, this prevented the trees from growing extremely large.

One of the drawbacks of LGP is overfitting. There was a large difference in the performance of the obtained classifier on the transformed training set and the transformed test set. On the one hand, it might be the case that GP was evolving over-specialized features. On the other hand, the LGP was designed in such a way that only the classifier's performance on the training set was considered when evaluating the fitness of individuals and that might have led to the overfitting. More precisely, to obtain the fitness value, the classifier was trained and evaluated on the same set. It might be useful to have two separate subsets of training data where one subset would be used for training the classifier and another subset that would be used only for evaluating the performance of the classifier and obtaining a value that would be used as fitness of GP individual.

Overall, when designing such classification system as LGP, it is very important to choose appropriate classification algorithms as each of them has its own advantages and disadvantages which may have impact on the overall performance of the system. The classification methods employed in the system described in this work were LR, KNN, and DT. These methods were mostly chosen because of their simplicity. However, other classification method should be tested too. For instance, an ensemble method could be employed in the system. It combines predictions of several classifiers, and thus gives a more robust final prediction. On the other hand, using very sophisticated classification methods, less work would be left for GP. Moreover, in terms of computational complexity, the current system is already expensive as the classifier needs to be trained and evaluated every time the fitness of an individual is being evaluated. Using more complex classification methods which take longer to train would make the process even more computationally demanding. Due to its computational complexity the system is not suitable for highly dimensional datasets.

So far, the proposed system was tested only on binary classification problems. Applying the proposed system to multi-class classification problems is left for the future. In fact, to use LGP for multi-class classification problems, we would only need to employ a classification method that can perform multiclass classification.

6. BIBLIOGRAPHY

[1] A. F. M. Agarap, "On breast cancer detection: an application of machine learning algorithms on the wisconsin diagnostic dataset," in *ICMLSC '18 Proceedings of the 2nd International Conference on Machine Learning and Soft Computing*, 2018, pp. 5-9.

[2] E. Zafiropoulos, I. Maglogiannis and I. Anagnostopoulos, "A support vector machine approach to breast cancer diagnosis and prognosis," in *Artificial Intelligence Applications and Innovations. AIAI 2006. IFIP International Federation for Information Processing*, Springer, Boston, MA, 2006, pp. 500-507.

[3] S. A. Soliman, S. Abbas and A. B. M. Salem., "Classification of thrombosis collagen diseases based on C4.5 algorithm," in *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, 2015, pp. 131-136.

[4] W. Xu, J. Zhang, Q. Zhang and X. Wei, "Risk prediction of type II diabetes based on random forest model," in *2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, 2017, pp. 382-386.

[5] T. K. Paul and H. Iba, "Prediction of cancer class with majority voting genetic programming classifier using gene expression data," *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM,* vol. 6, no. 2, pp. 353-367, 2009.

[6] P. Sharma and U. Bhardwaj, "Machine learning based spam e-mail detection," *International Journal of Intelligent Engineering and Systems,* vol. 11, no. 3, 2018.

[7] W. A. Awad and S. M. Elseuofi, "Machine learning methods for spam e-mail classification," *International Journal of Computer Science & Information Technology (IJCSIT),* vol. 3, no. 1, pp. 173-184, 2011.

[8] N. Khare and S. Y. Sait, "Credit card fraud detection using machine learning models and collating machine learning models," *International Journal of Pure and Applied Mathematics,* vol. 118, no. 20, pp. 825-837, 2018.

[9] K. Zou, W. Sun, H. Yu and F. Liu, "ID3 decision tree in fraud detection application," *2012 International Conference on Computer Science and Electronics Engineering,* vol. 3, pp. 399-402, 2012.

[10] S. F. Sabbeh, "Machine learning techniques for customer retention: a comparative study," *International Journal of Advanced Computer Science and Applications,* vol. 9, no. 2, pp. 273-281, 2018.

[11] T. Vafeiadis, K. I. Diamantaras, G. Sarigiannidis and K. C. Chatzisavvas, "A comparison of machine learning techniques for customer churn prediction," *Simulation Modelling Practice and Theory,* vol. 55, pp. 1-9, 2015.

[12] P. N. Tan, M. Steinbach and V. Kumar, Introduction to data mining, Pearson, 2006.

[13] I. Guyon and A. Elisseeff, "An Introduction to Feature Extraction," in *Feature Extraction. Studies in Fuzziness and Soft Computing*, vol. 207, Springer, Berlin, Heidelberg, 2006, pp. 1-25.

[14] H. Motoda and H. Liu, "Feature selection, extraction and construction," *Communication of IICM,* vol. 5, no. 2, pp. 67-72, 2002.

[15] L. Guo, D. Rivero, J. Dorado, C. R. Munteanu and A. Pazos, "Automatic feature extraction using genetic programming: An application to epileptic EEG classification," *Expert Systems with Applications,* vol. 38, no. 8, pp. 10425-10436, 2011.

[16] H. M. Ebied, "Feature extraction using PCA and Kernel-PCA for face recognition," in *2012 8th International Conference on Informatics and Systems (INFOS)*, 2012.

[17] Y. Tajiri, R. Yabuwaki, T. Kitamura and S. Abe, "Feature extraction using support vector machines," in *Neural Information Processing. Models and Applications. ICONIP 2010. Lecture Notes in Computer Science*, vol. 6444, Springer, Berlin, Heidelberg, 2010, pp. 108-115.

[18] M. Pei, E. D. Goodman and W. F. Punch, "Feature extraction using genetic algorithms," in *Proceeding of International Symposium on Intelligent Data Engineering and Learning'98 (IDEAL'98)*, 1997.

[19] T. M. Mitchell, Machine Learning, McGraw-Hill, 1997.

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

[21] C. Darwin, On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life, 1859.

[22] R. Poli, W. B. Langdon and N. F. McPhee, A Field Guide to Genetic Programming (With contributions by J. R. Koza), Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.

[23] J. F. Miller and P. Thomson, "Cartesian Genetic Programming," in *Genetic Programming. EuroGP 2000. Lecture Notes in Computer Science*, vol. 1802, Springer, Berlin, Heidelberg, 2000, pp. 121-132.

[24] P. A. Whigham, "Grammatically-based genetic programming," in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, 1995.

[25] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan and M. O'Neill, "Grammar-based Genetic Programming: a survey," *Genetic Programming and Evolvable Machines,* vol. 11, no. 3-4, pp. 365-396, 2010.

[26] M. L. Wong and T. Mun, "Evolving recursive programs by using adaptive grammar based genetic programming," *Genetic Programming and Evolvable Machines,* vol. 6, no. 4, pp. 421-455, 2005.

[27] W. Fan, E. A. Fox, P. Pathak and H. Wu, "The effects of fitness functions on genetic programming-based ranking discovery for Web search," *Journal of the American Society for Information Science and Technology,* vol. 55, pp. 628-636, 2004.

[28] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Systems,* vol. 9, pp. 193- 212, 1995.

[29] K. Neshatian, M. Zhang and M. Johnston, "Feature construction and dimension reduction using genetic programming," in *AI 2007: Advances in Artificial Intelligence. AI 2007. Lecture Notes in Computer Science*, vol. 4830, Springer, Berlin, Heidelberg, 2007, pp. 160-170.

[30] M. L. Raymer, W. F. Punch, E. D. Goodman and L. A. Kuhn, "Genetic programming for improved data mining: application to the biochemistry of protein interactions," in *Proceedings of the 1st Annual Conference on Genetic Programming*, Cambridge, Massachusetts, MIT Press, 1996, pp. 375-380.

[31] M. C. J. Bot, "Feature extraction for the K-Nerest Neighbors Classifier with Genetic Programming," in *Genetic Programming. EuroGP 2001. Lecture Notes in Computer Science*, vol. 2038, Springer, Berlin, Heidelberg, 2001, pp. 256-267.

[32] J. Sherrah, "Automatic feature extraction for pattern recognition," *Ph.D. Thesis, The University of Adelaide,* 1998.

[33] M. Kotani, M. Nakai and K. Akazawa, "Feature extraction using evolutionary computation," *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99,* vol. 2, p. 1230–1236, 1999.

[34] K. Krawiec, "Genetic programming-based construction of features for machine learning and knowledge discovery tasks," *Genetic Programming and Evolvable Machines,* vol. 3, no. 4, pp. 329-343, 2002.

[35] B. Bhanu and K. Krawiec, "Coevolutionary construction of features for transformation of representation in machine learning," in *Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIPWM'03 Conference held in Zakopane, Poland, June 2-5, 2003*, Springer, Berlin, Heidelberg, 2004, pp. 139-150.

[36] H. Guo, L. B. Jack and A. K. Nandi, "Feature Generation Using Genetic Programming With Application to Fault Classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B,* vol. 35, no. 1, pp. 89-99, 2005.

[37] H. Guo and A. K. Nandi, "Breast cancer diagnosis using genetic programming generated feature," *Pattern Recognition,* vol. 39, no. 5, pp. 980-987, 2006.

[38] H. Firpi, E. Goodman and J. Echauz, "Genetic programming artificial features with applications to epileptic seizure prediction," *IEEE-EMBS 2005, 27th Annual International Conference on Engineering in Medicine and Biology Society,* pp. 4510-4513, 2005.

[39] M. G. Smith and L. Bull, "Genetic Programming with a Genetic Algorithm for feature construction and selection," *Genetic Programming and Evolvable Machines,* vol. 6, no. 3, p. 265–281, 2005.

[40] F. E. B. Otero, M. M. S. Silva, A. A. Freitas and J. C. Nievola, "Genetic programming for attribute construction in data mining," in *Genetic Programming. EuroGP 2003. Lecture Notes in Computer Science*, vol. 2610, Springer, Berlin, Heidelberg, 2003, pp. 384-393.

[41] S. Afzali, H. Al-Sahaf, B. Xue, C. Hollitt and M. Zhang, "Genetic programming for feature selection and feature combination in salient object detection," in *Theory and Applications of Models of Computation*, 2019, pp. 308-324.

[42] Y. Bi, B. Xue and M. Zhang, "An automatic feature extraction approach to image classification using genetic programming," in *Applications of Evolutionary Computation. EvoApplications 2018. Lecture Notes in Computer Science*, vol. 10784, Springer, Cham, 2018, pp. 421-438.

[43] B. Tran, B. Xue and M. Zhang, "Genetic programming for feature construction and selection in classification on high-dimensional data," *Memetic Computing,* vol. 8, no. 1, p. 3–15, 2016.

[44] M. Muharram and G. D. Smith, "Evolutionary Constructive Induction," *IEEE Transactions on Knowledge and Data Engineering,* vol. 17, no. 11, pp. 1518 - 1528, 2005.

[45] S. Ahmed, M. Zhang, L. Peng and B. Xue, "Multiple feature construction for effective biomarker identification and classification using genetic programming," in *GECCO'14 Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 249-256.

[46] X. Tan, B. Bhanu and Y. Lin, "Fingerprint classification based on learned features," *IEEE Transactions On Systems, Man, And Cybernetics—Part C: Applications And Reviews,* vol. 35, no. 3, pp. 287 - 300, 2005.

[47] J. Huertas, J. Rodríguez-Benítez, D. Pozo, R. Aler and I. M. Galván, "Genetic programming to extract features from the whole-sky camera for cloud type classification," *Renewable Energy and Power Quality Journal,* vol. 1, no. 15, pp. 132-136, 2017.

[48] A. Heinle, A. Macke and A. Srivastav, "Automatic cloud classification of whole sky images," *Atmospheric Measurement Techniques,* no. 3, p. 557–567, 2010.

[49] M. W. Aslam, Z. Zhu and A. K. Nandi, "Feature generation using genetic programming with comparative partner selection for diabetes classification," *Expert Systems with Applications,* vol. 40, no. 13, p. 5402–5412, 2013.

[50] M. Sokolova, N. Japkowicz and S. Szpakowicz, "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation," in *AI 2006: Advances in Artificial Intelligence, 19th Australian Joint Conference on Artificial Intelligence, Hobart, Australia, December 4-8, 2006, Proceedings*, 2006, pp. 1015-1021.

[51] I. D. Dinov, "Decision Tree Divide and Conquer Classification," in *Data Science and Predictive Analytics*, Springer, Cham, 2018, pp. 307-343.

[52] T. Hastie, R. Tibshirani and J. Friedman, The elements of statistical learning: data mining, inference, and prediction, Springer, 2009.

[53] G. James, D. Witten, T. Hastie and R. Tibshirani, An introduction to statistical learning with application in R, Springer, 2013.

[54] Z. Yao and W. L. Ruzzo , "A Regression-based K nearest neighbor algorithm for gene function prediction from heterogeneous data," *BMC Bioinformatics,* vol. 7, no. S11, 2006.

[55] G. H. Chen and D. Shah, "Explaining the Success of Nearest Neighbor Methods in Prediction," *Foundations and Trends in Machine Learning,* vol. 10, no. 5-6, pp. 337-588, 2018.

[56] N. Kouiroukidis and G. Evangelidis, "The effects of dimensionality curse in high dimensional KNN search," *2011 15th Panhellenic Conference on Informatics,* 2011.

[57] M. Kuhn and K. Johnson, Applied predictive modeling, Springer, 2013.

[58] S. Domínguez-Almendros, N. Benítez-Parejo and A. R. Gonzalez-Ramirez, "Logistic regression models," *Allergologia et Immunopathologia,* vol. 39, no. 5, pp. 295-305, 2011.

[59] H. Deng, Y. Sun, Y. Chang and J. Han, "Probabilistic Models for Classification," in *Data classification: algorithms and applications*, Chapman & Hall/CRC, 2014.

[60] D. E. Goldberg, Genetic algorithms in search, optimization and machine learning, Addison-Wesley Publishing Company, 1989.

[61] QSAR biodegradation Data Set. Available:
https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation#. [Accessed 5 11 2018].

[62] Connectionist Bench (Sonar, Mines vs. Rocks) Data Set. Available:
https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar,+Mines+vs.+Rocks%29
[Accessed 5 11 2018].

[63] Breast Cancer Wisconsin (Diagnostic) Data Set. Available:
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29.
[Accessed 5 11 2018].

[64] Ionosphere Data Set. Available: https://archive.ics.uci.edu/ml/datasets/ionosphere. [Accessed 5 11 2018].

[65] Credit Approval Data Set. Available: https://archive.ics.uci.edu/ml/datasets/Credit+Approval. [Accessed 5 11 2918].

[66] D. Dua and C. Graff, "UCI Machine Learning Repository," Irvine, CA: University of California, School of Information and Computer Science, 2019. Available: http://archive.ics.uci.edu/ml.

[67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg and and others, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[68] N. J. Nilsson, Introduction to machine learning, 1996.

[69] A. B. Hassanat, M. A. Abbadi and G. A. Altarawneh, "Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach," *International Journal of Computer Science and Information Security,* vol. 12, no. 8, August 2014.

[70] B. Lantz, Machine learning with R, Birmingham: Packt Publishing, 2013.

[71] K. H. Rosen, Ed., Handbook of Discrete and Combinatorial Mathematics, 2nd ed., Chapman and Hall/CRC, 2017.

[72] T. Stephens, "gplearn: Genetic Programming in Python," 2018. Available: https://github.com/trevorstephens/gplearn. [Accessed 15 09 2018].