On genetic programming representations and fitness functions for interpretable dimensionality reduction

Thomas Uriot
Leiden University Medical Center
Leiden, The Netherlands

Marco Virgolin
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands

Tanja Alderliesten
Leiden University Medical Center
Leiden, The Netherlands

Peter A.N. Bosman
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands

ABSTRACT

Dimensionality reduction (DR) is an important technique for data exploration and knowledge discovery. However, most of the main DR methods are either linear (e.g., PCA), do not provide an explicit mapping between the original data and its lower-dimensional representation (e.g., MDS, t-SNE, isomap), or produce mappings that cannot be easily interpreted (e.g., kernel PCA, neural-based autoencoder). Recently, genetic programming (GP) has been used to evolve interpretable DR mappings in the form of symbolic expressions. There exists a number of ways in which GP can be used to this end and no study exists that performs a comparison. In this paper, we fill this gap by comparing existing GP methods as well as devising new ones. We evaluate our methods on several benchmark datasets based on predictive accuracy and on how well the original features can be reconstructed using the lower-dimensional representation only. Finally, we qualitatively assess the resulting expressions and their complexity. We find that various GP methods can be competitive with state-of-the-art DR algorithms and that they have the potential to produce interpretable DR mappings.

CCS CONCEPTS

• **Computing methodologies** → *Genetic programming*; **Dimensionality reduction and manifold learning**.

KEYWORDS

Dimensionality reduction, genetic programming, interpretability, unsupervised learning

ACM Reference Format:
Thomas Uriot, Marco Virgolin, Tanja Alderliesten, and Peter A.N. Bosman. 2022. On genetic programming representations and fitness functions for interpretable dimensionality reduction. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3512290.3528849

GECCO '22, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9237-2/22/07…$15.00
https://doi.org/10.1145/3512290.3528849

1 INTRODUCTION

Dimensionality reduction (DR) is a key instrument for knowledge discovery [8, 11]. Informally, we can define DR (see Section 2.1 for a formal definition) to be the area of study concerned with finding lower dimensional representations of the original data, such that some meaningful properties of the original dataset are preserved (e.g., distance between points, original variance) . It is based on the fact that most real-world datasets only have artificially high dimensionality due to, for instance, interdependent or noisy features. Current state-of-the-art non-linear DR algorithms act as a black-box when going from high to low dimensions (e.g., t-SNE [21], isomap [25], LLE [25]), and do not provide any means to inspect the resulting lower dimensions in terms of the original variables, i.e., there is no functional mapping that describes how the original dimensions are compressed into the lower-dimensional (or also called latent) representation. While other methods, such as kernel PCA [37] and neural autoencoders provide a functional mapping between the original data and its lower dimensional representation, these mappings are not interpretable. However, having an interpretable mapping can be particularly important in the absence of meaningful labels or classes, as it becomes even more challenging to assess and inspect whether a lower-dimensional representation produced by a DR model is meaningful and trustworthy. In the context of this paper, we consider a model to be interpretable if a human can carry out and verify the computations by which the original dimensions are shrunk into the lower-dimensional representation in a reasonable amount of time (as suggested in [39]).

We believe that interpretable DR models are of primary importance as DR is often used for data exploration to draw insights from the data. In addition, human-interpretable models allow to assess whether or not the compression the mapping performs is reasonable and, e.g., safe to use in high-stakes applications of machine learning [36].

Recently, the use of genetic programming [14] (GP) has been an increasingly popular way to produce interpretable models. For instance, GP has been used to learn programs encoded by simple symbolic expression representing physical laws [4, 9]. Furthermore, recent works have investigated the use of such symbolic expressions produced by GP as interpretable models [7, 17, 31, 33]. In this work, we leverage the interpretability potential of GP and apply it to non-linear DR.

GP has been shown to be a promising avenue to produce interpretable lower-dimensional representations in several recent works [17, 18, 20]. However, more representations and evaluation

schemes can be imagined, but no work has been done to provide a comparison of these different options. We attempt to fill this gap by designing new GP-based methods for DR and establish a robust evaluation protocol across two objectives (predictive power of the newly constructed features and their ability to reconstruct the original input). The main contributions of this paper are threefold: (i) we investigate how GP can best be used for DR by considering standard DR loss functions and GP representations (see Section 3), (ii) additionally, we perform experiments with several DR loss functions that have previously never been applied to GP, such as weighted rank-correlation, using a neural-based autoencoder teacher, or directly using a GP-based autoencoder to reconstruct the input, and (iii), we show that our GP-based methods are able to learn non-linear and concise formulas of the lower-dimensional latent space. We benchmark our methods against three mainstream DR algorithms: principal components analysis [10] (PCA), locally linear embedding [25] (LLE), and isomap [29].

2 BACKGROUND

2.1 Dimensionality Reduction

The goal of DR is to find a lower-dimensional representation of the original data such that some properties of the original data, such as total variance (e.g., PCA), distances between points (e.g., isomap), ability to reconstruct the original features (e.g., autoencoders) are preserved. Formally, let us denote the original dataset by $X \in \mathbb{R}^{n \times p}$ where n and p represent the number of instances and features, respectively. Then, the goal is to find another representation $\widetilde{X} \in \mathbb{R}^{n \times k}$ of X, such that $k \ll p$, and such that the new representation retains some meaningful properties of the original data. Note that throughout this paper, we will refer to X^j and $\widetilde{X}^j$ as the j^{th} dimension (feature), and x_i and $\tilde{x}_i$ as the i^{th} observation of the original and lower-dimensional representation, respectively.

2.2 Genetic Programming for Dimensionality Reduction

Recently, DR using GP has gained traction and has started being more thoroughly explored. In [17], the authors use GP to perform DR by using a fitness function that encourages the preservation of local neighborhoods. In [20], the authors build on their previous work [17] by taking a bi-objective approach where the first objective encourages to preserve local structure, while the second objective represents the number of dimensions in the latent representation. However, both in [17] and [20], there is no constraint imposed on the complexity of the evolved expressions, potentially rendering them non-interpretable. This issue is addressed in [18] where the authors use a bi-objective approach with the first objective being the cost function used in the t-SNE algorithm [21] and the second objective being a proxy for complexity. However, t-SNE is devised to strictly preserve local neighborhoods and does not result in a functional mapping, thus it is unclear whether the cost function of t-SNE is well-suited to be used for GP.

In addition, GP has recently been used to perform feature extraction and transfer learning by constructing non-linear combinations of features to then be used in subsequent supervised learning tasks [16, 31, 38]. This type of application requires to specify a predictive task with ground truth labels. However, we may be interested in discovering intrinsic relationships between the variables or find an interpretable functional mapping for the compression of the original dataset without necessarily having an associated predictive task. One way to tackle the aforementioned problem was explored in [27], where the authors used GP to predict the value of each feature by using the remaining ones as input. However, this process does not belong to the realm of DR since the number of features remains the same.

3 METHODOLOGY

In GP, the main way to represent symbolic expressions is by defining it as a computational tree [24]. In this paper, we follow the literature and use trees to represent our GP programs. A detailed description of the tree-based representations we consider is given in the following subsection.

3.1 Computational Tree Representations

3.1.1 Single tree. The single-tree representation, as shown in Figure 1a, is one of the simplest way to represent a symbolic expression. In the single-tree representation, a single tree is responsible to output a unique single value. Formally, if we denote a single tree by m, we have that $m_j : \mathbb{R}^p \rightarrow \mathbb{R}$, for $j = 1, \ldots, k$. In words, each tree maps the original dimensions (the terminals at the leaves) to a single latent dimension (the root of the tree).

This representation is limited when it comes to modeling multi-output functions which is the case in DR, if $k > 1$. Indeed, each latent dimension is evolved separately (see Figure 1a). Therefore, this representation cannot efficiently model non-separable fitness functions, where variables are dependent and should thus be jointly optimized, as is the case in DR.

3.1.2 Classic multi-tree. Similarly to [17, 18, 20], we use a multi-tree representation as illustrated in Figure 1b. Each tree of the multi-tree takes the original data as input and outputs a single dimension of the lower-dimensional representation. During evolution, the trees of the multi-tree evolve simultaneously, different from single-tree representation, where each tree evolves separately during independent runs. This representation has two main advantages: (i) it only requires a single evolution run, (ii) it is able to learn non-separable fitness functions, where the variables are dependent, which is the case in cost functions that are typically used in DR.

In this paper, we will denote the multi-tree representation by $\mathcal{MT}(\cdot) = \{m_j(\cdot)\}_{j=1}^{k}$, where m_j represents the j^{th} tree of the multi-tree and k is the number of dimensions in the lower-dimensional representation.

3.1.3 Autoencoder multi-tree. In addition to the classic multi-tree, we propose a new representation for DR (inspired from automatically defined functions [13]), composed of two classic multi-trees, with one multi-tree acting as encoder and one as decoder. Specifically, the encoder is a multi-tree composed of k trees and is responsible to map the original input to its lower-dimensional representation. On the other hand, the decoder is a multi-tree composed of p trees where p is the number of original features. It is responsible to map the output of the encoder to reconstruct the original

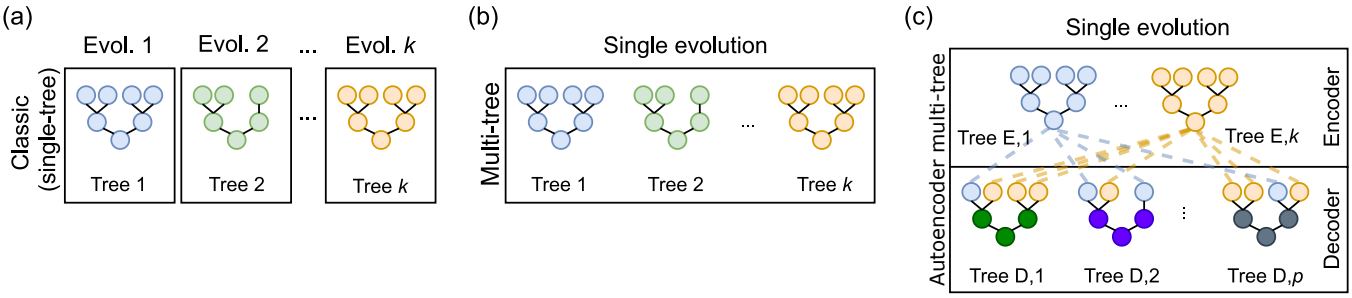

Figure 1: Three ways of mapping the original data to its lower-dimensional representation using computational trees. Note that the trees are read top-down, from leaves to output. (a) Classic: A GP individual is represented by a single tree. Each tree maps the input to a unique latent dimension, and is evolved separately. It thus requires k independent runs. (b) Multi-tree: A GP individual is a collection of k trees that are evolved jointly in a single evolution run. (c) Autoencoder multi-tree multi-tree: A GP individual is made of two parts: the encoder and the decoder. The encoder is a multi-tree composed of k trees and is responsible to map the original input to its lower-dimensional representation. The decoder is a multi-tree composed of p trees, where p is the number of original features, and is responsible to map the output of the encoder to reconstruct the original features. The encoder and the decoder evolve jointly.

features. The encoder and the decoder are jointly evolved in a single run. Formally, let us denote the encoder and decoder of the autoencoder multi-tree by $\mathcal{AMT}^{(e)}$ and $\mathcal{AMT}^{(d)}$ respectively, then we have that $\mathcal{AMT}^{(e)}(\cdot) = \{m_j^{(e)}(\cdot)\}_{j=1}^{k}$ and $\mathcal{AMT}^{(d)}(\cdot) = \{m_j^{(d)}(\cdot)\}_{j=1}^{p}$, where $m_j^{(e)} : \mathbb{R}^p \to \mathbb{R}$ and $m_j^{(d)} : \mathbb{R}^k \to \mathbb{R}$.

The purpose of the autoencoder multi-tree is to test whether a fully GP-based approach, i.e., one where both the encoder and decoder are evolved at the same time, is capable of performing DR effectively, as explained in the following subsection.

3.2 Fitness Functions

We investigate several fitness functions, denoted by $\mathcal{F}$, to guide the evolutionary process of the multi-tree GP. As mentioned in Section 2.1, we denote the lower-dimensional representation of the original dataset X by $\widetilde{X}$, where the i^{th} data point is given by $\widetilde{x}_i = \mathcal{MT}(x_i)$, $i = 1, \ldots, n$, for a given multi-tree individual $\mathcal{MT}$. Furthermore, we denote the distance matrices of the original dataset and its lower-dimensional representation by D and $\widetilde{D}$, respectively, where $D_{i,j} = d_{i,j}$ and $\widetilde{D}_{i,j} = \widetilde{d}_{i,j}$. For the distance preserving and rank preserving fitness functions, we investigate two distance metrics: (i) Euclidean distance, and, (ii) geodesic distance. The geodesic distance is particularly suited to represent the local low-dimensional geometry of the data manifold [29] that the Euclidean distance may not be able to capture.

Formally, for a fitness function denoted by $\mathcal{F}(X, \cdot)$, we seek $\mathcal{MT}^*$ such that:

$$\mathcal{MT}^* = \underset{\mathcal{MT}}{\operatorname{argmin}}\, \mathcal{F}(X, \cdot). \tag{1}$$

Note that for the GP-based autoencoder fitness function, we use an autoencoder multi-tree and the best individual would thus be denoted by $\mathcal{AMT}^*$.

3.2.1 Distance preserving. This fitness function is based on preserving distances between pair of instances when mapping the original data to its lower-dimensional representation. To this end, we use Sammon mapping [26] which aims to minimize the following:

$$\mathcal{F}_{\text{dist}} = \mathcal{F}(X, \widetilde{X}) = \frac{1}{\sum_{i<j} d_{i,j}} \sum_{i<j} \frac{(d_{i,j} - \widetilde{d}_{i,j})^2}{d_{i,j}}. \tag{2}$$

The effect of $d_{i,j}$ in the denominator is to put emphasis on points that are close to each-other in the original space. In other words, we give more importance in preserving local structure than global.

3.2.2 Rank preserving. This fitness function is similar to preserving distances except we are only interested in the ranks of the distances. Essentially, this is a measure of the monotonicity of the relationship between the distances in the original and lower-dimensional representations. This has the advantage of being more flexible, less sensitive to outliers (i.e., large distances) and thus more easily learnable than the distance preserving fitness. In [20], the authors use Spearman's rank correlation coefficient [35] as the fitness function to perform dimensionality reduction using GP. In this paper, we use Kendall's τ which is preferred to Spearman's rank correlation coefficient for smaller samples [12]. Our aim is to maximize Kendall's τ:

$$\tau = \frac{n_c - n_d}{\frac{N}{2}(N-1)}, \tag{3}$$

where n_c is the number of *concordant pairs*, n_d is the number of *discordant pairs*, and N is the total number of pairs. Let us denote the i^{th} row of the distance matrices $\widetilde{D}$ and D by $\widetilde{d}_i$ and d_i respectively. Then, we have that a pair is concordant if $\text{sgn}(d_{i,j} - d_{i,l}) = \text{sgn}(\widetilde{d}_{i,j} - \widetilde{d}_{i,l})$, and discordant if $\text{sgn}(d_{i,j} - d_{i,l}) = -\text{sgn}(\widetilde{d}_{i,j} - \widetilde{d}_{i,l})$, for $j \neq l$ and $\text{sgn}(\cdot)$ being the standard signum function.

Taking the average across instances, the final fitness function (to be minimized) is defined as:

$$\mathcal{F}_{\text{rank}} = \mathcal{F}(X, \widetilde{X}) = -\frac{1}{n}\sum_{i=1}^{n} \tau_i = -\frac{1}{n}\sum_{i=1}^{n} \frac{n_{c_i} - n_{d_i}}{\frac{N_i}{2}(N_i - 1)}, \tag{4}$$

where n_{c_i}, n_{d_i}, and N_i are computed using d_i and $\widetilde{d}_i$.

However, this approach fails to take into account that discordances between observations with high rank (a rank of 0 being the highest rank) are more important than those between items with low rank. This is because we want to emphasize preserving local structure over global structure, and so, if the distance between two points in the original space is large (i.e. the rank is low), we want it to have a lesser importance in the final fitness function. This serves similar purpose to the denominator in the Sammon mapping in Equation (2).

Therefore, we use a weighted version of Kendall's τ proposed in [30], where the weighting is defined by ranking the original distances and assigning the highest rank of 0 to the shortest distance, and so on. Then, a weighting function assigns a weight w depending on the rank r. In this paper, we use a hyperbolic weighting function, and thus, we have that $w = \frac{1}{r+1}$.

3.2.3 Neural-based autoencoder teacher. This fitness function is inspired by the student-teacher literature, where a (usually) simple model (e.g., linear regression, low-depth trees, symbolic regression) is trained to approximate the output of another (complex) model. This is a two-step process where we first train a base model and then train the surrogate model on the output (or hidden layer) produced by the base model.

In our case, let us denote the latent layer of the neural-based autoencoder by $\mathcal{L} = \{l_j\}_{j=1}^{k}$, where l_j represents the j^{th} neuron and k is the number of latent dimensions (e.g., the number of neurons in the latent layer). Our fitness function is then given by:

$$\mathcal{F}_{\text{AE}} = \mathcal{F}(\mathcal{X}, \widetilde{\mathcal{X}}) = \frac{1}{nk} \sum_{i=1}^{n} \sum_{j=1}^{k} (l_j(x_i) - \widetilde{x}_{i,j})^2, \tag{5}$$

where $\widetilde{x}_{i,j} = m_j(x_i)$ is the output of the j^{th} tree of the multi-tree $\mathcal{MT} = \{m_j\}_{j=1}^{k}$. In words, the fitness function is the mean-squared-error between the output of the multi-tree and the latent layer of the autoencoder.

3.2.4 GP-based autoencoder. This fitness function is similar to the neural-based autoencoder teacher fitness function in that it also makes use of an autoencoder. However, instead of evolving a classic multi-tree GP to match the latent layer of a neural autoencoder teacher, we let GP evolve a complete autoencoder, using the representation shown in Figure 1c. As mentioned, the GP-based autoencoder is composed of two multi-trees representing the encoder and the decoder of the autoencoder. In this case, the lower-dimensional representation $\widetilde{X}$ of the original dataset $\mathcal{X}$ is given by the output of the encoding multi-tree. Formally, using the notation introduced in 3.1.3, we have that $\widetilde{x}_i = \mathcal{AMT}(x_i)^{(e)}$, $i = 1, \ldots, n$, and the fitness function is given by:

$$\mathcal{F}_{\text{GP}} = \mathcal{F}(\mathcal{X}, \widehat{\mathcal{X}}) = \frac{1}{np} \sum_{i=1}^{n} \sum_{j=1}^{p} (x_i - \widehat{x}_{i,j})^2, \tag{6}$$

where $\widehat{X}_{i,j} = \widehat{x}_{i,j} = m_j^{(d)}(x_i)$ is the output of the j^{th} tree of the GP's decoder $\mathcal{AMT}^{(d)}$. In other words, the cost function is the reconstruction error between the output of the decoder and the original data, and the resulting lower-dimensional representation is given by the output of the encoder.

3.3 Baseline Methods

We compare the performance of the GP-based methods with three dimensionality reduction baselines: principal components analysis (PCA), locally linear embedding (LLE), and isomap. These methods are chosen because they learn a mapping on the training data that can be applied to unseen data, which is not possible with other methods such as t-SNE or multi-dimensional scaling (MDS). This enables a fair comparison since our GP-methods are trained to perform dimensionality reduction, and the evaluation is done on unseen data, as depicted in Figure 2. In addition, the baselines are heterogeneous as PCA seeks a global orthogonal linear transformation of the data, while LLE is a local and non-linear method, and isomap seeks to preserve the estimated intrinsic geometry of the data manifold using geodesic distance.

3.4 Performance Metrics

In this section, we discuss the choices made to evaluate the performance of the various fitness functions in guiding the evolution process of GP. This amounts to evaluating the quality of the resulting lower-dimensional representation of the original data. This is still an open problem in data science and will depend on the application and the questions that one wants to answer during the analysis. In addition, the fitness function being optimized during the dimensionality reduction process may be biased towards our evaluation metric or even be misaligned with it.

3.4.1 Predictive performance. In some situations, there exists a metric that truly represents our end goal, such as predictive performance on a supervised downstream task. If there exists such an objective (task) for a given dataset, then we do not have to worry whether the metric (e.g., classification accuracy or regression error) is biased towards any of our GP methods since it is a metric that we are *objectively* interested in. Using the predictive power of the low-dimensional representation on downstream supervised tasks to evaluate dimensionality reduction algorithms has been done in recent studies [17, 18, 20].

However, there are three main drawbacks with this approach. Firstly, we have to assume that the response (e.g., class structure for classification or target distribution for regression) is a significant factor within the manifold structure. Arguably, this assumption holds for the majority of curated datasets, and datasets for which supervised learning algorithms can perform well (i.e., one can learn labels from the data). Secondly, there may be some structure in the data that is not captured by the labels but that may still be of interest for knowledge discovery, and thus using predictive performance as a proxy would be insufficient. Finally, and perhaps the most critical drawback is that we may have datasets on which we want to perform dimensionality reduction but that have no labels.

In this work, we use balanced accuracy [3] as a performance metric. In particular, we follow [17, 20] and use a random forest classifier as evaluation model for accuracy (denoted by E in Section 4.4), trained on the low-dimensional representation to predict the labels and compute the predictive performance.

3.4.2 Reconstruction error. Besides accuracy, we also consider the reconstruction error, as an additional metric, and, in this case, we use a neural-based decoder as the evaluation model. Note that, however, the reconstruction error is biased to favor the methods using the neural-based autoencoder teacher ($\mathcal{F}_{\text{AE}}$) and the GP-based autoencoder ($\mathcal{F}_{\text{GP}}$) fitness functions since they both minimize the reconstruction error during optimization.

4 EXPERIMENTAL SETUP

4.1 Datasets

We test our methods on 5 datasets from the UCI repository [6] for which a brief description is provided in Table 1. Note that for the purpose of our analysis, we only keep the continuous attributes, which is why the number of features is different from the original one for datasets that contain categorical variables. This is because we only use arithmetic operators and do not use any conditional or logical operators (e.g., *if*, *and*, *or*) in our GP representation (see Section 4.2). In addition, some of our fitness functions are based on the Euclidean distance, which would not be appropriate for a mix of continuous and categorical variables.

Table 1: Datasets considered, where n, d, and c represent the number of instances, features, and classes, respectively.

Name	n	d	c
Ionosphere	350	34	2
German Credit	1000	24	2
Libras Movement	359	90	15
Segmentation	2310	19	7
Telescope (MGT)	19019	10	2

4.1.1 Principal component initialization. Note that the fitness functions described in Section 3.2 make use of all the variables from the original data. This happens either at the time of computing the distance matrices or when computing the reconstruction error. In doing so, we may be using noisy variables as well as redundant variables (e.g., co-linear variables) which could be dominating other important variables during the optimization process. A simple and well established solution is to perform PCA before applying any sort of analysis that requires to compute distances between points, such as clustering analysis [2, 5]. Thus, in this paper, we transform the original dataset using PCA (retaining 99% of the original variance) to compute the fitness functions in Section 3.2. This amounts to replacing $\mathcal{X}$ with $\mathcal{X}_{\text{pca}}$ in $\mathcal{F}(\mathcal{X}, \cdot)$, where $\mathcal{X}_{\text{pca}}$ is the transformation of $\mathcal{X}$ after applying PCA and keeping a sufficient number of principal components to account for 99% of the original variance. Note that in order to keep the GP expressions interpretable in terms of the original input, the input to the multi-tree GP is still the original data $\mathcal{X}$, as opposed to the PCA transformed data $\mathcal{X}_{\text{pca}}$.

4.2 Genetic Programming Parameters

We use standard GP parameter settings, as shown in Table 2. A basic tournament selection [24] process is used to choose the individuals that will act as parents to produce offspring via the variation operators. Each parent can be subject to up to three different genetic operators: (i) *crossover*, (ii) *subtree mutation*, and (iii) *one-point mutation*, with probabilities of p_c, p_s, and p_o, respectively. These can be easily experimented with, using our publicly available code[1]. Note that the genetic operators, maximum tree size, and tree depth are applied to single trees within a multi-tree. This means that, for instance, each tree within a multi-tree has a p_s probability of having a subtree mutation. For the crossover operator, where subtrees are exchanged between two trees, one has to decide whether cross-pollination between trees at different index from the two multi-tree individuals is allowed. In [19], the authors randomly select two trees, one from each individual, and perform standard crossover between them (random-index crossover), while in [1], the authors only allow trees at the same position in the multi-tree to crossover with each other (same-index crossover). Here, we use same-index crossover when mixing trees from different multi-trees. The reason we make this choice is that same-index crossover encourages trees at the same index to specialize [1] and converge towards learning the same latent dimension of the lower-dimensional representation.

Finally, the population is initialized using the popular ramped half-and-half method [14], and ephemeral constants are drawn from a $\mathcal{N}(0, 1)$. The terminal set (i.e., the set of all possible leaf nodes) is $\mathcal{T} = \{X, \mathcal{R}\}$, where $\mathcal{R}$ represents the set of random constants and $X = \{X^j\}_{j=1}^{p}$ is the set of variables. The function set (i.e., the set of possible non-leaf nodes) is $\mathcal{F} = \{-, +, \times\}$. This means that GP will evolve polynomials, with arbitrarily complex interactions only bounded by the depth of the tree. This results in simple expressions that can be used as a starting point to further interpretability studies on performing DR with GP. Furthermore, we conducted the same experiments using the extended function set $\mathcal{F}' = \{-, +, \times, \cos, \log_p\}$, where $\log(\cdot)_p = \log(|\cdot| + \varepsilon)$, but obtained significantly worse results than when using $\mathcal{F}$. Due to space limitations, the results obtained with $\mathcal{F}'$ are not presented here but can be found in the code repository.

Table 2: GP parameters and their settings.

Parameter	Value
Crossover rate (p_c)	0.8
Subtree mutation rate (p_s)	0.2
Operator muration rate (p_o)	0.2
Population size (P)	1000
Generations	100
Tree depth: (Min, Max)	(2, 7)
Tournament Size	7

4.3 Mini-batch Training

Note that during the evolutionary process, for each generation, we have to compute the distance matrix $\widetilde{D}$ for all the multi-tree individuals in the population, which has an overall computational cost of $O(kn^2 \times P)$, where P is the population size used by GP, n is the number of data points, and k the number of latent variables of the lower-dimensional representation. For large values of n, this can be prohibitively slow. Therefore, at each generation, we randomly

[1] https://github.com/pinouche/gp_dr

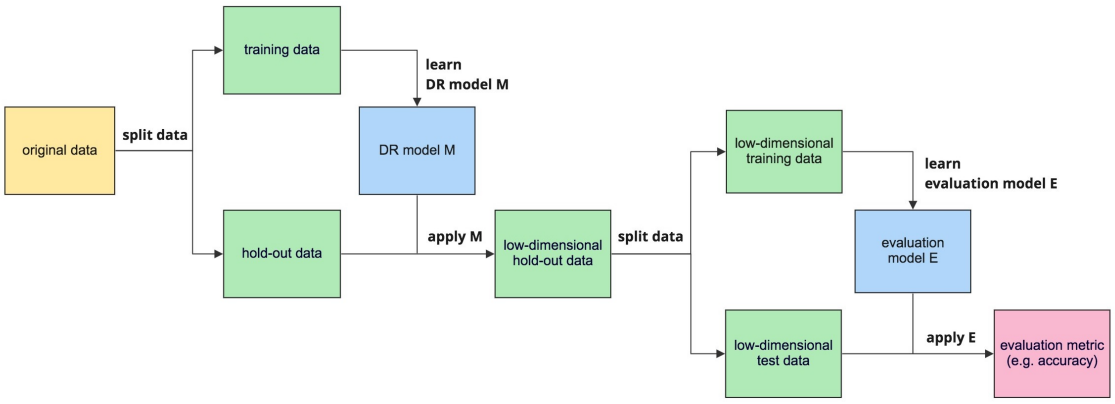

Figure 2: Evaluation protocol for our DR pipeline. First, we learn a DR model M on a subset of the original data. This model is then applied on the held-out subset of the original data to obtain a lower-dimensional representation. We then learn an evaluation model E on a subset of the lower-dimensional data and apply it to the held-out subset in order to compute the final performance metric.

sample a mini-batch of $b < n$ data points from the data and compute the distance matrices on that mini-batch. In addition, mini-batch training is thought to improve generalization by changing the loss landscape at each batch, and thus escaping local optima [23].

4.4 Evaluation Protocol

A detailed account of the overall evaluation process, from the original data to the final predictions, is given in Figure 2. First, the original data is split into two subsets: (i) the training data for the DR model M, and, (ii) a held-out set to apply M on and obtain its lower-dimensional representation. In turn, this lower-dimensional data is split in two subsets: (i) the training data for the evaluation model E, and, (ii) the hold-out set to apply the evaluation model E to compute the performance metrics. The evaluation model E (e.g., a neural decoder for reconstruction error and a random forest classifier for accuracy) is trained using a 10-fold cross validation and we report the results on the test set (see Section 5). Note that the models used for the evaluation phase are described in Section 3.4.

5 RESULTS

5.1 Quantitative Analysis

In Table 3 (a), we display the average balanced accuracy and associated standard deviation computed across 30 independent runs for all the GP-based methods and the baselines. Similarly, in Table 3 (b), we display the results for the reconstruction error. In addition, we compute statistical significance using the Mann-Whitney U rank test [22], denoted by the asterisk notation.

We can see that the non-linear baseline methods LLE and isomap seem to yield the better performances, in particular when $k = 2$. For $k = 3$, $\mathcal{F}_{\text{dist}}$ (Euclidean) performs best on 2 of the datasets but is not always statistically significant. Overall, it seems that for $k = 3$, the difference between the methods is less pronounced than for $k = 2$. This is intuitive since the more dimensions we keep the easier it is to capture relevant information from the original data, thus attenuating potential differences between methods. In fact, for more complex datasets with a higher number of features, such as Ionosphere and Libras Movement, we can still see a significant difference between the best performing method (isomap) and the other methods. Additionally, PCA (the only linear dimensionality reduction method in our experiments) does not perform well on these two datasets, suggesting that non-linearity allows for capturing relevant relationships between the original variables.

In Table 3 (b), we can see that $\mathcal{F}_{\text{AE}}$ and $\mathcal{F}_{\text{GP}}$ have the lowest reconstruction error out of the GP-based methods. This is expected because these methods minimize the reconstruction error during the optimization process. In addition, we can see that PCA and isomap yield good results across several datasets. Interestingly, while LLE performs well when balanced accuracy is used as metric, it is arguably the worst performing method to reconstruct the original data. This may be explained by the fact that co-linear features are equally weighted when computing the distance matrices D and $\widetilde{D}$. This is detrimental to maximizing accuracy as there may be independent variables (e.g., non co-linear) that are important for the downstream classification task but which would be outweighted when computing the distance matrices. For this reason, we performed PCA before computing distances, as mentioned in Section 4.1.1. However, we only experimented with keeping 99% of the original variance and one would need to experiment further to identify the effect of performing PCA on the balanced accuracy and reconstruction error. Finally, while $\mathcal{F}_{\text{AE}}$ and $\mathcal{F}_{\text{GP}}$ do perform well, there exists drawbacks to these methods. For $\mathcal{F}_{\text{AE}}$ we have to train a neural network autoencoder, which adds a layer of complexity to the overall method. That is, if our autoencoder teacher does not perform well in the first place, the GP-based student is bound to fail. On the other hand, while $\mathcal{F}_{\text{GP}}$ does not require an autoencoder teacher model, it does not scale well with the number of input

Table 3: Score of (a) balanced accuracy and (b) reconstruction error (mean ± standard deviation of 30 runs). For a confidence level of $1-\alpha$, statistical significance is denoted as follows: *,**,***, for $\alpha = 0.1, 0.05, 0.01$, respectively. Statistical testing is performed between the best method and the remaining 8 methods. If the mean performance is equal across several methods, the method with the lowest standard deviation is chosen as the best method. The best method and those that are not statistically significantly different are in bold.

(a) Balanced accuracy (to maximize)

Dimensions	Methods	Datasets: Segmentation	Ionosphere	Credit	Libras	MGT
$k=2$	PCA	0.67±0.01 ***	0.74±0.03 ***	**0.55±0.02**	0.28±0.04 ***	**0.66±0.01**
	LLE	**0.82±0.01**	0.79±0.06 *	0.51±0.02 ***	**0.44±0.07**	**0.66±0.02**
	Isomap	0.78±0.03 ***	**0.82±0.04**	0.53±0.03 **	**0.46±0.04**	0.63±0.01 ***
	$\mathcal{MT}, \mathcal{F}_{\text{dist}}$ (Euclidean)	0.75±0.07 ***	0.78±0.04 **	**0.55±0.02**	0.29±0.04 ***	**0.65±0.04**
	$\mathcal{MT}, \mathcal{F}_{\text{dist}}$ (geodesic)	0.77±0.06 *	0.77±0.06 **	0.54±0.02 *	0.33±0.04 ***	0.63±0.02 **
	$\mathcal{MT}, \mathcal{F}_{\text{rank}}$ (Euclidean)	0.77±0.06 **	0.77±0.06 **	0.53±0.03 **	0.34±0.05 ***	**0.64±0.05**
	$\mathcal{MT}, \mathcal{F}_{\text{rank}}$ (geodesic)	0.74±0.11 *	0.79±0.04 **	**0.56±0.04**	0.34±0.06 ***	0.64±0.03*
	$\mathcal{MT}, \mathcal{F}_{\text{AE}}$	**0.77±0.11**	0.79±0.05 *	0.51±0.02 ***	0.35±0.04 ***	0.63±0.03 **
	$\mathcal{AMT}, \mathcal{F}_{\text{GP}}$	0.77±0.07 *	0.77±0.05 **	0.54±0.03 *	0.35±0.06 ***	0.63±0.03 **
$k=3$	PCA	**0.83±0.01**	0.83±0.03 ***	**0.57±0.03**	0.43±0.05 ***	**0.70±0.01**
	LLE	**0.85±0.01**	0.80±0.05 **	0.51±0.02 ***	0.49±0.06 *	**0.69±0.02**
	Isomap	**0.84±0.02**	**0.84±0.03**	0.53±0.03 **	**0.52±0.04**	**0.69±0.01**
	$\mathcal{MT}, \mathcal{F}_{\text{dist}}$ (Euclidean)	**0.82±0.07**	0.81±0.05 **	**0.58±0.05**	0.45±0.05 ***	**0.71±0.04**
	$\mathcal{MT}, \mathcal{F}_{\text{dist}}$ (geodesic)	**0.85±0.04**	0.82±0.04 **	**0.54±0.04**	0.42±0.06 ***	**0.69±0.05**
	$\mathcal{MT}, \mathcal{F}_{\text{rank}}$ (Euclidean)	**0.83±0.06**	0.83±0.04 **	0.54±0.04 *	0.48±0.07 **	**0.70±0.05**
	$\mathcal{MT}, \mathcal{F}_{\text{rank}}$ (geodesic)	**0.85±0.05**	0.81±0.04 **	0.54±0.03 *	0.46±0.06 ***	**0.68±0.05**
	$\mathcal{MT}, \mathcal{F}_{\text{AE}}$	**0.85±0.05**	0.83±0.05 *	0.54±0.02 *	0.45±0.08 ***	**0.68±0.03**
	$\mathcal{AMT}, \mathcal{F}_{\text{GP}}$	**0.83±0.05**	0.83±0.04 *	0.53±0.04 **	0.46±0.04 ***	**0.69±0.04**

(b) Reconstruction error (to minimize)

Dimensions	Methods	Datasets: Segmentation	Ionosphere	Credit	Libras	MGT
$k=2$	PCA	**0.73±0.06**	**1.27±0.13**	0.94±0.03 ***	**0.95±0.09**	**0.63±0.02**
	LLE	1.01±0.14 ***	**1.30±0.13**	0.96±0.03 ***	1.01±0.09 *	0.81±0.06 ***
	Isomap	0.81±0.10 **	**1.26±0.14**	**0.88±0.03**	**0.97±0.10**	**0.63±0.03**
	$\mathcal{MT}, \mathcal{F}_{\text{dist}}$ (Euclidean)	0.79±0.12 *	**1.29±0.13**	0.96±0.04 ***	**0.97±0.11**	0.66±0.04 *
	$\mathcal{MT}, \mathcal{F}_{\text{dist}}$ (geodesic)	0.79±0.12 *	**1.30±0.13**	0.94±0.04 ***	**0.98±0.10**	0.68±0.04 ***
	$\mathcal{MT}, \mathcal{F}_{\text{rank}}$ (Euclidean)	**0.79±0.15**	1.32±0.14 *	0.96±0.03 ***	**0.96±0.10**	0.69±0.05 ***
	$\mathcal{MT}, \mathcal{F}_{\text{rank}}$ (geodesic)	0.86±0.19 **	1.33±0.15 **	0.97±0.05 ***	**0.96±0.10**	0.72±0.05 ***
	$\mathcal{MT}, \mathcal{F}_{\text{AE}}$	**0.72±0.08**	**1.29±0.14**	**0.89±0.03**	0.95±0.09	0.66±0.04 **
	$\mathcal{AMT}, \mathcal{F}_{\text{GP}}$	0.78±0.09 *	**1.28±0.13**	0.95±0.04 ***	**0.94±0.09**	**0.63±0.03**
$k=3$	PCA	**0.60±0.08**	**1.24±0.14**	0.90±0.04 ***	**0.89±0.10**	0.62±0.02 *
	LLE	0.98±0.15 ***	**1.28±0.14**	0.94±0.03 ***	0.95±0.10 **	0.77±0.04 ***
	Isomap	0.74±0.14 **	**1.25±0.14**	**0.85±0.03**	0.97±0.08 **	**0.62±0.03**
	$\mathcal{MT}, \mathcal{F}_{\text{dist}}$ (Euclidean)	0.69±0.09 **	**1.28±0.14**	0.92±0.04 ***	**0.89±0.11**	0.64±0.02 ***
	$\mathcal{MT}, \mathcal{F}_{\text{dist}}$ (geodesic)	0.75±0.13 ***	**1.30±0.15**	0.92±0.04 ***	**0.92±0.10**	0.65±0.04 **
	$\mathcal{MT}, \mathcal{F}_{\text{rank}}$ (Euclidean)	0.73±0.11 ***	1.31±0.13 *	0.94±0.06 ***	**0.89±0.10**	0.65±0.04 ***
	$\mathcal{MT}, \mathcal{F}_{\text{rank}}$ (geodesic)	0.74±0.16 **	1.34±0.17 *	0.93±0.08 **	**0.87±0.07**	0.69±0.04 ***
	$\mathcal{MT}, \mathcal{F}_{\text{AE}}$	0.67±0.08 **	**1.26±0.14**	0.88±0.03 *	**0.86±0.09**	0.63±0.02 ***
	$\mathcal{AMT}, \mathcal{F}_{\text{GP}}$	0.72±0.11 **	**1.26±0.14**	0.91±0.03 ***	**0.86±0.09**	**0.60±0.03**

Table 4: Expressions of the lower-dimensional representation of the best solution in terms of reconstruction error (out of the 30 independent runs), for all GP-based methods on the Libras and Segmentation datasets from UCI [6].

		Datasets	
Method	**Dim**	**Segmentation**	**Libras Movement**
$\mathcal{F}_{\text{dist}}$ (Euclidean)	$\widetilde{X}^1$	$x_{18} - x_6 - x_8$	$-x_{14} - x_{26} + x_{45} + x_{55}$
	$\widetilde{X}^2$	$x_{14} - x_{17} - x_2$	$x_{55} + x_{58}$
$\mathcal{F}_{\text{dist}}$ (geodesic)	$\widetilde{X}^1$	$x_0 + 3x_{14} + x_7$	$-x_{11}x_{67} + x_{42} + x_{62} + x_{70}$
	$\widetilde{X}^2$	$x_{15}x_{18} + x_{15} + x_8$	$x_0^2 x_{36}$
$\mathcal{F}_{\text{rank}}$ (Euclidean)	$\widetilde{X}^1$	x_{18}	$x_{11} + x_{15} + x_{16} + x_{42} + x_{52}$
	$\widetilde{X}^2$	$x_{14} - x_{17}(x_5 + x_9) - x_{18} + x_2$	$x_{20}(x_{18} + x_{53} - x_{55} + x_7 - x_{74}) - x_{85} + 0.561$
$\mathcal{F}_{\text{rank}}$ (geodesic)	$\widetilde{X}^1$	$x_{11} - x_{15} - x_{16} - x_6$	$x_{10}x_{78} + x_{12} - x_{80}$
	$\widetilde{X}^2$	$0.767x_{14} - x_{18}$	$x_{19} + x_{20} + x_{42} + x_{72} - x_{74}$
$\mathcal{F}_{\text{AE}}$	$\widetilde{X}^1$	$-x_{10} - x_{12} + x_{18} - x_6$	$x_0 - x_{20}x_{86} + x_7$
	$\widetilde{X}^2$	$x_{12}x_5$	$x_{16}x_{53} + x_{29} + x_{59}$
$\mathcal{F}_{\text{GP}}$	$\widetilde{X}^1$	$x_{11} + x_8$	$-x_{28} + x_{40} + x_{53} + x_{86}$
	$\widetilde{X}^2$	$x_2^2 + x_6$	$-x_{15} + x_{20} + x_{59} + x_{73}$

features since the GP-decoder requires a number of trees equal to the number of input features, as depicted in Figure 1c.

Overall, our results show that GP-based dimensionality reduction methods can be on a par with proven baseline methods. In addition, a major advantage of GP-based methods is that they have the potential to produce interpretable mappings in the form of symbolic expressions, which we discuss next.

5.2 Qualitative Analysis

In Table 4, we display the expressions that map the original data into a lower-dimensional representation, for the various GP-based methods. We can see that the resulting embeddings are small enough to have the potential of being interpretable. This is in contrast to the baseline methods where only PCA provide a global functional mapping from the original data to its lower-dimensional representation. However, PCA strictly returns a linear combination of all the variables, limiting its complexity and rendering it unintelligible when the number of input variables is large. On the other hand, GP-based methods can construct non-linear expressions while only using a small subset of the original variables.

6 DISCUSSION

In this paper, we have seen that GP can be competitive with widely used DR methods, while producing small, non-linear, and potentially interpretable functional mappings of how the original data can be compressed into the latent dimensions. In particular, when the fitness used for GP is the same as the objective we want to optimize for, (e.g., reconstruction error), GP often outperforms baselines.

A limitation of this paper is that we adopted classic GP evolutionary operators for variation and selection, yet state-of-the-art GP methods include more interesting mechanisms. For instance, one could use a GP version that is more expressive such as differentiable Cartesian GP [9] (DCGP). This is because in DCGP, one can optimize internal weights of the expression using gradient descent. Another example could be to adopt GP-GOMEA [32] as it is state-of-the-art to discover accurate yet small expressions [15].

Another limitation of this study is that we only used a fixed set of parameters and did not study how different GP representations and fitness functions inter-relate to different evolutionary budgets and settings. For example, we used a classic population size of 1000. However, the population size is a key parameter because it determines the supply of (high-order) building blocks available for the evolution. Some recent works in GP such as [28, 32] suggest that better performance is only achievable when the population size is of tens of thousands or more. In light of this, future work on understanding the potential of GP for DR should include an analysis of the effect of important parameter settings. Moreover, future work should also include more baselines and loss functions in the comparison, including, e.g., the recenly introduced PaCMAP [34].

An interesting aspect emerging from our results is that DR methods achieving high accuracy often performed worse in terms of reconstruction error (and vice versa). This can happen because, when performing reconstruction, each reconstructed feature is equally important; whereas to answer a predictive task as per classification, only a subset of them may be.

7 CONCLUSION

In conclusion, we have considered different versions, i.e., representations and fitness functions of which two are novel (GP autoencoder representation and autoencoder teacher fitness), of genetic programming for dimensionality reduction. We have found that genetic programming is a competitive approach to obtain interpretable mappings for dimensionality reduction. We thus believe that genetic programming for dimensionality reduction is a promising research avenue.

8 ACKNOWLEDGEMENTS

This research is part of the research programme Open Competition Domain Science-KLEIN with project number OCENW.KLEIN.111, which is financed by the Dutch Research Council (NWO).

REFERENCES

[1] Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Multi-tree genetic programming with new operators for transfer learning in symbolic regression with incomplete data. IEEE Transactions on Evolutionary Computation (2021)

[2] Ben-Hur, A., Guyon, I.: Detecting stable clusters using principal component analysis. In: Functional genomics, pp. 159–182. Springer (2003)

[3] Brodersen, K.H., Ong, C.S., Stephan, K.E., Buhmann, J.M.: The balanced accuracy and its posterior distribution. In: 2010 20th international conference on pattern recognition. pp. 3121–3124. IEEE (2010)

[4] Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., Ho, S.: Discovering symbolic models from deep learning with inductive biases. arXiv preprint arXiv:2006.11287 (2020)

[5] Ding, C., He, X.: K-means clustering via principal component analysis. In: Proceedings of the twenty-first international conference on Machine learning. p. 29 (2004)

[6] Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml

[7] Evans, B.P., Xue, B., Zhang, M.: What's inside the black-box? a genetic programming method for interpreting complex machine learning models. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1012–1020 (2019)

[8] Huang, S.H., Wulsin, L.R., Li, H., Guo, J.: Dimensionality reduction for knowledge discovery in medical claims database: application to antidepressant medication utilization study. Computer methods and programs in biomedicine **93**(2), 115–123 (2009)

[9] Izzo, D., Biscani, F., Mereta, A.: Differentiable genetic programming. In: European conference on genetic programming. pp. 35–51. Springer (2017)

[10] Jolliffe, I.T., Cadima, J.: Principal component analysis: a review and recent developments. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **374**(2065), 20150202 (2016)

[11] Katole, S.N., Karmore, S.P.: A new approach of microarray data dimension reduction for medical applications. In: 2015 2nd International Conference on Electronics and Communication Systems (ICECS). pp. 409–413. IEEE (2015)

[12] Kendall, M.G.: A new measure of rank correlation. Biometrika **30**(1/2), 81–93 (1938)

[13] Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Statistics and computing **4**(2), 87–112 (1994)

[14] Koza, J.R., Koza, J.R.: Genetic programming: On the programming of computers by means of natural selection, vol. 1. MIT press (1992)

[15] La Cava, W., Orzechowski, P., Burlacu, B., de França, F.O., Virgolin, M., Jin, Y., Kommenda, M., Moore, J.H.: Contemporary symbolic regression methods and their relative performance. arXiv preprint arXiv:2107.14351 (2021)

[16] Lensen, A., Xue, B., Zhang, M.: Improving k-means clustering with genetic programming for feature construction. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 237–238 (2017)

[17] Lensen, A., Xue, B., Zhang, M.: Can genetic programming do manifold learning too? In: European Conference on Genetic Programming. pp. 114–130. Springer (2019)

[18] Lensen, A., Xue, B., Zhang, M.: Genetic programming for evolving a front of interpretable models for data visualization. IEEE transactions on cybernetics (2020)

[19] Lensen, A., Xue, B., Zhang, M.: Genetic programming for evolving similarity functions for clustering: Representations and analysis. Evolutionary computation **28**(4), 531–561 (2020)

[20] Lensen, A., Zhang, M., Xue, B.: Multi-objective genetic programming for manifold learning: balancing quality and dimensionality. Genetic Programming and Evolvable Machines pp. 1–33 (2020)

[21] Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(Nov), 2579–2605 (2008)

[22] Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. The annals of mathematical statistics pp. 50–60 (1947)

[23] Masters, D., Luschi, C.: Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612 (2018)

[24] Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R.: A field guide to genetic programming. Lulu. com (2008)

[25] Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. science **290**(5500), 2323–2326 (2000)

[26] Sammon, J.W.: A nonlinear mapping for data structure analysis. IEEE Transactions on computers **100**(5), 401–409 (1969)

[27] Schofield, F., Lensen, A.: Evolving simpler constructed features for clustering problems with genetic programming. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8. IEEE (2020)

[28] Schweim, D., Wittenberg, D., Rothlauf, F.: On sampling error in genetic programming. Natural Computing pp. 1–14 (2021)

[29] Tenenbaum, J.B., De Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. science **290**(5500), 2319–2323 (2000)

[30] Vigna, S.: A weighted correlation index for rankings with ties. In: Proceedings of the 24th international conference on World Wide Web. pp. 1166–1176 (2015)

[31] Virgolin, M., Alderliesten, T., Bosman, P.A.: On explaining machine learning models by evolving crucial and compact features. Swarm and Evolutionary Computation **53**, 100640 (2020)

[32] Virgolin, M., Alderliesten, T., Witteveen, C., Bosman, P.A.: Improving model-based genetic programming for symbolic regression of small expressions. Evolutionary computation **29**(2), 211–237 (2021)

[33] Virgolin, M., De Lorenzo, A., Medvet, E., Randone, F.: Learning a formula of interpretability to learn interpretable formulas. arXiv preprint arXiv:2004.11170 (2020)

[34] Wang, Y., Huang, H., Rudin, C., Shaposhnik, Y.: Understanding how dimension reduction tools work: an empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. Journal of Machine Learning Research **22**(201), 1–73 (2021)

[35] Zwillinger, D., Kokoska, S.: CRC standard probability and statistics tables and formulae. Crc Press (1999)

[36] Adadi, A., Berrada, M.: Peeking Inside the Black-box: A Survey on eXplainable Artificial Intelligence (XAI)

[37] Schölkopf, B., Smola, A., Müller, K.: Nonlinear component analysis as a kernel eigenvalue problem Neural computation **10**(5), 1299–1319 (1998)

[38] Muñoz, L., Trujillo, L., Silva, S.: Transfer learning in constructive induction with Genetic Programming Genetic Programming and Evolvable Machines **21**(4), 529–569. Springer (2020)

[39] Lipton, Z.: The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery Queue 16.3, 31–57 (2018)