Is k Nearest Neighbours Regression Better Than GP?

Leonardo Vanneschi[1,2(✉)], Mauro Castelli[1], Luca Manzoni[3], Sara Silva[2], and Leonardo Trujillo[4]

[1] NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa, Portugal
{lvanneschi,mcastelli}@novaims.unl.pt
[2] LASIGE, Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisboa, Portugal
{lvanneschi,sara}@fc.ul.pt
[3] Department of Mathematics and Geosciences, University of Trieste, Via Valerio 12/1, 34127 Trieste, Italy
lmanzoni@units.it
[4] Tecnológico Nacional de México/IT de Tijuana, Tijuana, BC, Mexico
leonardo.trujillo@tectijuana.edu.mx

Abstract. This work starts from the empirical observation that k nearest neighbours (KNN) consistently outperforms state-of-the-art techniques for regression, including geometric semantic genetic programming (GSGP). However, KNN is a memorization, and not a learning, method, i.e. it evaluates unseen data on the basis of training observations, and not by running a learned model. This paper takes a first step towards the objective of defining a learning method able to equal KNN, by defining a new semantic mutation, called random vectors-based mutation (RVM). GP using RVM, called RVMGP, obtains results that are comparable to KNN, but still needs training data to evaluate unseen instances. A comparative analysis sheds some light on the reason why RVMGP outperforms GSGP, revealing that RVMGP is able to explore the semantic space more uniformly. This finding opens a question for the future: is it possible to define a new genetic operator, that explores the semantic space as uniformly as RVM does, but that still allows us to evaluate unseen instances without using training data?

1 Introduction

Geometric Semantic Genetic Programming (GSGP) [1,2] is a variant of Genetic Programming (GP) [3] that uses Geometric Semantic Operators (GSOs) instead of the standard crossover and mutation. It induces a unimodal fitness landscape for any supervised learning problem; so it is an extremely powerful optimizer and, at the same time, it can limit overfitting [4]. The popularity of GSGP has steadily grown in the last few years, and it is nowadays a well established Machine Learning (ML) method. Nevertheless, GSGP generates very large predictive models,

which are extremely hard to read and understand [1,2]. Even though several implementations have been introduced, that make GSGP usable and efficient [5–7], the lack of interpretability of the model is still an issue. Aware that in GSGP the most important GSO is mutation, and that GSGP without crossover can often outperform GSGP that uses crossover [8,9], in this paper, we introduce a mutation intending to ease a model's interpretability.

The mutation traditionally used by GSGP, called Geometric Semantic Mutation (GSM) uses two random trees. The evaluation of those random trees on training instances is used to obtain a different (random) value for each observation, which is subsequently used to calculate the modification caused by mutation to the outputs of the individual. The operator we introduce in this work, called Random Vectors-based Mutation (RVM), replaces the random trees with a vector of random numbers of the same length as the number of training instances. In this way, for each observation, one different random number is used to decide the modification of the output. GP using RVM as the sole genetic operator will be called RVMGP. Clearly, at the end of a RVMGP evolution, we do not have a real "model" (intended as a program that can be executed on unseen data), and so a different strategy has to be designed for generalizing. In this paper, we adopt a method that is very similar to the one used by k Nearest Neighbours (KNN) [10,11]: the output on an unseen instance is calculated using the similarity between the unseen instance and the training observations. Given that RVMGP works similarly to KNN on unseen data, it makes sense to compare RVMGP not only to GSGP, but also to KNN itself. To make the experimental comparison more complete, we also compare these methods to Random Forest (RF) regression [12], that is currently considered by several researchers as the state of the art for regression with ML, at least for "non-big data" problems [13,14]. The outcome of this experimental study, carried on six real-life regression problems, paves the way to fundamental questions on the relevance itself of using GP, a discussion that is tackled at the end of this paper.

The manuscript is organized as follows: Sect. 2 introduces GSGP. Section 3 introduces RVM. Section 4 presents our experimental study; after presenting the test problems and the employed experimental settings, the comparison between RVMGP and GSGP is presented in Sect. 4.2 and the one between RVMGP, KNN and RF regression in Sect. 4.3. Finally, Sect. 5 concludes the paper.

2 Geometric Semantic Genetic Programming

Let $\mathbf{X} = \{\overrightarrow{x_1}, \overrightarrow{x_2}, ..., \overrightarrow{x_n}\}$ be the set of input data (training instances, observations or fitness cases) of a symbolic regression problem, and $\overrightarrow{t} = [t_1, t_2, ..., t_n]$ the vector of the respective expected output or target values (in other words, for each $i = 1, 2, ..., n$, t_i is the expected output corresponding to input $\overrightarrow{x_i}$). A GP individual (or program) P can be seen as a function that, for each input vector $\overrightarrow{x_i}$ returns the scalar value $P(\overrightarrow{x_i})$. Following [2], we call *semantics* of P the vector $\overrightarrow{s_P} = [P(\overrightarrow{x_1}), P(\overrightarrow{x_2}), ..., P(\overrightarrow{x_n})]$. This vector can be represented as a point

in a n-dimensional space, that we call *semantic space*. Remark that the target vector $\overrightarrow{t}$ itself is a point in the semantic space.

As explained above, GSGP is a variant of GP where the standard crossover and mutation are replaced by new operators called Geometric Semantic Operators (GSOs). The objective of GSOs is to define modifications on the syntax of GP individuals that have a precise effect on their semantics. More in particular: geometric semantic crossover generates one offspring, whose semantics stands in the line joining the semantics of the two parents in the semantic space and geometric semantic mutation, by mutating an individual i, allows us to obtain another individual j such that the semantics of j stands inside a ball of a given predetermined radius, centered in the semantics of i. One of the reasons why GSOs became popular in the GP community is probably related to the fact that GSOs induce an unimodal error surface (on training data) for any supervised learning problem, where fitness is calculated using an error measure between outputs and targets. In other words, using GSOs the error surface on training data is guaranteed to not have any locally optimal solution. This property holds, for instance, for any regression or classification problem, independently on how big and how complex data are (reference [1] contains a detailed explanation of the reason why the error surface is unimodal and its importance). The definitions of the GSOs are, as given in [2], respectively:

Geometric Semantic Crossover (GSC). Given two parent functions $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic crossover returns the real function $T_{XO} = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$, where T_R is a random real function whose output values range in the interval $[0, 1]$.

Geometric Semantic Mutation (GSM). Given a parent function $T : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic mutation with mutation step ms returns the real function $T_M = T + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random real functions.

The reason why GSM uses two random trees T_{R1} and T_{R2} is that the amount of modification caused by GSM must be centered in zero. In other words, a random expression is needed that has the same probability of being positive or negative. Even though this is not in the original definition of GSM, later contributions [1,4,9] have clearly shown that limiting the codomain of T_{R1} and T_{R2} in a predefined interval (for instance $[0, 1]$, as it is done for T_R in GSC) helps to improve the generalization ability of GSGP. As in several previous works [1,5], we constrain the outputs of T_R, T_{R1}, and T_{R2} by wrapping them in a logistic function. Only the definitions of the GSOs for symbolic regression problems are given here, since they are the only ones used in this work. For the definition of GSOs for other domains, the reader is referred to [2].

As reported in [1,2], the property of GSOs of inducing a unimodal error surface has a price. The price, in this case, is that GSOs always generate larger offspring than the parents, and this entails a rapid growth of the size of the individuals in the population. To counteract this problem, in [5–7] implementations of GSOs were proposed, that make GSGP not only usable in practice, but also significantly faster than standard GP. This is possible through a smart

representation of GP individuals, that allows us to not store their genotypes during the evolution. The implementation presented in [5] is the one used here. Even though this implementation is efficient, it does not solve the problem of the size of the final model: the genotype of the final solution returned by GSGP can be reconstructed, but it is so large that it is practically impossible to understand it. This turns GSGP into a "black-box" system, as many other popular ML systems are, including deep neural networks.

Several previous contributions (see for instance [8]) have clearly demonstrated that, in GSGP, the most important genetic operator is GSM and in many cases a GSGP system using only GSM, and no GSC, can obtain comparable (or even better) results to the ones of a system using both these operators. Even though GSM limits the problem of the rapid growth of code inside the population (this growth is exponential for GSC, but slower for GSM), the issue remains. In other words, even using only GSM, the final model is often so large that it is hardly readable and practically impossible to understand. Trying to solve this issue is one of the motivations for introducing the novel mutation operator presented in the next section.

3 Random Vector Based Mutation

The rapid code growth caused by GSM can be explained by the fact that the offspring (T_M in the definition of GSM given in Sect. 2) contains the genotype of the parent (T), plus the genotype of two random trees (T_{R1} and T_{R2}) and 4 further nodes. Replacing the two random trees with a random number (i.e. a scalar constant) would vastly limit the code growth. Nevertheless, as explained in [1], this would not allow us to implement ball mutation on the semantic space, which is the objective. Such a mutation would, in fact, modify the semantics of parent T *of the same constant amount* for all its coordinates. On the other hand, the optimization power of GSM is given by the fact that GSM can modify the semantics of T by *a different* amount for each one of its coordinates, since T_{R1} and T_{R2} typically return different values when evaluated on the different training observations. To understand the importance of this, one may consider the case in which one coordinate of T is extremely "close" to the corresponding target, while another coordinate is extremely "far". Modifying both these coordinates of the same quantity would never allow us to transform T into the global optimum.

In this work, we propose to use a *vector* of random numbers $\overrightarrow{v}$, of the same length as the number of training observations, to modify the semantics of the individuals by different quantities for each one of its coordinates. Each element $\overrightarrow{v}[i]$ of vector $\overrightarrow{v}$ stores the particular modification that mutation apports to the i^{th} coordinate of the semantics of T. In GSM, if the codomain of T_{R1} and T_{R2} is constrained in $[0, 1]$ (as it is customary [1,4,9]), then, for each coordinate of the semantic vector, the modification is given by a random number included in $[-ms, ms]$. To simulate as closely as possible this behaviour of GSM, in this work each coordinate $\overrightarrow{v}[i]$ of vector $\overrightarrow{v}$ will contain a random number extracted with uniform distribution from $[-ms, ms]$.

The functioning should be clarified by the following example. Let us assume that we have the following training set D, composed by 3 observations (lines) and 2 features (columns), and the following corresponding target vector $\overrightarrow{t}$:

$$D = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 40 & 20 \end{bmatrix} \qquad \overrightarrow{t} = \begin{bmatrix} 4 \\ 10 \\ 100 \end{bmatrix}$$

Let us also assume that we have a GP individual $P = x_1 + x_2$. The semantics of P is equal to: $\overrightarrow{s_P} = [3, 7, 60]$. Let us also assume, for simplicity, that $ms = 1$. All we have to do to mutate P is to generate a vector $\overrightarrow{v}$ of random numbers in $[-1, 1]$, of the same length as the number of training observations; for instance: $\overrightarrow{v} = [0.75, -0.25, 0.4]$. In this way, the offspring P_M of the mutation of P will be an individual whose semantics is:

$$\overrightarrow{s_{P_M}} = \overrightarrow{s_P} + \overrightarrow{v} \tag{1}$$

or, in other words: $\overrightarrow{s_{P_M}} = [3.75, 6.75, 60.4]$. As we can see, each coordinate of $\overrightarrow{v}$ has been used to update the corresponding coordinate of $\overrightarrow{s_P}$. We call this type of mutation Random Vector based Mutation (RVM), and GP that uses RVM as the unique genetic operator RVMGP. Both GSM and RVM can be defined as follows:

$$T_M = T + \Delta T \tag{2}$$

where the only difference between GSM and RVM is given by a different ΔT: ΔT is equal to $ms \cdot (T_{R1} - T_{R2})$ for GSM (as in the definition of Sect. 2) and it is equal to a *different* random number in $[-ms, ms]$ for each training observation for RVM.

At this point, a question comes natural: how can RVMGP be used to calculate the output on unseen observations? The idea proposed in this work is inspired by the KNN algorithm. In particular, given that only regression applications will be used as test problems, the inspiration is taken from KNN regression [10]. It consists in calculating the average of the outputs of the model on the k nearest training observations. Considering the previous example again, let us consider an unseen observation like, for instance: $\overrightarrow{u} = [2, 3]$. What is the output of individual P_M on observation $\overrightarrow{u}$? If we assume, for instance, that $k = 2$, all we have to do is to calculate the two closest instances to $\overrightarrow{u}$ in the training set D and calculate the average of the outputs of P_M on those instances. Considering, for instance, Euclidean distance as the metric used to calculate the k nearest training observations, the two observations that are closer to $\overrightarrow{u}$ in D are the first and the second observations, i.e. $[1, 2]$ and $[3, 4]$. Considering the output values of P_M on those two observations, i.e. the first two coordinates of s_{P_M}, the output of P_M on unseen instance $\overrightarrow{u}$ is equal to:

$$P_M(\overrightarrow{u}) = \frac{3.75 + 6.75}{2} = 5.25 \tag{3}$$

Let us now take a moment to ponder what is the final model (i.e. the minimum amount of information to calculate the output on unseen instances) for RVMGP.

Actually, the model can be seen from two different viewpoints: the first one is to consider the initial tree, plus the vector of random numbers used to translate its semantics, plus the training set. For instance, for the previous example, if P_M was the final individual returned by RVMGP, one may say that the model is given by:

$$P = x_1 + x_2, \qquad \overrightarrow{v} = [0.75 \quad -0.25 \quad 0.4], \qquad D = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 40 & 20 \end{bmatrix} \tag{4}$$

It should be noticed that, if a new generation is executed by RVMGP, P_M will probably be mutated, generating a new individual P'_M, where the semantic of P'_M can be obtained by summing the semantics of P_M to a vector of random trees:

$$\overrightarrow{s_{P'_M}} = \overrightarrow{s_{P_M}} + \overrightarrow{v_1} \tag{5}$$

But, replacing Eq. (1) into Eq. (5), we obtain: $\overrightarrow{s_{P'_M}} = \overrightarrow{s_P} + \overrightarrow{v} + \overrightarrow{v_1}$, and if we define $\overrightarrow{w} = \overrightarrow{v} + \overrightarrow{v_1}$, we obtain: $\overrightarrow{s_{P'_M}} = \overrightarrow{s_P} + \overrightarrow{w}$. In other words, also $s_{P'_M}$ can be defined using the semantics of the initial individual P and a vector of random numbers. This reasoning can be generalized to any number of generations. So, independently from the number of generations performed by RVMGP, it will always be possible to interpret the final model as an individual from the initial population, plus a vector of random numbers, plus the training set (as in Eq. (4)).

A second possible way of interpreting the model returned by RVMGP is to consider the semantics of the final individual, plus the training set. Considering the previous example, and assuming that P_M is the final solution returned by RVMGP, the model would be:

$$\overrightarrow{s_{P_M}} = [3.75 \quad 6.75 \quad 60.4], \qquad D = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 40 & 20 \end{bmatrix} \tag{6}$$

It should be noticed that this second way of interpreting the RVMGP model (reported in Eq. (6)) is completely equivalent to the first one (reported in Eq. (4)), since s_{P_M} can be obtained directly by evaluating P on each line of D, and summing $\overrightarrow{v}$. It is only a different way of presenting the same information: while the first interpretation (Eq. (4)) still contains a GP tree, and so vaguely reminds a traditional GP model, the second interpretation (Eq. (6)) allows us to save memory space and to calculate the output on unseen instances faster. If the model is stored as in Eq. (4), to calculate the output on an unseen instance $\overrightarrow{u}$ we have to evaluate $o = P(\overrightarrow{u})$, calculate the k nearest instances to $\overrightarrow{u}$ in D, and sum to o the average of the corresponding coordinates in $\overrightarrow{v}$. On the other hand, if the model is stored as in Eq. (6), all we have to do is to calculate the k nearest instances to $\overrightarrow{u}$ in D and return the average of the corresponding coordinates in $\overrightarrow{s_{P_M}}$. It is not hard to convince oneself that these two processes lead exactly to the same result, but the second one is faster because it does not involve the evaluation of a program on the unseen instance. We are aware that, in the presence of vast training sets, this could be a large amount of information to store

(as vastly discussed in the literature as a drawback of KNN [10]). However, this is still convenient, in terms of memory occupation, compared to storing the huge models generated by GSGP, when GSM is employed.

Finally, it is worth pointing out that the only difference between RVMGP and KNN regression is that KNN regression uses the target values corresponding to the k nearest training observations, instead of the corresponding output of an individual. In other words, considering the previous example, the output of KNN regression for observation $\overrightarrow{u}$ would have been equal to: $\text{KNN}(\overrightarrow{u}) = \frac{4+10}{2} = 14$.

4 Experimental Study

This section is organized as follows: Sect. 4.1 presents the test problems used for our experimental study and the employed parameter settings. Section 4.2 contains an experimental comparison between GSGP using GSM and RVMGP using RVM (no crossover is considered in this study). Finally, Sect. 4.3 extends the experimental comparison, by including also KNN regression and RF regression. From now on, for simplicity, GP using only GSM will be indicated as GSGP. The notation GSGP-log will be used to indicate the variant of GSGP in which the codomain of random trees T_{R1} and T_{R2} used by GSM are constrained in $[0, 1]$ by wrapping them with a logistic function, as in [1,4,9]. The notation GSGP-nolog will be used to indicate the variant in which the codomains of T_{R1} and T_{R2} are not constrained at all. Finally, to indicate the variant of RVMGP using a particular value $k = x$, we will use the notation RVM-kx.

4.1 Test Problems and Experimental Settings

The six real-life datasets used as test problems are described in Table 1. The table shows, for each dataset, the number of features, the number of observations, and a reference where more information about the data and the application can be found. The six datasets have already been used as test problems for GP before.

Table 1. Description of the test problems. For each dataset, the number of features (independent variables) and the number of instances (observations) are reported.

Dataset	# Features	# Instances
Bioavailability [15]	241	359
Concrete [16]	8	1029
Energy [17]	8	768
Park Motor [18]	18	5875
Park Total [18]	18	5875
PPB [15]	628	131

Table 2. Parameter setting used in our experiments for the studied GP variants.

Parameter	Setting
Population size	100
Max. # of generations	2000
Initialization	Ramped H-H
Crossover rate	0
Mutation rate	1
Max. depth for initialization	6
Tournament selection, size	4

Previous contributions, including the ones referenced in Table 2, clearly show that GSGP outperforms standard GP (i.e. GP using the standard Koza's genetic operators [3]) on all these test problems. For this reason, standard GP is not studied here. For each one of these datasets, 30 independent runs of each one of the studied methods where performed. For each run, a different partition of the dataset into training and test set was used, where 70% of the instances, randomly selected with uniform distribution, form the training set and the remaining 30% were used as a test set.

Table 2 reports the values of the parameters that were used in our GP experiments. Besides, elitism was applied by copying the best individual in the next population at each generation. The mutation step, for all the studied methods, was a random number, extracted with uniform distribution from $[0, 1]$, as proposed in [9]. Concerning RVMGP, different values of k were studied ($k = 1, 5, 10, 20, 50$) and experimentally compared. Concerning KNN, the same values of k as for RVMGP were studied. For both algorithms, the measure used to calculate the similarity between instances was the Euclidean distance, calculated using all the features in the dataset. Concerning RF, least-squares boosting was used, with a maximum of 10 splits per tree and 100 trees.

4.2 Experimental Results: RVMGP vs GSGP

Figure 1 reports the results on the training set obtained by RVMGP, GSGP-log and GSGP-nolog. On training data, RVMGP clearly outperforms both GSGP-log and GSGP-nolog for all the studied test problems. Figure 2 reports the results on the test set. Concerning RVMGP, to avoid cluttering the plots, only the best and worse values of k for each test problem are reported. Concerning GSGP, only the curve of GSGP-log is reported, because, for each studied problem, GSGP-nolog returns results on the test set that are so much worse than the other studied methods that reporting the curve of GSGP-nolog would not allow us to appreciate the mutual differences between the other methods. The fact that GSGP-nolog has a poor generalization ability, hence the need of constraining the output of the random trees generated by GSM, was already known in the literature [1,4], and our study is a further confirmation of this finding.

Concerning Fig. 2, let us not consider, for the moment, the horizontal straight lines, that represent the results returned by KNN regression. Those results will be discussed in Sect. 4.3. Figure 2 clearly shows that RVMGP with the best-studied k consistently outperforms GSGP on all the studied test problems, while RVMGP with the worst studied k outperforms GSGP in 4 cases over 6. To assess the statistical significance of these results, a set of tests has been performed. The Lilliefors test has shown that the data are not normally distributed and hence a rank-based statistic has been used. The Wilcoxon rank-sum test for pairwise data comparison with Bonferroni correction has been used, under the alternative hypothesis that the samples do not have equal medians at the end of the run, with a significance level $\alpha = 0.05$. The p-values are reported in Table 3, where statistically significant differences are highlighted with p-values in bold. As we can observe, all the differences are statistically significant, except the

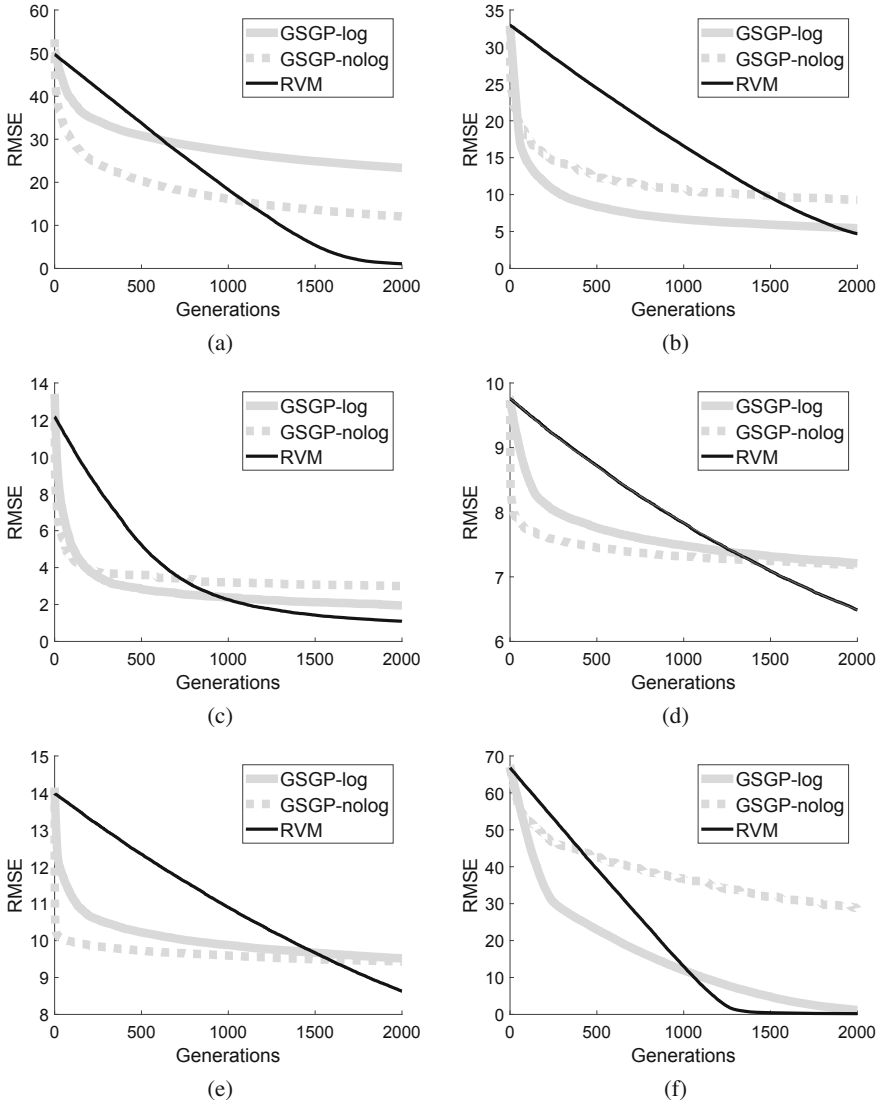

Fig. 1. Median best RMSE on the *training set* obtained by GSGP-log, GSGP-nolog and RVM. (a) = Bioavailability; (b) = Concrete; (c) = Energy; (d) = ParkMotor; (e) = ParkTotal; (f) = PPB.

difference between GSGP and RVMGP with the worst k for the Concrete and PPB datasets. The fact that, for different problems, the best value of k changes is an issue that has been already discussed in the literature for KNN regression [19]. The experimental results reported here seem to confirm that this issue also exists for RVMGP.

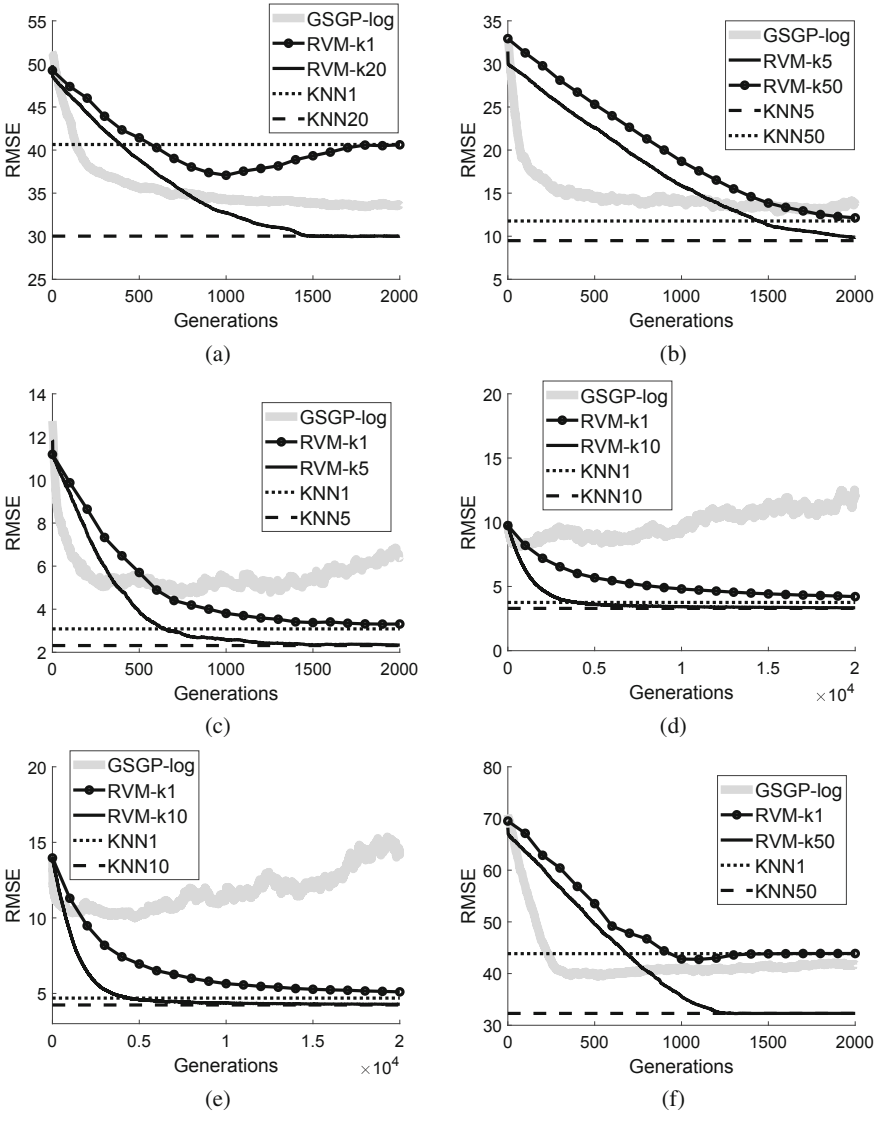

Fig. 2. Median RMSE on the *test set* obtained by GSGP-log and RVMGP. For RVMGP, only the values of k that have allowed us to obtain the best and the worse results are reported. The best is represented with a black-continuous line, the worst with a black line annotated with points. For KNN, the same k values as for RVMGP are reported. The results of KNN are shown as horizontal straight lines. (a) = Bioavailability; (b) = Concrete; (c) = Energy; (d) = Park Motor; (e) = Park Total; (f) = PPB.

Table 3. p-values of the Wilcoxon rank-sum test on unseen data for the experiments of Fig. 2, under the alternative hypothesis that the samples do not have equal medians. Bold denotes statistically significant values.

	GSGP-log *vs* best RVM	GSGP-log *vs* worst RVM	best RVM *vs* worst RVM
Bioavailability	$\mathbf{7.70 \times 10^{-8}}$	$\mathbf{1.46 \times 10^{-10}}$	$\mathbf{3.02 \times 10^{-11}}$
Concrete	$\mathbf{3.82 \times 10^{-9}}$	0.0933	$\mathbf{3.34 \times 10^{-11}}$
Energy	$\mathbf{3.02 \times 10^{-11}}$	$\mathbf{4.08 \times 10^{-5}}$	$\mathbf{3.02 \times 10^{-11}}$
Park Motor	$\mathbf{7.39 \times 10^{-11}}$	$\mathbf{6.12 \times 10^{-10}}$	$\mathbf{4.62 \times 10^{-10}}$
Park Total	$\mathbf{3.02 \times 10^{-11}}$	$\mathbf{2.19 \times 10^{-8}}$	$\mathbf{3.02 \times 10^{-11}}$
PPB	$\mathbf{1.33 \times 10^{-10}}$	0.1154	$\mathbf{3.16 \times 10^{-10}}$

All this considered, we can state that RVMGP, once the best value of k is discovered, is preferable to GSGP for the quality of the returned solutions. An attempt to motivate this result is given in Fig. 3. In this figure, the amounts of modification of the different studied mutation operators (i.e. the quantities ΔT in Eq. (2)) are reported. More in particular, for each individual in the population to which mutation was applied, the used value of ΔT for each fitness case was stored. Given that there is no reason why the values of ΔT should change along the evolution, only the values at the first generation are reported. The scatterplots of Fig. 3 have the fitness cases (i.e. the training instances) on the horizontal axis, ordered randomly. In other words, the values on the horizontal axis are discrete and they consist in the integer values $1, 2, ..., N$, where N is the number of training instances. For each one of the values on the horizontal axis (i.e. for each fitness case) a "column" of points is reported, one for each individual in the population. For each one of those points, the value on the vertical axis corresponds to the ΔT value that mutation applied to that individual. To save space, only the results concerning one of the studied test problems are reported here (specifically, Fig. 3 reports the results on the Concrete dataset), but on the other five test problems, the situation is qualitatively the same, leading to the same conclusions.

Figure 3 offers a clear picture of the differences between the mutation operators used by RVMGP (plot (a)), GSGP-log (plot (b)) and GSGP-nolog (plot (c)). Let us begin by discussing the case of GSGP-nolog (plot (c)). Since the codomain of the random trees is not limited, ΔT often assumes very large (positive and negative) values (up to 10^{16}). As a consequence, GSGP-nolog can cause huge modifications in the semantics of the individuals. This may be the cause for an unstable search process, and thus the poor generalization ability of GSGP-nolog.

Let us now focus on the ΔT values of RVMGP (Fig. 3(a)) and GSGP-log (Fig. 3(b)). First of all, it is worth pointing out that, observing the scatterplots, one should not be surprised by the concentration of points around $\Delta T = 0$. In fact, the mutation step ms is a different random number in $[0, 1]$ at each mutation event, and not a constant value. Given that $\Delta T \in [-ms, ms]$, the interval of variation of ΔT changes at each mutation event, and it is expected

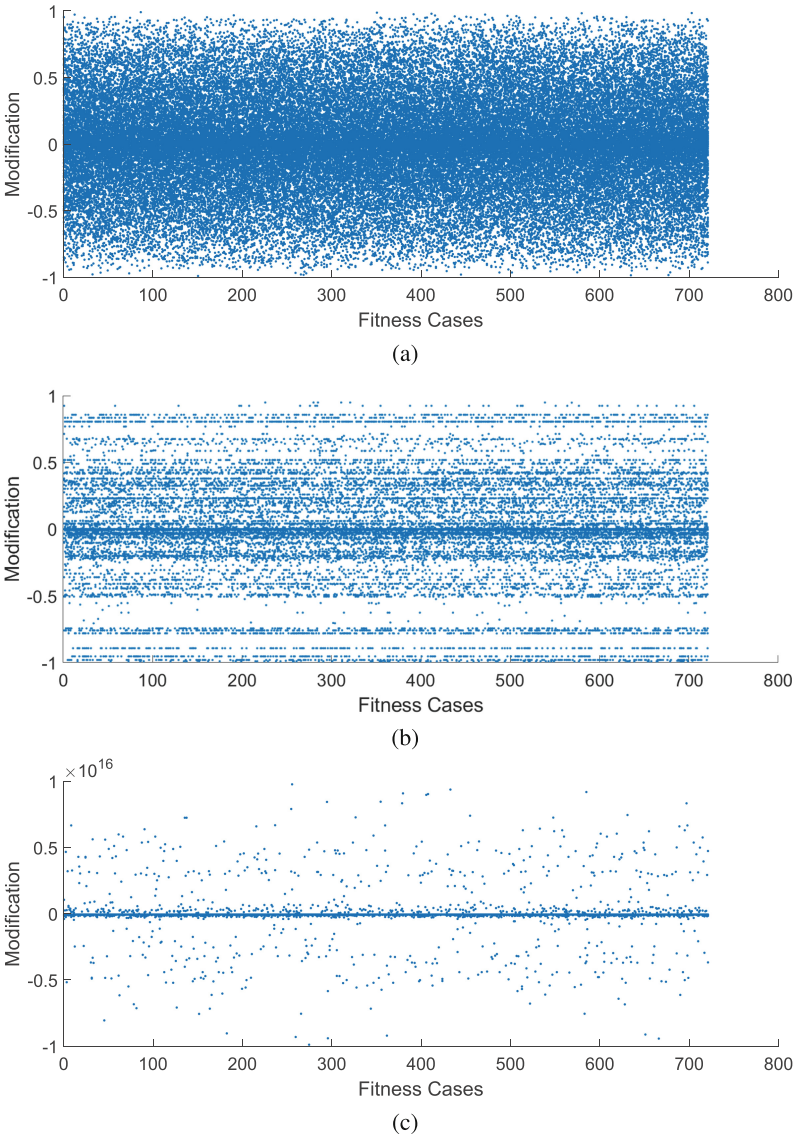

Fig. 3. The modifications ΔT (see Eq. (2)) made by the studied mutations on each training case for each one of the individuals in the population at the first generation for the Concrete dataset. (a) = RVMGP, (b) = GSGP-log, (c) = GSGP-nolog.

that more points are concentrated around zero, while a smaller number of points appear close to the values $\Delta T = 1$ and $\Delta T = -1$. Secondly, one important difference between the scatterplot of RVMGP and the one of GSGP-log is visible: the scatterplot of RVMGP is clearly more "dense" and "uniform". In other words,

practically all possible values of ΔT have been achieved for each fitness case. On the other hand, observing the scatterplot of GSGP-log, we can see that it is less uniform, which makes us hypothesize that some values of ΔT are harder to obtain than others. This depends on the data and on the particular random trees that were generated. From this observation, it is straightforward to infer that there are some points (or even "regions") in the semantic space that are harder than others to reach by GSM, while this is not the case for RVM. We could say that RVM induces a *dense* and regular semantic space, while GSM induces a *sparse* and irregular semantic space. Also, more diverse semantic values appear in a RVMGP population than in a GSGP-log population. In other words, more semantic diversity is offered by RVMGP than by GSGP-log. Given that semantic diversity has been demonstrated as one of the factors promoting generalization ability [20], we hypothesize that the better results achieved by RVMGP in Fig. 2 can be motivated by the different behaviour highlighted in Fig. 3.

4.3 Experimental Results: RVMGP vs KNN Regression vs RF Regression

It is now time to look back at Fig. 2, and consider the horizontal lines, that correspond to the median value of the RMSE achieved by KNN regression on the test set (the same values of k as for RVMGP are reported). Of course, this value is a constant (there is no evolution in KNN), but reporting that value as a horizontal line in the plots helps visibility. Two facts can be observed: first of all, KNN consistently outperforms GSGP on all studied problems; as a consequence, KNN also outperforms standard GP, given that GSGP was able to obtain better results than standard GP on all these problems, as discussed above. Secondly, the evolution of RVMGP approximates KNN, until a point in which it obtains practically identical results, without being able to significantly improve them. That point arrives within generation 2000 for all the studied test problems, except Park Motor and Park Total (Fig. 2(d) and (e), respectively). To see the same behaviour, for those two problems we have extended the runs until generation 20000. Table 4 reports, for each studied problem, the numeric values of the median errors for the worst and best KNN and the worst and best RVMGP. To have a more complete vision of how these results compare with other ML algorithms, also the results obtained by RF regression [12] are reported, since RF regression is often considered the ML state of the art for regression. The interested reader is referred to [13,14] to support the use of RF. To assess the statistical significance of these results, once again the Wilcoxon rank-sum test for pairwise data comparison with Bonferroni correction has been used, under the alternative hypothesis that the samples do not have equal medians, with a significance level $\alpha = 0.05$. The p-values are reported in Table 5, where statistically significant differences are highlighted with p-values in bold.

As we can observe, RF regression outperforms the other studied methods only on two of the six studied problems. Discussing the results of RF is beyond the scope of this paper, nevertheless it is worth pointing out the RF outperforms the other methods for the two problems that have the smaller dimensionality of

Table 4. Median error over 30 independent runs returned by the worst and best KNN, the worst and best RVM and RF regression. The best result for each test problem is highlighted in bold.

	worst KNN	best KNN	worst RVM	best RVM	RF Regr.
Bioavailability	40.64	30.01	40.61	**29.99**	36.45
Concrete	12.13	9.86	11.76	9.48	**5.84**
Energy	3.31	2.33	3.08	2.31	**0.40**
Park Motor	3.75	**3.29**	4.20	3.34	3.80
Park Total	4.69	**4.23**	5.10	4.28	4.62
PPB	43.85	**32.29**	43.88	**32.29**	42.0

Table 5. p-values of the Wilcoxon rank-sum test on unseen data for the experiments of Table 4, under the alternative hypothesis that the samples do not have equal medians. Bold denotes statistically significant values.

	best RVM vs best KNN	best RVM vs RF Regr.	best KNN vs RF Regr.
Bioavailability	0.95	$\mathbf{5.49 \times 10^{-11}}$	$\mathbf{6.70 \times 10^{-11}}$
Concrete	0.92	$\mathbf{3.02 \times 10^{-11}}$	$\mathbf{3.02 \times 10^{-11}}$
Energy	0.34	$\mathbf{3.02 \times 10^{-11}}$	$\mathbf{3.02 \times 10^{-11}}$
Park Motor	0.02	$\mathbf{4.50 \times 10^{-11}}$	$\mathbf{3.02 \times 10^{-11}}$
Park Total	0.23	$\mathbf{4.20 \times 10^{-10}}$	$\mathbf{1.33 \times 10^{-10}}$
PPB	0.92	$\mathbf{8.99 \times 10^{-11}}$	$\mathbf{8.99 \times 10^{-11}}$

the feature space. For the other problems, that are characterized by a larger dimensionality of the feature space, RFs are consistently outperformed both by KNN and by RVMGP. All the differences are statistically significant, with the only exception of the differences between RVMGP and KNN, that are not statistically significant for any of the studied problems.

These observations lead to questions that may look dramatic for the GP community: is KNN better than GP? Are we missing the boat by using GP, while KNN, that is a simpler algorithm, can achieve better results?

Answering these questions is not straightforward, and possibly a single answer does not even exist. It is worth pointing out that the excellent generalization ability of KNN, when compared to other ML algorithms, was already known [21], and has recently been discussed in [22]. In the latter contribution, Cohen and colleagues offer an interesting discussion about the difference between learning and memorizing. It is clear that KNN is memorizing, and not learning. KNN, in fact, does not even have a learning phase, and does not have a real model, intended as a program that can be executed on observations. The model is replaced by the training set and generalization is achieved only by comparing an unseen instance to the training observations. However, as pointed out by Cohen *et al.*, memorization and generalization, which are traditionally considered to be contradicting to each other, are compatible and complementary in ML, and this explains the excellent generalization ability of KNN.

On the other hand, having a model of the data can be convenient in many cases. First of all, because a model, if readable, can be interpreted by a domain expert. If a model can be understood and "makes sense" to a domain expert, she will also more likely trust the predictions. Secondly, KNN bases its functioning on the similarity between training and unseen data, which implies that the functioning of KNN is strongly dependent on the distance metric used to quantify this similarity. This dependence can be avoided, in principle, if we have a model, that can potentially make predictions based on concepts that go beyond the immediate similarity between data.

All this considered, instead of answering the previous questions, one may ask another question: is it possible to obtain the same results as KNN, but by means of learning instead of memorization? Our answer is that, in some senses, this is exactly what RVMGP is doing. Furthermore, although only on two test problems, RF regression was able to outperform KNN, and RF regression is learning and not memorizing.

RVMGP is obtaining the same results as KNN, but after a learning process. However, what leaves us unsatisfied with RVMGP is that, as for KNN, also with RVMGP we need the training data to be able to generalize. But *why* does RVMGP need the training data? Simply because RVM uses a vector of random numbers, one for each training observation, and there is no other available element, unless data similarity, that can let us have the appropriate corresponding number to use on unseen data.

From these considerations, a new and final question comes to our mind: is it possible to simulate the behaviour of RVM using random trees, so that generalization can be obtained simply evaluating the final expression? In the end, as explained in Sect. 4.2, our interpretation of what makes RVMGP able to outperform GSGP comes from the difference in the scatterplots of the ΔTs reported in Fig. 3. So, what if we were able to obtain a "dense" and "regular" scatterplot as the one of Fig. 3(a), but using random trees, instead of vectors of random numbers? We hypothesize that this would allow us to obtain results that are comparable to the ones of KNN, but with the big advantage of having a final, executable, expression, that can be evaluated on unseen data.

These ideas open to new and exciting research questions: why is the scatterplot of the ΔTs of GSM different from the one in Fig. 3(a)? Does it depend on the way we are normalizing data? Does it depend on the primitive operators we are using to build the random trees? Does it depend on their size and shape? Is there any way to obtain a behaviour like the one of Fig. 3(a) using random trees, or is it impossible after all? And even further: can we use trained expressions, instead of random expressions, to obtain a scenario like the one in Fig. 3(a)? Can novelty search [23] help learn such expressions? In the end, from Fig. 3(b) it is clear that GSM is sampling similar values of ΔT several times, while disregarding others. Can we simply reward diversity in the creation of the random expressions used by GSM? Is it enough to obtain an algorithm that works like KNN, but learns instead of memorizing? All these questions deserve future work and answering those questions is one of the main interests of our current research.

5 Conclusions and Future Work

A new geometric semantic mutation, called Random Vector-based Mutation (RVM) was presented in this paper. It has the advantage of reducing the size of the model compared to traditional geometric semantic mutation, and it clearly outperforms it on six real-life regression problems. On the other hand, as for the k Nearest Neighbors (KNN), the only way to evaluate unseen instances is by using the similarity with the training observations, which forces us to include the training set in the model. Furthermore, RVM can approximate KNN, until a point in which it is able to return practically identical results, but it is not able to outperform it significantly. The presented results highlighted an excellent generalization ability of KNN, often better than a state-of-the-art method like Random Forest regression. Furthermore, KNN is a much simpler algorithm than GP. These considerations force GPers to a basic reflection on the reason why we are using GP, questioning whether it even makes sense at all. We conclude that learning, as GP does, can be more important than memorizing, as KNN. This puts our future research in front of a clear and ambitious challenge: obtaining the same results as KNN through a GP-based learning process. The first attempt will come from a deeper analysis of the density of the semantic space, induced by different mutation operators.

Acknowledgments. This work was partially supported by FCT, Portugal, through funding of LASIGE Research Unit (UIDB/00408/2020) and projects BINDER (PTDC/CCI-INF/29168/2017), GADgET (DSAIPA/DS/0022/2018), AICE (DSAIPA/DS/0113/2019), INTERPHENO (PTDC/ASP-PLA/28726/2017), OPTOX (PTDC/CTA-AMB/30056/2017) and PREDICT (PTDC/CCI-CIF/29877/2017), and by the Slovenian Research Agency (research core funding No. P5-0410). We also thank Reviewer 2 for the interesting comments, and apologize for not having had enough time to follow all the helpful suggestions.

References

1. Vanneschi, L.: An introduction to geometric semantic genetic programming. In: Schütze, O., Trujillo, L., Legrand, P., Maldonado, Y. (eds.) NEO 2015. SCI, vol. 663, pp. 3–42. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-44003-3_1
2. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012. LNCS, vol. 7491, pp. 21–31. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32937-1_3
3. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
4. Gonçalves, I., Silva, S., Fonseca, C.M.: On the generalization ability of geometric semantic genetic programming. In: Machado, P., et al. (eds.) EuroGP 2015. LNCS, vol. 9025, pp. 41–52. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16501-1_4
5. Castelli, M., Silva, S., Vanneschi, L.: A C++ framework for geometric semantic genetic programming. Genetic Program. Evolvable Mach. **16**(1), 73–81 (2015)

6. Moraglio, A.: An efficient implementation of GSGP using higher-order functions and memoization. In: Semantic Methods in Genetic Programming, Workshop at Parallel Problem Solving from Nature (2014)
7. Martins, J.F.B.S., Oliveira, L.O.V.B., Miranda, L.F., Casadei, F., Pappa, G.L.: Solving the exponential growth of symbolic regression trees in geometric semantic genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, pp. 1151–1158. ACM, New York (2018)
8. Moraglio, A., Mambrini, A.: Runtime analysis of mutation-based geometric semantic genetic programming for basis functions regression. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO 2013, pp. 989–996. ACM, New York (2013)
9. Vanneschi, L., Silva, S., Castelli, M., Manzoni, L.: Geometric semantic genetic programming for real life applications. In: Riolo, R., Moore, J.H., Kotanchek, M. (eds.) Genetic Programming Theory and Practice XI. GEC, pp. 191–209. Springer, New York (2014). https://doi.org/10.1007/978-1-4939-0375-7_11
10. Kramer, O.: K-nearest neighbors. In: Kramer, O. (ed.) Dimensionality Reduction with Unsupervised Nearest Neighbors. Intelligent Systems Reference Library, vol. 51, pp. 13–23. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38652-7_2
11. Mucherino, A., Papajorgji, P.J., Pardalos, P.M.: k-nearest neighbor classification. In: Mucherino, A., Papajorgji, P.J., Pardalos, P.M. (eds.) Data Mining in Agriculture. Springer Optimization and Its Applications, vol. 34, pp. 83–106. Springer, New York (2009). https://doi.org/10.1007/978-0-387-88615-2_4
12. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
13. Verikas, A., Gelzinis, A., Bacauskiene, M.: Mining data with random forests: a survey and results of new tests. Pattern Recogn. **44**(2), 330–349 (2011)
14. Ziegler, A., König, I.: Mining data with random forests: current options for real-world applications. Wiley Interdisc. Rev. Data Min. Knowl. Discov. **4**, 55–63 (2014)
15. Archetti, F., Lanzeni, S., Messina, E., Vanneschi, L.: Genetic programming for computational pharmacokinetics in drug discovery and development. Genetic Program. Evolvable Mach. **8**(4), 413–432 (2007)
16. Castelli, M., Vanneschi, L., Silva, S.: Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. Expert Syst. Appl. **40**(17), 6856–6862 (2013)
17. Castelli, M., Trujillo, L., Vanneschi, L., Popovič, A.: Prediction of energy performance of residential buildings: a genetic programming approach. Energy Buildings **102**, 67–74 (2015)
18. Castelli, M., Vanneschi, L., Silva, S.: Prediction of the unified Parkinson's disease rating scale assessment using a genetic programming system with geometric semantic genetic operators. Expert Syst. Appl. **41**(10), 4608–4616 (2014)
19. Cheng, D., Zhang, S., Deng, Z., Zhu, Y., Zong, M.: kNN algorithm with data-driven k value. In: Luo, X., Yu, J.X., Li, Z. (eds.) ADMA 2014. LNCS (LNAI), vol. 8933, pp. 499–512. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14717-8_39
20. Galván, E., Schoenauer, M.: Promoting semantic diversity in multi-objective genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, pp. 1021–1029. ACM, New York (2019)

21. Chen, G.H., Shah, D.: Explaining the success of nearest neighbor methods in prediction. Found. Trends® in Mach. Learn. **10**(5–6), 337–588 (2018)
22. Cohen, G., Sapiro, G., Giryes, R.: DNN or k-NN: that is the generalize vs. memorize question. ArXiv abs/1805.06822 (2018)
23. Slavinec, M., et al.: Novelty search for global optimization. Appl. Math. Comput. **347**, 865–881 (2019)