P-Mixup: Improving Generalization Performance of Evolutionary Feature Construction with Pessimistic Vicinal Risk Minimization

Hengzhe Zhang[1(✉)], Qi Chen[1], Bing Xue[1], Wolfgang Banzhaf[2], and Mengjie Zhang[1]

[1] Centre for Data Science and Artificial Intelligence and School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
{hengzhe.zhang,qi.chen,bing.xue,mengjie.zhang}@ecs.vuw.ac.nz

[2] Department of Computer Science and Engineering in the College of Engineering and BEACON Center, Michigan State University, East Lansing, MI 48824, USA
banzhafw@msu.edu

Abstract. Genetic programming (GP)-based feature construction has achieved great success as an automated machine learning technique to improve learning performance. The key challenge in GP-based feature construction is that it is easy to overfit the training data. In supervised learning, unseen data usually lie in the vicinity of the training data and behave similar to the training data. However, a rugged model may make significantly different predictions, thus resulting in poor generalization performance. Here, we propose pessimistic vicinal risk minimization method to control overfitting in GP-based feature construction. The idea is to minimize the worst-case loss on vicinal examples of training instances, where vicinal examples are synthesized using an instance-wise mixing method. The experimental results on 58 datasets demonstrate that GP with the proposed overfitting control method clearly outperforms standard GP and seven other overfitting control methods for GP, validating the superiority of using pessimistic vicinal risk minimization to control overfitting in GP for feature construction.

Keywords: Genetic Programming · Evolutionary Machine Learning · Symbolic Regression · Feature Construction · Vicinal Risk Minimization

1 Introduction

Automated feature construction has become a well-studied topic in the recent domain of automated machine learning [1]. For a given dataset (X, Y), the goal of automated feature construction is to construct a set of features $\Phi(X)$ to improve the learning performance of algorithm $\mathcal{A}$ over the original feature space X. Among automated feature construction methods, genetic programming (GP)-based methods [1,2] are popular due to their flexible and variable-length

M. Affenzeller et al. (Eds.): PPSN 2024, LNCS 15148, pp. 201–220, 2024.
https://doi.org/10.1007/978-3-031-70055-2_13

representation, gradient-free optimization, and global search mechanism. These characteristics enable GP to explore a symbolic space to discover interpretable features and optimize non-differentiable losses, such as the L0-norm.

One challenge of existing GP-based feature construction methods is their tendency to overfit training data, especially in the presence of label noise or with limited samples. A widely used solution is to control model sizes; however, this approach has been challenged because the generalization of GP is often more about the semantics/behavior of the GP models rather than their sizes [3]. For instance, $sin(sin(x))$ and $x \times x$ have the same model size but differ significantly in their semantics, which can affect how well the models generalize.

Given the limitations of optimizing tree size to control overfitting, numerous works have proposed to use theoretical statistical learning techniques, such as Tikhonov regularization [4], VC dimension [5], and Rademacher complexity [6], to control the complexity of GP and improve generalization. Although these metrics have been found to be effective in traditional machine learning (ML), such as in support vector machines, their effectiveness is challenged by large deep neural networks with VC dimensions lower bounded by $\Omega(WL\log(W/L))$ for W parameters and L layers [7], which is usually large for modern neural networks. Thus, these metrics may not be suitable optimization objectives for controlling overfitting [8].

In recent years, vicinal risk minimization (VRM) has achieved significant success in controlling overfitting in deep learning [9]. For a model f and a loss function $\mathcal{L}$, a general loss is defined as follows:

$$\mathcal{L}(f) = \int \mathcal{L}\left(f(x), y\right)\ dP(x, y) \tag{1}$$

where $f(x)$ represents the prediction of the model on sample x, and $P(x, y)$ represents the probability of sampling the pair (x, y) from the underlying data distribution. The empirical risk considers only the loss on training samples, i.e., $\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(f\left(\mathbf{x}_i\right), y_i\right)$. In contrast, VRM considers both the training samples $x_i \in X$ and their neighborhoods $\mathbf{x} \in \mathbb{N}(x_i)$, known as vicinal samples. Formally, the vicinal risk is defined as follows:

$$\mathcal{VIC}(f) = \frac{1}{n}\sum_{i=1}^{n}\int \mathcal{L}\left(f(\mathbf{x}), y_i\right) dP_{x_i}(\mathbf{x}) \tag{2}$$

Here, $P_{x_i}(\mathbf{x})$ is a probability density function based on the distance between $\mathbf{x}$ and sample x_i, meaning that vicinal samples closer to the training samples have a higher occurrence probability. By optimizing such an objective, it is hoped that the model will not only perform well on training samples but also exhibit stable behavior on samples generated from the same distribution as the training samples.

However, in ML, one issue is that within a radius τ around a sample x, for all vicinity examples $x+\sigma$ subject to $||\sigma|| \leq \tau$, only specific perturbations σ may worsen the training loss, as observed in adversarial attacks [10]. Thus, this paper explores controlling overfitting by minimizing the vicinal risk in pessimistic cases to avoid overfitting, with the hope that the model achieves uniform stability [11]

across all possible unseen scenarios. Nonetheless, unlike adversarial training, the unseen sample should lie on the manifold of the underlying data distribution. Therefore, this paper focuses on mixup-synthesized samples [12], which are more likely to synthesize data that lie on the real manifold, as defined by Eq. (3):

$$x_{\text{mixup}} = \lambda \cdot x_a + (1 - \lambda) \cdot x_b, \tag{3}$$

where x_a and x_b are two nearby training instances, and $\lambda \in [0, 1]$ is a randomly sampled mixing ratio.

1.1 Goals

In this paper, we propose pessimistic vicinal risk as a regularizer to control overfitting, aiming for learned models to exhibit stable behavior on potential unseen data. The objectives of this paper are summarized as follows:

- To control overfitting, a pessimistic VRM (P-VRM) framework is proposed based on a multi-objective evolutionary feature construction algorithm.
- A theoretical analysis is conducted to demonstrate that P-VRM implicitly promotes local linearity between each training instance and its worst-case vicinal instance, thereby improving generalization performance.
- To validate the effectiveness of the proposed method, we compare P-VRM with seven complexity measures and standard GP for their effectiveness in controlling overfitting across 58 datasets.

2 Related Work

2.1 Overfitting Control Techniques in GP

Overfitting control techniques in GP can be divided into three main types. The first type is based on the probably approximately correct learning theory, using measures such as model size, Tikhonov regularization [4], VC dimension [5], and Rademacher complexity [6] to control the complexity of GP models. These approaches are based on solid theoretical foundations, yet their effectiveness is somewhat limited for deep learning-based feature learning techniques [8]. The second technique is ensemble learning, which is based on the bias-variance theory, either through homogeneous [13] or heterogeneous ensemble learning [14], assembling a set of GP models to enhance generalization performance. However, these ensemble models often lack interpretability. The third category includes several practical overfitting control techniques from ML that have been adapted for GP, such as early stopping [15], semi-supervised learning [16], random sampling [17], and soft targets [18]. However, these techniques often lack a strong theoretical foundation.

2.2 Vicinal Risk Minimization

For deep learning-based feature construction, VRM has been demonstrated to be effective in improving generalization performance [12]. VRM considers vicinal examples during training and offers benefits like Jacobian regularization and robustness against noisy targets [19]. However, VRM is rarely studied in GP, especially for regression or feature construction.

VRM in Deep Learning. In deep learning, particularly in computer vision, VRM is commonly achieved through data augmentation techniques like randomly cutting an image (CutOut) [20], mixing two images (MixUp) [12], mixing patches of images (CutMix) [21], and mixing the original image with an augmented image (AugMix) [22]. These approaches follow the idea of VRM, where the average loss on synthesized samples directs gradient descent. Nevertheless, adversarial attacks [23] demonstrate that very specific perturbations can drastically alter the loss, suggesting that optimizing the *average* vicinal risk might not be sufficient.

VRM in Genetic Programming. In GP, VRM has primarily been investigated for classification. For classification, defining vicinity samples is straightforward, as adding a small perturbation ϵ to an input x typically does not change the class label y. Therefore, VRM has been shown to be useful for learning decision trees [24] and GP classifiers [25]. However, this assumption may not hold for regression, where the output is continuous, which poses a challenge for employing VRM in GP for regression.

2.3 Evolutionary Feature Construction

Based on evaluation methods, evolutionary feature construction can be categorized into filter-based, wrapper-based, and embedded methods. Filter-based methods use general metrics like impurity [26] or information gain [27]. They are cost-effective but may yield sub-optimal accuracy. Wrapper-based methods evaluate features using a specific learning algorithm. For wrapper-based evolutionary feature construction, multi-tree GP has been widely used, including algorithms like M3GP [2], FEAT [1], and GP-GOMEA [28]. These methods can construct highly discriminative features for a specific learning algorithm but may lead to overfitting. Embedded methods integrate feature construction into the learning process, such as symbolic regression [29]. They provide a balance between training time and accuracy. This paper studies wrapper-based methods, with a focus on addressing generalization issues.

3 Proposed Algorithm Framework

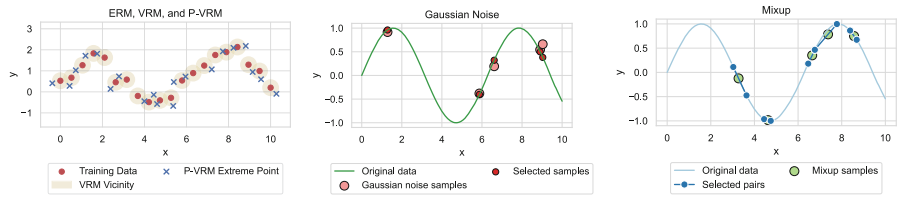
3.1 General Framework

In this paper, a multi-tree GP, denoted as Φ, is used to represent multiple features $\phi_1, \ldots, \phi_m$, where m is dynamically evolved for different GP individuals. These constructed features are employed to transform original features X into the constructed features $\Phi(X)$, which are then fed into a linear regression model LM to make predictions. This work follows the general framework of evolutionary feature construction, incorporating additional steps such as vicinal data synthesis, vicinal risk estimation, and archive maintenance[1]. The key steps are as follows:

[1] Source Code: https://github.com/hengzhe-zhang/EvolutionaryForest/blob/master/experiment/methods/P_VRM_GP.py.

- **Population Initialization**: At this stage, a population of GP individuals is randomly initialized. For each individual, a GP tree is initialized using either the full or grow method with equal probability, i.e., the ramped-half-and-half method. The number of trees in an individual can be increased through mutation operators later during the evolutionary process.
- **Parent Selection**: Automatic ϵ lexicase selection [30] is used for parent selection. Lexicase selection iteratively chooses an instance k to set a threshold for eliminating individuals with poor performance. The threshold is defined as $\mathcal{L}_k(p) < \min_{p' \in P} \mathcal{L}_k(p') + \epsilon_k$, where $\min_{p' \in P} \mathcal{L}_k(p')$ is the minimum loss on instance k, and ϵ_k is the median absolute deviation. Those with a loss larger than the threshold are eliminated from the candidate parent pool. This process repeats until one individual remains and is selected as the parent. The lexicase selection operator is applied $|P|$ times to select $|P|$ parents.
- **Offspring Generation**: Offspring are generated using random subtree crossover and mutation operators. Additionally, random tree addition and deletion operators [2] are used to dynamically increase or decrease the number of trees m in an individual Φ.
- **Solution Evaluation**: The evaluation of candidate solutions involves two objectives: pessimistic vicinal risk based on the mixup strategy, i.e., $O_1(\Phi)$, and leave-one-out cross-validation loss, i.e., $O_2(\Phi)$. These two objectives are obtained based on four functions:
  - **Vicinal Data Synthesis**: First, vicinal data $\tilde{X}$ is synthesized using the instance-wise mixup strategy outlined in Sect. 3.3.
  - **Feature Transformation**: For each individual Φ, both the original data X and the vicinal data $\tilde{X}$ are transformed using all GP trees ϕ within the individual Φ to form $\Phi(X)$ and $\Phi(\tilde{X})$, respectively.
  - **Pessimistic Vicinal Risk Estimation ($O_1(\Phi)$)**: To evaluate the performance of the constructed features on vicinal data, first, the original data $\Phi(X)$ is used to train a linear regression model LM. Then, the pessimistic vicinal risk is calculated by applying the linear model LM to the vicinal data $\Phi(\tilde{X})$ according to the vicinal risk estimation method detailed in Sect. 3.4.
  - **Leave-one-out Cross-validation ($O_2(\Phi)$)**: This objective is obtained by evaluating the leave-one-out cross-validation loss on the constructed features $\Phi(X)$ using a linear model LM.
- **Survival/Environmental Selection**: After evaluation, non-dominated sorting with crowding distance [31] is used to select promising candidates for the next generation. This process reduces the set of parents and offspring, initially sized at $2|P|$ individuals, back to $|P|$ individuals.
- **Archive Maintenance**: At the end of each generation, the model with the lowest vicinal risk $O_1(\Phi)$ is stored in an external archive as the final prediction model.

The processes of solution evaluation, parent selection, offspring generation, environmental selection, and archive maintenance are repeated until a termination criterion is met.



(a) ERM/VRM/P-VRM. (b) Gaussian Synthesis. (c) MixUp Synthesis.

Fig. 1. (a) Comparison between empirical risk minimization (ERM), VRM, and P-VRM using the Gaussian synthesis strategy. (b, c) Comparison between the Gaussian synthesis strategy and the mixup synthesis strategy.

3.2 P-VRM

In this paper, we consider the loss in the worst-case scenario over all vicinal examples, thus employing "pessimistic" VRM. Formally, pessimistic vicinal risk is defined as:

$$\mathcal{V}(f) = \frac{1}{n}\sum_{i=1}^{n} \max_{x \in \mathbb{N}(x_i)} \mathcal{L}\left(f(\mathbf{x}), y_i\right), \tag{4}$$

where $\mathbb{N}(x_i)$ represents the neighborhood region of point x_i. By optimizing Eq. (4), we aim for the model to achieve stable behavior even in the presence of the worst perturbations around each data point. Figure 1a illustrates the differences between ERM, VRM, and P-VRM. Typically, vicinal samples used to calculate vicinal risk are synthesized by adding Gaussian noise $\mathcal{N}(0, \sigma)$ to existing points $x \in X$, as shown in Fig. 1b. In this paper, our mixup is based on synthesizing a mixture of a sample $x_i \in X$ with its neighboring point $x_j \in \mathbb{N}(x_i)$ with a randomly sampled ratio $\lambda \sim \text{Beta}(\alpha, \beta)$. An example of mixup-based vicinal example synthesis is shown in Fig. 1c. Intuitively, mixup can be understood as a linear interpolation between two nearby points, thereby synthesizing data without disrupting the data manifold.

3.3 Instance-Wise Vicinal Data Synthesis

In the traditional mixup framework, vicinal data are generated by mixing two randomly selected examples from the original dataset [12]. This can lead to some examples not being selected, while others are selected multiple times, resulting in inaccurate and biased vicinal risk estimation. To achieve accurate VRM, we propose an instance-wise mixup strategy to ensure that all examples are selected exactly the same number of times. Specifically, for each instance $x_a \in X$, we mix it with K nearby instances $x_{b,1}, \ldots, x_{b,K}$ to generate vicinal samples x_{mixup}. There are two key questions in this approach: how to select the K nearby instances $x_b \in \mathbb{N}(x_a)$, and how to mix the instance x_a with x_b.

Kernel-Based Sample Selection. To synthesize data, a nearby sample $x_b \in \mathbb{N}(x_a)$ needs to be selected based on sample x_a. Defining the vicinity in feature space is challenging because it is difficult to capture the underlying data manifold that represents the true data distribution. Therefore, we

define vicinity based on labels, y_a and y_b. Specifically, to determine the similarity between samples (x_a, y_a) and (x_b, y_b), we use a Gaussian kernel function: $K(y_b, y_a) = e^{-\gamma \|y_b - y_a\|^2}$ [32]. In each round of vicinal sample generation, distances are first normalized to create a probability distribution, i.e., $P(y_b|y_a) = \frac{K(y_b, y_a)}{\sum_{y_i \in Y} K(y_i, y_a)}$, which is then used to sample instances based on their proximity to sample (x_a, y_a). Based on this probability distribution, K samples are selected to generate vicinal samples.

Vicinal Data Synthesis. For a pair of samples $\{(x_a, y_a), (x_b, y_b)\}$, the standard mixup equation is $x_{\text{mixup}} = \lambda \cdot x_a + (1-\lambda) \cdot x_b$ [12], where λ is the mixing ratio sampled from a Beta distribution with parameters $\alpha = \beta$ [12]. The probability density function of the Beta distribution is given by: $\text{Beta}(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}$, where B is the beta function, $B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\ \text{dt}$. The choice of the Beta distribution follows convention [12] and is motivated by its property of ensuring the sampled mixing ratio λ falls within the interval $[0, 1]$ with arbitrary parameter settings.

However, for P-VRM, the vicinal sample x_{mixup} of instance x_a should contain more of x_a than x_b. Thus, the instance-wise mixup is defined as in Eq. (5),

$$x_{\text{mixup}} = (0.5 + |\lambda - 0.5|) \cdot x_a + (0.5 - |\lambda - 0.5|) \cdot x_b, \tag{5}$$

where, by setting the sample ratio for x_a to be $0.5 + |\lambda - 0.5|$, we ensure that the ratio of x_a in x_{mixup} is greater than or equal to 0.5. The label of the synthesized sample y_{mixup} is synthesized using the same equation and ratio. The instance-wise mixup equation can be viewed as adding a perturbation to the sample (x_a, y_a):

$$x_{\text{mixup}} = x_a + (0.5 - |\lambda - 0.5|) \cdot (x_b - x_a), \tag{6}$$

$$y_{\text{mixup}} = y_a + (0.5 - |\lambda - 0.5|) \cdot (y_b - y_a). \tag{7}$$

Theorem 1. *For a predictive model $f : X \to Y$, pessimistic mixup implicitly promotes local linearity around each sample $x_a \in X$ by optimizing the objective:* $\max_{\lambda, (x_b, y_b) \in \mathbb{N}(x_a)} (0.5 - |\lambda - 0.5|)^2 (y_b - y_a - \nabla f(x_a)^\top (x_b - x_a))^2$, *where λ is a mixup ratio drawn from a Beta distribution Beta(α, β), and $\mathbb{N}(x_a)$ denotes the neighborhood of x_a.*

Proof: The prediction of a model for a mixup sample, $f(x_{\text{mixup}})$, can be approximated using a first-order Taylor expansion around the sample point x_a as

$$f(x_{\text{mixup}}) \approx f(x_a) + \nabla f(x_a)^\top (x_{\text{mixup}} - x_a) + \mathcal{O}((x_{\text{mixup}} - x_a)^2), \tag{8}$$

where $f(x_a)$ is the model prediction on the original sample x_a, $\nabla f(x_a)$ is the gradient of the model prediction with respect to the input at x_a. The term $\nabla f(x_a)^\top (x_{\text{mixup}} - x_a)$ represents the linear approximation of the change in the model prediction due to the perturbation $x_{\text{mixup}} - x_a$, and $\mathcal{O}((x_{\text{mixup}} - x_a)^2)$

represents the higher-order remaining terms. For conciseness, let $\delta = (0.5 - |\lambda - 0.5|)$. Substituting Eq. (8) into the vicinal risk loss $(y_{\text{mixup}} - f(x_{\text{mixup}}))^2$, we have:

$$
\begin{aligned}
&(y_{\text{mixup}} - f(x_{\text{mixup}}))^2 && (9)\\
&\approx (y_a + \delta(y_b - y_a) - f(x_a) - \nabla f(x_a)^\top (x_{\text{mixup}} - x_a))^2 && (10)\\
&= (y_a + \delta(y_b - y_a) - f(x_a) - \nabla f(x_a)^\top ((x_a + \delta(x_b - x_a)) - x_a))^2 && (11)\\
&= ((y_a - f(x_a)) + \delta(y_b - y_a - \nabla f(x_a)^\top (x_b - x_a)))^2 && (12)\\
&\leq 2(y_a - f(x_a))^2 + 2\delta^2 (y_b - y_a - \nabla f(x_a)^\top (x_b - x_a))^2. && (13)
\end{aligned}
$$

This bound indicates that minimizing both the mean square error and the term $\delta^2(y_b - y_a - \nabla f(x_a)^\top (x_b - x_a))^2$ can achieve the same effect as minimizing the mixup-based vicinal risk. To investigate the regularization effect of P-VRM based on mixup, assuming a constant training error $(y_a - f(x_a))^2$, the optimization process focuses on the following objective:

$$
\mathcal{V} = \max_{\lambda, (x_b, y_b)} (0.5 - |\lambda - 0.5|)^2 (y_b - y_a - \nabla f(x_a)^\top (x_b - x_a))^2. \tag{14}
$$

Discussion: To find the optimal model f^* that minimizes $\mathcal{V}$ for a specific instance x_b^*, y_b^*, the optimal gradient can be determined by setting $\frac{\partial \mathcal{V}}{\partial \nabla f(x_a)} = 0$:

$$
\frac{\partial \mathcal{V}}{\partial \nabla f(x_a)} = 2\delta^2 (\nabla f(x_a)^\top (x_b^* - x_a) - (y_b^* - y_a))(x_b^* - x_a) = 0. \tag{15}
$$

The optimal gradient is achieved when the gradient of the function $\nabla f(x_a)$ is parallel to the vector difference $x_b^* - x_a$, scaled by the difference $y_b^* - y_a$, i.e.,$\nabla f(x_a)^\top (x_b^* - x_a) = y_b^* - y_a$. Ideally, this alignment should occur for every instance x_b within a local vicinity of x_a to minimize the pessimistic vicinal risk. In practice, this implies that P-VRM encourages the model f to maintain a constant gradient, i.e., to approximate a linear function on the worst-case instance lying within the linear manifold, which is formed by each sample x_a and its neighboring points x_b, thereby improving the generalization performance of the model f.

3.4 Pessimistic Vicinal Risk Estimation

The process of estimating the pessimistic vicinal risk is outlined in Algorithm 1. To ensure reliable estimation, vicinal risk is estimated over K rounds for each sample x, and the maximum loss on each synthesized sample x_{mixup} is considered as the pessimistic risk. Subsequently, the final pessimistic risk of the GP individual Φ is computed as the average of the pessimistic risks across all instances $x \in X$. In each iteration, two key components are worth highlighting:

- **Cached Vicinal Data (Line 4)**: The mixup data $(X_{\text{mixup}}, Y_{\text{mixup}})$ are regenerated for the K rounds but generated only once for each round and cached in memory for consistency across different individuals. There are two

Algorithm 1. Pessimistic Vicinal Risk Estimation

Require: A GP Tree Φ, Input X, Target Output Y, Linear Model LM, Gaussian Noise Standard Deviation σ, Number of Iterations K

1: Initialize Vicinal Risk: $\mathcal{V} \leftarrow 0$
2: $\Phi(X) \leftarrow$ Feature Construction (X, Φ)
3: **for** $k = 1, \ldots, K$ **do**
4: $\quad X_{\text{mixup}}, Y_{\text{mixup}} \leftarrow$ Instance-wise Mixup (X, Y, k) ▷ Cached
5: $\quad \Phi(X_{\text{mixup}}) \leftarrow$ Feature Construction $(X_{\text{mixup}}, \Phi, \sigma)$
6: $\quad \hat{Y} \leftarrow$ Prediction$(LM, \Phi(X_{\text{mixup}}))$
7: $\quad$ **for** $i = 1, \ldots, N$ **do**
8: $\quad\quad \mathcal{V}_i \leftarrow \max(\mathcal{V}_i, (\hat{y}_i - y_{\text{mixup}})^2)$

Ensure: Pessimistic Vicinal Risk $\sum_{i=1}^{N} \mathcal{V}_i / N$

benefits of using the caching strategy. Firstly, it can save computational costs for synthesizing data. Secondly, it ensures a fair comparison on vicinal risk across individuals by providing a shared reference for evaluations.

– **Pessimistic Risk Estimation (Lines 7-8)**: After computing vicinal risk, $(\hat{y}_i - y_{\text{mixup}})^2$, it is compared to the historically worst vicinal risk on example i, denoted as $\mathcal{V}_i$. The historical worst vicinal risk $\mathcal{V}_i$ is updated if the new risk exceeds the previously recorded worst risk.

4 Experimental Settings

4.1 Datasets

The datasets used in this paper are from the Penn Machine Learning Benchmark (PMLB) [33]. There are 120 "black-box" datasets in PMLB; after excluding datasets synthesized by Friedman functions, 58 datasets remain, which are used for experiments.

4.2 Comparison Methods

The comparison methods include standard GP and GP with seven common complexity measures to control overfitting:

– **Standard GP without Regularization**: This is the standard GP with leave-one-out cross-validation loss as the objective.
– **Parsimony Pressure (PP)** [34]: PP is based on Occam's razor principle. The number of nodes in GP trees within an individual is considered as the complexity measure.
– **Tikhonov Regularization (TK)** [4]: This paper only focuses on zero-order TK due to its simplicity and effectiveness [4], where zero-order TK of model f is defined as $||f(X)||$. The aim of zero-order TK is to prevent extreme predictions.

- **Grand Complexity (GC)** [4]: GC is a combination of parsimony pressure and Tikhonov regularization, where the dominance relationship between these two objectives is used as an optimization objective.
- **Rademacher Complexity (RC)** [6]: RC measures a model's ability to fit a given dataset with randomly labeled data. It is formally defined as: $\mathrm{R}_n(\mathcal{L}) = \mathbb{E}\left[\sup_{l\in\mathcal{L}} \frac{1}{n}\sum_{i=1}^{n} \sigma_i l\left(x_i, y_i\right)\right]$, where $\sigma_i \in \{-1, 1\}$ is a random variable.
- **Correlation between Input and Output Distances (IODC)** [18]: IODC measures the correlation between input pairwise distances and output pairwise distances to evaluate the smoothness of a model. It is defined as: $\mathrm{IODC}(\Phi) = \frac{\mathrm{Cov(I,O)}}{\sigma_\mathrm{I}\sigma_\mathrm{O}}$. A higher IODC value indicates a smoother model.
- **Weighted Maximum Information Coefficient between Residuals and Variables (WCRV)** [35]: WCRV seeks to minimize two factors: the correlation between highly informative features and the residual R, and the selection of uninformative features. It is formally defined as: $\mathrm{WCRV}\left(\Phi\right) = \sum_{\mathrm{MIC}_{x^k,Y}\geq mv} \mathrm{MIC}_{x^k,Y} \times \mathrm{MIC}_{x^k,R} + \sum_{\mathrm{MIC}_{x^k,Y}<mv} \left(1 - \mathrm{MIC}_{x^k,Y}\right)$.
- **VRM**: VRM refers to the conventional mixup-based VRM [12] that is commonly used in deep learning, which focuses on minimizing the average loss across all vicinal samples $\mathbb{N}(x)$, rather than just the worst-case samples. The objective function is formally defined as: $\mathcal{V}(f) = \frac{1}{n}\sum_{i=1}^{n}\sum_{x\in\mathbb{N}(x_i)} \mathcal{L}\left(f(\mathbf{x}), y_i\right) dP_{\mathbf{x}_i}(\mathbf{x})$, which corresponds to Eq. (2).

To ensure fair comparisons, we evaluate these methods using the same multi-objective evolution framework as this paper. Standard GP adheres to this framework without a survival selection operator. In VRM, the model with the lowest vicinal risk is chosen as the final model. For other benchmark methods, to balance training accuracy and model complexity, the minimum Manhattan distance-based (MMD) knee point is used to select the final solution from the non-dominated set of trade-off solutions for prediction [36]. Given the different scales of complexity measures compared to mean squared error in the benchmark methods, we first normalize the two objectives, denoted by $\mathcal{O}_1(\Phi)$ and $\mathcal{O}_2(\Phi)$. Then, the individual with the smallest sum of the two objectives, $\mathcal{O}_1(\Phi)+\mathcal{O}_2(\Phi)$, is selected as the final model in the MMD method.

4.3 Parameter Settings

The parameter settings follow the convention in GP, as shown in Table 1. A large crossover rate and a small mutation rate are used to encourage the exchange of building blocks. To prevent division by zero, we use the analytical quotient, defined as $AQ(a, b) = \frac{a}{\sqrt{1+b^2}}$ [37]. Additionally, the analytical logarithm, which is defined as $Log(a) = log(\sqrt{1 + a^2})$, is used to replace the traditional logarithm operator.

4.4 Evaluation Protocol

For each experiment, each baseline method is executed on each dataset with 30 different random seeds [38]. In each round, the dataset is randomly split into a training set with 100 points, and all remaining data serve as the test data.

Table 1. GP Parameter Settings

Parameter	Value	Parameter	Value
Maximum Population Size	200	Maximum Number of Trees	10
Number of Generations	100	Elitism (Number of Individuals)	1
Crossover Rate	0.9	Bandwidth of Gaussian Kernel	0.5
Mutation Rate	0.1	β of Beta Distribution	1
Tree Addition Rate	0.5	Iterations of Risk Estimation (K)	10
Tree Deletion Rate	0.5	Functions	+, -, *, AQ, Abs, Sqrt, Neg, Log, Max, Min, Sin, Cos, Square
Initial Tree Depth	0-3		
Maximum Tree Depth	10		
Initial Number of Trees	1		

Table 2. Statistical comparisons of test R^2 scores across various overfitting control strategies. ("+", "∼", and "-" indicate that the method in a row performs better than, similarly to, or worse than the method in a column, respectively.)

	VRM	PP	RC	GC
P-VRM	22(+)/29(∼)/7(-)	22(+)/31(∼)/5(-)	46(+)/10(∼)/2(-)	23(+)/32(∼)/3(-)
VRM	—	11(+)/38(∼)/9(-)	38(+)/10(∼)/10(-)	16(+)/29(∼)/13(-)
PP	—	—	38(+)/13(∼)/7(-)	15(+)/34(∼)/9(-)
RC	—	—	—	5(+)/10(∼)/43(-)
	IODC	**TK**	**WCRV**	**Standard GP**
P-VRM	31(+)/24(∼)/3(-)	44(+)/12(∼)/2(-)	38(+)/18(∼)/2(-)	31(+)/21(∼)/6(-)
VRM	24(+)/25(∼)/9(-)	28(+)/29(∼)/1(-)	27(+)/23(∼)/8(-)	22(+)/34(∼)/2(-)
PP	23(+)/29(∼)/6(-)	25(+)/31(∼)/2(-)	26(+)/26(∼)/6(-)	25(+)/25(∼)/8(-)
RC	5(+)/21(∼)/32(-)	5(+)/20(∼)/33(-)	11(+)/17(∼)/30(-)	13(+)/14(∼)/31(-)
GC	25(+)/26(∼)/7(-)	23(+)/33(∼)/2(-)	25(+)/30(∼)/3(-)	24(+)/21(∼)/13(-)
IODC	—	19(+)/24(∼)/15(-)	15(+)/30(∼)/13(-)	21(+)/16(∼)/21(-)
TK	—	—	11(+)/33(∼)/14(-)	10(+)/27(∼)/21(-)
WCRV	—	—	—	17(+)/22(∼)/19(-)

This setting is used in existing GP literature to simulate the scenario of limited samples [39]. For datasets with fewer than 100 points, a 50%-50% training-test split is used to ensure enough training and test data. For the evaluation metric, R^2 scores are used to account for different scales across datasets. The Wilcoxon signed-rank statistical comparison, at a significance level of 0.05, is used to compare the statistical differences between pairs of algorithms [40].

5 Experimental Results

5.1 Comparisons of Test R^2 Scores

General Analysis. The results of R^2 scores are presented in Table 2 and Fig. 2. All complexity measures effectively control overfitting to varying degrees. Even

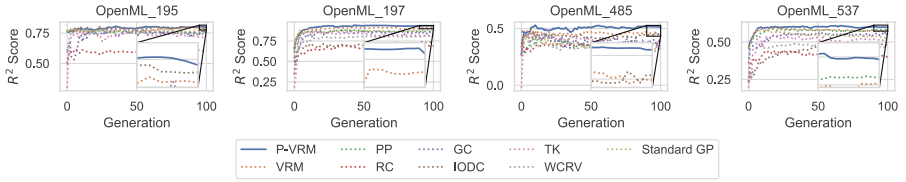


Fig. 2. Evolutionary plots of the *test* R^2 *scores* for different complexity measures.

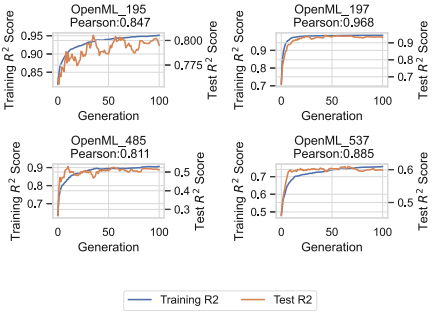


Fig. 3. Evolutionary plots of the *training and test* R^2 *scores* for P-VRM.

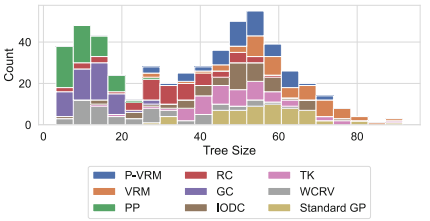


Fig. 4. Distribution of *tree sizes* across 58 datasets when optimizing different complexity measures.

the least effective method, TK, reduces overfitting in 10 datasets. However, P-VRM is particularly successful, outperforming standard GP on 31 datasets and only underperforming on 6 datasets. In this section, we mainly focus on comparing P-VRM with VRM and PP, as P-VRM surpasses the other methods on at least 23 datasets.

P-VRM vs. Parsimony Pressure. In GP, tree size has been widely used to control overfitting. For GP-based evolutionary feature construction, the results in Table 2 show that tree size remains a competitive measure for controlling overfitting [41], outperforming standard GP on 25 datasets and only performing worse on 8 datasets. However, tree size does not account for the semantics of a GP tree; it cannot distinguish between the complexity of x^2 and $sin(x)$. The results in Table 2 indicate that merely controlling the tree size is insufficient for controlling overfitting. Optimizing VRM outperforms optimizing tree size on 22 datasets while only underperforming on 5 datasets, and is similar on 31 datasets. This confirms that considering the complexity from the perspective of the semantics of GP trees, especially pessimistic vicinal risk, leads to better generalization performance.

P-VRM vs. VRM. The only difference between P-VRM and VRM is that P-VRM focuses on the worst-case vicinal risk, while VRM considers the aver-

age risk over all vicinal examples. Experimental results in Table 2 show that P-VRM significantly outperforms VRM on 22 datasets and only performs worse on 7 datasets. This suggests that pessimistic vicinal risk is more effective for controlling overfitting than average vicinal risk.

5.2 Comparisons of Training R^2 Scores

Statistical comparisons of training R^2 results are presented in Table 3. The results reveal that P-VRM is a stronger regularizer than VRM but weaker than PP. P-VRM significantly underperforms in training R^2 compared to VRM on 44 datasets out of 58, yet only underperforms compared to PP on 13 datasets out of 58. Given the suboptimal results of VRM and PP on test R^2 scores, this suggests that effective overfitting control requires a carefully designed inductive bias towards complexity regularization. Both overly strong and overly weak regularization can negatively affect test performance. Figure 3 further illustrates the evolutionary plots of training and test R^2 for the P-VRM method. The positive Pearson correlations between training and test R^2 scores further affirm that P-VRM is an effective measure for controlling overfitting.

Table 3. Statistical comparisons of training R^2 scores across various overfitting control strategies.

	VRM	PP	RC	GC
P-VRM	0(+)/14(∼)/44(-)	28(+)/17(∼)/13(-)	56(+)/2(∼)/0(-)	46(+)/10(∼)/2(-)
VRM	—	48(+)/10(∼)/0(-)	58(+)/0(∼)/0(-)	56(+)/2(∼)/0(-)
PP	—	—	58(+)/0(∼)/0(-)	46(+)/12(∼)/0(-)
RC	—	—	—	1(+)/4(∼)/53(-)
	IODC	**TK**	**WCRV**	**Standard GP**
P-VRM	37(+)/15(∼)/6(-)	36(+)/15(∼)/7(-)	39(+)/13(∼)/6(-)	1(+)/13(∼)/44(-)
VRM	53(+)/5(∼)/0(-)	53(+)/4(∼)/1(-)	48(+)/8(∼)/2(-)	2(+)/38(∼)/18(-)
PP	29(+)/23(∼)/6(-)	32(+)/20(∼)/6(-)	38(+)/15(∼)/5(-)	0(+)/3(∼)/55(-)
RC	1(+)/4(∼)/53(-)	0(+)/1(∼)/57(-)	2(+)/9(∼)/47(-)	0(+)/0(∼)/58(-)
GC	12(+)/27(∼)/19(-)	11(+)/26(∼)/21(-)	23(+)/25(∼)/10(-)	0(+)/1(∼)/57(-)
IODC	—	16(+)/19(∼)/23(-)	27(+)/19(∼)/12(-)	0(+)/0(∼)/58(-)
TK	—	—	31(+)/12(∼)/15(-)	0(+)/4(∼)/54(-)
WCRV	—	—	—	0(+)/2(∼)/56(-)

5.3 Comparisons of Tree Sizes

The comparisons of tree sizes are presented in Fig. 4. The results indicate that P-VRM exhibits a similar magnitude of tree sizes compared to standard GP.

This confirms that improving generalization performance is more about regularizing semantics rather than tree size [3]. Thus, rather than simply applying the traditional Occam's Razor philosophy, which suggests controlling tree sizes to prevent overfitting, it is more effective to control functional complexity in GP.

In addition to comparisons of tree sizes, training time for different complexity measures is also compared, as provided in Appendix A of the supplementary material.

6 Further Analysis

In this section, we further analyze the advantage of P-VRM by considering two questions:

- Is it useful to adopt mixup in place of Gaussian perturbation to synthesize vicinal samples?
- Is the proposed vicinal risk also helpful when using Gaussian perturbation to synthesize vicinal samples?

To answer these two questions, we compare the proposed P-VRM with two variants under the same evolutionary feature construction framework:

- GVRM: This variant introduces a small Gaussian noise, $\mathcal{N}(0, 0.1)$, to synthesize vicinal data [25], while maintaining the original labels.
- P-GVRM: Similar to GVRM, but this variant focuses on pessimistic vicinal risk rather than average vicinal risk.

Effectiveness of MixUp. The comparisons of test R^2 scores between P-VRM and P-GVRM are presented in Table 4. As indicated by the R^2 scores, P-VRM outperforms P-GVRM on 23 datasets and performs worse on only 8 datasets, highlighting the effectiveness of using mixup instead of Gaussian noise [25] to synthesize vicinal examples. The inferiority of P-GVRM is because Gaussian noise may synthesize some examples that do not conform to the data manifold, whereas mixup synthesizes samples more closely aligned with the data manifold. Therefore, using mixup to synthesize vicinal data is more effective.

Effectiveness of Pessimistic Vicinal Risk. Another interesting finding is that P-GVRM outperforms GVRM on 22 datasets and is similar on 34 datasets. This suggests that even when Gaussian noise is used to synthesize vicinal examples, employing pessimistic vicinal risk benefits evolutionary feature construction in controlling overfitting.

Table 4. Statistical comparisons of R^2 scores using mixup or Gaussian noise for synthesizing vicinal samples.

	P-GVRM	GVRM
P-VRM	23(+)/27(∼)/8(-)	30(+)/21(∼)/7(-)
P-GVRM	—	22(+)/34(∼)/2(-)

7 Conclusions

In this paper, we propose P-VRM to reduce overfitting and improve the generalization performance of GP-based evolutionary feature construction. P-VRM minimizes the vicinal risk on the worst-case vicinal examples, and theoretical analysis shows that P-VRM encourages the model to maintain a constant gradient and approximate a linear function between each instance and its worst-case vicinal instance, thereby improving generalization performance. Experiments on 58 datasets show that P-VRM effectively controls overfitting compared to seven other complexity measures, including classical VRM and parsimony pressure. An analysis of model size suggests that controlling overfitting is more about controlling semantic complexity rather than merely model size. Further comparison between mixup-based vicinal data synthesis and Gaussian noise-based synthesis reveals that using mixup to synthesize vicinal data is more effective in controlling overfitting.

For future work, exploring the effectiveness of other strategies, such as CutMix [21], to further control overfitting in GP-based feature construction is worthwhile. Additionally, investigating the efficacy of the proposed P-VRM in broader classes of symbolic regression techniques, including those based on reinforcement learning [42] or transformer [43] techniques, is also promising.

Appendices

A Comparisons of Training Time

The comparisons of training time are presented in Fig. 5. It shows the distribution of median training time on each dataset over 30 random seeds. Experimental results indicate that P-VRM requires more time than standard GP and simple complexity measures, such as parsimony pressure. Specifically, P-VRM takes 1481 seconds in median, while standard GP takes 320 seconds in median. The standard deviations of P-VRM and standard GP are 121 and 28 seconds, respectively. The reason P-VRM is more time-consuming is that it needs to compute the semantics of GP trees on vicinal data, whereas parsimony pressure merely examines the syntax of GP trees. Hence, P-VRM is more time-consuming. However, the training time for P-VRM remains within a reasonable range. Also, the overfitting issue cannot be solved by simply increasing the number of iterations or training time. Therefore, the increase in training time of P-VRM to address the overfitting issue is still acceptable.

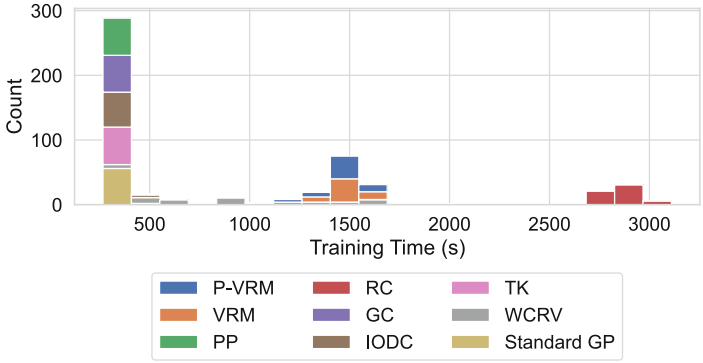


Fig. 5. Distribution of *training time* across the 58 datasets when optimizing different complexity measures.

B Post-hoc Parameter Analysis

P-VRM involves two hyperparameters. The first is the bandwidth of the Gaussian kernel used to sample instances x_b based on x_a, and the second is the α, β values in the beta distribution used to determine the mixing ratio. This section examines the sensitivity of these parameters to different settings.

B.1 Bandwidth

In P-VRM, a Gaussian kernel samples a paired instance y_b based on y_a as follows:

$$K(y_b, y_a) = e^{-\gamma \|y_b - y_a\|^2} \tag{16}$$

Here, the bandwidth γ, is a critical parameter that defines the scope of the vicinity for each point. A smaller bandwidth implies a larger vicinity, while a larger bandwidth suggests that only proximate samples are likely to synthesize mixup instances. This study considers three different bandwidths, and the kernel values $K(y_b, y_a)$ for various combinations of bandwidth γ and instances y_a, y_b are depicted in Fig. 6.

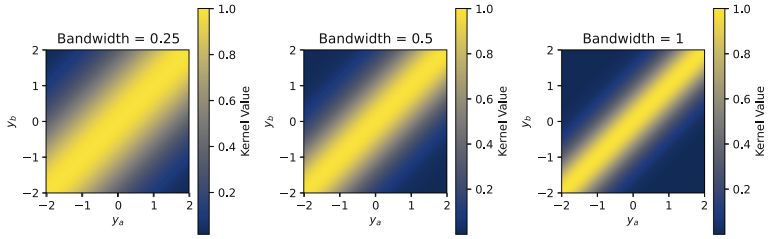


Fig. 6. Kernel values for different bandwidths γ and instances y_a, y_b.

Experimental results, as shown in Table 5, indicate that the proposed method is robust to changes in bandwidth settings, as changing from 0.5 to 0.25 has no significant impact on 54 out of the 58 datasets.

Table 5. Statistical comparisons of test R^2 scores across various bandwidths.

Bandwidth	0.25	0.5
1.0	3(+)/53(∼)/2(-)	0(+)/58(∼)/0(-)
0.25	—	1(+)/54(∼)/3(-)

B.2 Alpha/Beta

In mixup, the mixing ratio λ is sampled from a Beta distribution $\text{Beta}(\alpha, \beta)$. The probability density function of the Beta distribution is defined as:

$$p(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad 0 \leq x \leq 1, \tag{17}$$

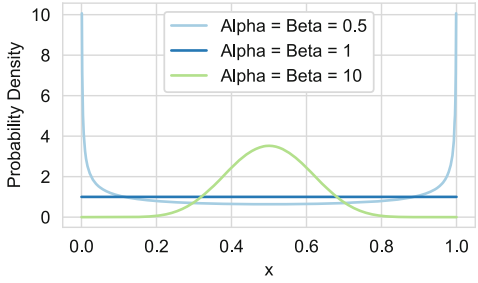


Fig. 7. Probability density functions of the Beta distribution for different α values.

where $B(\alpha, \beta)$ denotes the Beta function. For simplicity, α is set to be equal to β in mixup, i.e., $\alpha = \beta$. In this section, three different settings of α are examined: 0.5, 1, and 10, and their probability density functions are illustrated in Fig. 7.

Table 6 presents a comparison of test R^2 scores obtained using different α values in the Beta distribution for sampling the mixing ratio. The experimental results indicate that an α value of 1 yields reasonably good results. This suggests that, in practice, a uniform distribution can be used in P-Mixup to speed up the sampling process, as the uniform distribution is more straightforward to sample from than the Beta distribution.

Table 6. Statistical comparisons of test R^2 scores across various α values.

α	10.0	1.0
0.5	6(+)/46(∼)/6(-)	4(+)/54(∼)/0(-)
10.0	—	4(+)/49(∼)/5(-)

References

1. La Cava, W., Moore, J.H.: Learning feature spaces for regression with genetic programming. Genet. Program Evolvable Mach. **21**, 433–467 (2020)
2. Muñoz, L., Trujillo, L., Silva, S., Castelli, M., Vanneschi, L.: Evolving multidimensional transformations for symbolic regression with M3GP. Memetic Comput. **11**, 111–126 (2019)
3. Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complexity in genetic programming. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 877–884 (2010)
4. Ni, J., Rockett, P.: Tikhonov regularization as a complexity measure in multiobjective genetic programming. IEEE Trans. Evol. Comput. **19**(2), 157–166 (2014)
5. Chen, Q., Zhang, M., Xue, B.: Structural risk minimization-driven genetic programming for enhancing generalization in symbolic regression. IEEE Trans. Evol. Comput. **23**(4), 703–717 (2018)
6. Chen, Q., Xue, B., Zhang, M.: Rademacher complexity for enhancing the generalization of genetic programming for symbolic regression. IEEE Trans. Cybernetics **52**(4), 2382–2395 (2022)
7. Bartlett, P.L., Harvey, N., Liaw, C., Mehrabian, A.: Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. J. Mach. Learn. Res. **20**(1), 2285–2301 (2019)
8. Jiang, Y., Neyshabur, B., Mobahi, H., Krishnan, D., Bengio, S.: Fantastic generalization measures and where to find them. In: International Conference on Learning Representations (2020). https://openreview.net/forum?id=SJgIPJBFvH
9. Pinto, F., Yang, H., Lim, S.N., Torr, P., Dokania, P.: Using mixup as a regularizer can surprisingly improve accuracy & out-of-distribution robustness. Adv. Neural. Inf. Process. Syst. **35**, 14608–14622 (2022)
10. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May, 2015, Conference Track Proceedings (2015) http://arxiv.org/abs/1412.6572
11. Bousquet, O., Klochkov, Y., Zhivotovskiy, N.: Sharper bounds for uniformly stable algorithms. In: Conference on Learning Theory, pp. 610–626. PMLR (2020)
12. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: International Conference on Learning Representations (2018)
13. Zhang, H., Zhou, A., Zhang, H.: An evolutionary forest for regression. IEEE Trans. Evol. Comput. **26**(4), 735–749 (2021)
14. Zhang, H., Zhou, A., Chen, Q., Xue, B., Zhang, M.: SR-Forest: a genetic programming based heterogeneous ensemble learning method. IEEE Trans. Evolutionary Comput. (2023)
15. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: Early stopping criteria to counteract overfitting in genetic programming. In: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, pp. 203–204 (2011)

16. Silva, S., Vanneschi, L., Cabral, A.I., Vasconcelos, M.J.: A semi-supervised genetic programming method for dealing with noisy labels and hidden overfitting. Swarm Evol. Comput. **39**, 323–338 (2018)
17. Gonçalves, I., Silva, S.: Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In: Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A.Ş, Hu, B. (eds.) EuroGP 2013. LNCS, vol. 7831, pp. 73–84. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37207-0_7
18. Vanneschi, L., Castelli, M.: Soft target and functional complexity reduction: a hybrid regularization method for genetic programming. Expert Syst. Appl. **177**, 114929 (2021)
19. Carratino, L., Cissé, M., Jenatton, R., Vert, J.P.: On mixup regularization. J. Mach. Learn. Res. **23**(1), 14632–14662 (2022)
20. Sajjadi, M., Javanmardi, M., Tasdizen, T.: Regularization with stochastic transformations and perturbations for deep semi-supervised learning. Advances in Neural Information Processing Systems 29 (2016)
21. Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y.: Cutmix: regularization strategy to train strong classifiers with localizable features. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 6023–6032 (2019)
22. Hendrycks, D., Mu, N., Cubuk, E.D., Zoph, B., Gilmer, J., Lakshminarayanan, B.: Augmix: a simple data processing method to improve robustness and uncertainty. In: International Conference on Learning Representations (2019)
23. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Trans. Evol. Comput. **23**(5), 828–841 (2019)
24. Cao, Y., Rockett, P.I.: The use of vicinal-risk minimization for training decision trees. Appl. Soft Comput. **31**, 185–195 (2015)
25. Ni, J., Rockett, P.: Training genetic programming classifiers by vicinal-risk minimization. Genet. Program Evolvable Mach. **16**, 3–25 (2015)
26. Neshatian, K., Zhang, M., Andreae, P.: A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. IEEE Trans. Evol. Comput. **16**(5), 645–661 (2012)
27. Ma, J., Gao, X., Li, Y.: Multi-generation multi-criteria feature construction using genetic programming. Swarm Evol. Comput. **78**, 101285 (2023)
28. Virgolin, M., Alderliesten, T., Bosman, P.A.: On explaining machine learning models by evolving crucial and compact features. Swarm Evol. Comput. **53**, 100640 (2020)
29. Wang, C., Chen, Q., Xue, B., Zhang, M.: Shapley value based feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. In: Australasian Conference on Data Science and Machine Learning, pp. 163–176. Springer (2023)
30. La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A probabilistic and multiobjective analysis of lexicase selection and ε-lexicase selection. Evol. Comput. **27**(3), 377–402 (2019)
31. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
32. Yao, H., Wang, Y., Zhang, L., Zou, J.Y., Finn, C.: C-mixup: improving generalization in regression. Adv. Neural. Inf. Process. Syst. **35**, 3361–3376 (2022)
33. Olson, R.S., La Cava, W., Orzechowski, P., Urbanowicz, R.J., Moore, J.H.: Pmlb: a large benchmark suite for machine learning evaluation and comparison. BioData Mining **10**, 1–13 (2017)
34. Zhang, B.T., Mühlenbein, H.: Balancing accuracy and parsimony in genetic programming. Evol. Comput. **3**(1), 17–38 (1995)

35. Chen, Q., Xue, B., Zhang, M.: Improving symbolic regression based on correlation between residuals and variables. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 922–930 (2020)
36. Chiu, W.Y., Yen, G.G., Juan, T.K.: Minimum manhattan distance approach to multiple criteria decision making in multiobjective optimization problems. IEEE Trans. Evol. Comput. **20**(6), 972–985 (2016)
37. Ni, J., Drieberg, R.H., Rockett, P.I.: The use of an analytic quotient operator in genetic programming. IEEE Trans. Evol. Comput. **17**(1), 146–152 (2012)
38. de Sá, A.G.C., Freitas, A.A., Pappa, G.L.: Automated selection and configuration of multi-label classification algorithms with grammar-based genetic programming. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.) PPSN 2018. LNCS, vol. 11102, pp. 308–320. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99259-4_25
39. Nicolau, M., Agapitos, A.: Choosing function sets with better generalisation performance for symbolic regression models. Genet. Program Evolvable Mach. **22**(1), 73–100 (2021)
40. Chu, T.H., Nguyen, Q.U., O'Neill, M.: Tournament selection based on statistical test in genetic programming. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 303–312. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_28
41. de França, F.O.: Alleviating overfitting in transformation-interaction-rational symbolic regression with multi-objective optimization. Genet. Program Evolvable Mach. **24**(2), 13 (2023)
42. Mundhenk, T., Landajuela, M., Glatt, R., Santiago, C.P., Petersen, B.K., et al.: Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. Adv. Neural. Inf. Process. Syst. **34**, 24912–24923 (2021)
43. Kamienny, P.A., d'Ascoli, S., Lample, G., Charton, F.: End-to-end symbolic regression with transformers. Adv. Neural. Inf. Process. Syst. **35**, 10269–10281 (2022)