Improving evolutionary algorithms by enhancing an approximative fitness function through prediction intervals

Christina Plump*, Bernhard J. Berger*, Rolf Drechsler*†
*Institute of Computer Science, University of Bremen, Bremen, Germany
†Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
Email: {cplump, berber, drechsle}@uni-bremen.de

***Abstract*—Evolutionary algorithms are a successful application of bio-inspired behaviour in the field of Artificial Intelligence. Transferring mechanisms such as selection, mutation, and recombination, evolutionary algorithms are capable of surmounting the disadvantages of traditional methods as—for example, hillclimbing—have. Adjusting an evolutionary algorithm to a specific problem requires both, a good understanding of the problem and deep knowledge of the effects of choosing one or another operator in the algorithm. This becomes an especially difficult task when the fitness function is not analytically given - that is, exists only as an approximation, that is highly dependent on the present training data. We propose using prediction intervals to modify the fitness function such, that worse fitness values are less penalized if they occur in a poorly fitted area. We evaluate this with an example from material sciences as well as four standard benchmark algorithms for evolutionary algorithms using a Support Vector Regression for training the approximative fitness function and find that our approach outperforms the naive approximative function.**

I. Introduction

Search, and optimisation problems are ubiquitous in everyday life, industrial application and political decision making. Finding an optimal solution to a given task has therefore been the subject of decade-long research in mathematics, computer science, engineering, economics—to name a few areas. Evolutionary algorithms are one technique that has gained widespread acceptance and attention since its development [1]. These are part of bioinspired approaches, like ant-colonisation and neural networks, as they adopt the evolutionary process seen in nature. As such, they find the optimal candidate in a search space by recombining and mutating individuals of the current population and selecting them for propagation to the next generation based on their fitness value. When the population converges, or the algorithm has produced a fixed number of generations, the algorithm stops and yields a potential optimum. Through its mutation and recombination operators, evolutionary algorithms can leave local optima—in contrast to, e.g. hillclimbing algorithms—and potentially reach the global optimum. In general, evolutionary algorithms surpass other techniques when used in unbecoming landscapes. Unbecoming refers to properties like discontinuities, noisiness, or multimodality. All of these characteristics are obstacles to usual algorithms. There are several intricate issues when applying evolutionary algorithms to real-world applications, e.g. choice of individuals' representation and design of operators. However, the most crucial is the application's fitness function. That is because evolutionary algorithms require a large number of evaluations of that fitness function. Thus, it has a relevant impact on the algorithm performance and the quality of the result. In many real-world applications, the fitness function is either unknown or computationally expensive. For example, this might be the case for experimental setups (unknown) or complex numerical differential equations (computationally expensive). The latter may be the case for many natural sciences applications, e.g. predictions of weather phenomena or predictions on earthquakes. The former, however, may be the case when no analytical relationship is yet known, and every connection has to be established through experiments. Another reason is a potential setup, where the fitness depends on subjective evaluation, as might be the case in music or art. A well-known approach to tackling this issue is employing approximative fitness functions [2], [3]. When talking about unknown fitness relations, this is the only possibility, whereas, for computationally expensive functions, using the correct evaluation is possible but has to be restricted due to performance issues [4], [5]. These are only some example of the wide variety of research that has been done in this area.

In this paper, we will focus on a situation that is quite common in experimental setups: Scientist look for the best input to produce specific output variables. For example, given a specified set of material properties, search for the best parameter configuration necessary to create a structural material compliant with these properties. However, there is no analytic relationship known between material property and parameter configuration. Therefore, experiments collect training data, which serve as input to a machine learning algorithm that predicts the relationship between dependent and independent variables. Thus, the machine learning algorithm yields the base for a fitness function targeting the abovementioned search. There is no possibility to obtain more training data or to reiterate the training process. However, statistical information as goodness-of-fit, distribution of error, prediction mean, and others are known. We propose incorporating this statistical

This work was partially founded by the German Research Foundation (DFG) in subproject P02 – Heuristic, Statistical and Analytical Experimental Design – of CRC1232 (project number 276397488)

knowledge into the fitness function to heal some of the perks inherent to approximative fitness functions. In doing so, we achieve two merits: First, we drive the fitness evaluation closer to the correct value, and second, we do not penalise poor fitness values as much when they stem from a search space area that has a low goodness-of-fit. For our approach, we chose a Support Vector Regression to train the approximative fitness function as it is the best fit for the application domain that led to our approach. Several other machine learning techniques could be employed, which we will investigate in further research.

We evaluate our proposed method with several well-known benchmark functions for evolutionary algorithms. To do so, we compare the results of an evolutionary algorithm with an approximative fitness function to an evolutionary algorithm with our adjusted approximative fitness function, hereby using the actual fitness values (stemming from the true fitness function). We analyse several influencing factors, like encoding, selection operators and distribution of training data. Furthermore, we apply our approach to an application from material sciences, similar to the one described above.

The remainder of this paper is structured as follows: Section 2 gives necessary background information about support vector regression and the method to compute prediction intervals for traceability. Section 3 introduces our approach, and Section 4 describes the implementation. In Section 5, we describe our evaluation process and results in detail. Section 6 concludes this paper and shows further research directions.

II. Background

We will shortly revisit Support Vector Regression (SVR) based on kernels and a technique to produce prediction intervals, presented by Brabanter et al. [6] because we focus on this learning technique in this paper.

A. Kernel-based Support Vector Regression

A standard prediction situation is given as follows: Let $T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$ a set of n data points, composing the training data. $\boldsymbol{x} = (x_1, ..., x_m)^\top$ is an m-dimensional vector and $y_i \in \mathbb{R}$. The goal is to learn a function $\phi : \mathscr{X} \to \mathscr{Y}$ that describes the functional relation the training data follow. We follow Smola et al. [7] in our description of kernel regression.

The easiest assumption for a function is a linear form. Then ϕ can be written as follows:

$$\phi(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \tag{1}$$

where $\boldsymbol{w} \in \mathscr{X}, b \in \mathbb{R}$. This reformulates the search for ϕ to a search for $\boldsymbol{w}, b$. In ϵ-SVR as introduced by Vapnik [8] the optimisation problem then becomes

$$\text{minimize } \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{2}$$

$$\text{following } |y_i - \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle - b| < \epsilon \tag{3}$$

This requires the existence of a function ϕ that explains the training data with an error smaller than ϵ. As this might not always be the case, one introduces slack variables that allow a training point to dismiss the ϵ-constraint. The problem then becomes:

$$\text{minimize } \frac{1}{2}\|\boldsymbol{w}\|^2 + \gamma \sum_{i=1}^{n} \zeta_i \tag{4}$$

$$\text{following } |y_i - \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle - b| < \epsilon + \zeta_i \tag{5}$$

$$\zeta_i \geq 0 \tag{6}$$

The slack variables are added to the minimising function to ensure that they are not overused.

Formulating Equation 6 as Lagrangian Problem, using lagrangian multipliers α_i, α_i^* to state the dual problem, one obtains the *support-vector expansion*:

$$\phi(\boldsymbol{x}) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \langle \boldsymbol{x}, \boldsymbol{x}_i \rangle + b \tag{7}$$

Starting from this formulation it is easy to go from linear to non-linear regressions: Equation 7 shows, that $\boldsymbol{w}$ is no longer explicitly needed, but only the scalar-product with all training points where $(\alpha_i - \alpha_i^*)$ doesn't vanish. At this point, one exploits the so-called *kernel-trick* which allows computation of a scalarproduct in the input space instead of the feature space and simply substitutes $\langle \boldsymbol{x}, \boldsymbol{x}_i \rangle$ by $k(\boldsymbol{x}, \boldsymbol{x}_i)$. Including the kernel-function in the problem formulation and writing the dual problem as matrix equation, we obtain

$$\begin{pmatrix} 0 & 1 & \cdots & 1 \\ 1 & k(\boldsymbol{x}_1, \boldsymbol{x}_1) + \frac{1}{\gamma} & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) + \frac{1}{\gamma} \end{pmatrix} \begin{pmatrix} b \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} 0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{8}$$

to be solved for b and $\boldsymbol{\alpha}$. Equation 7 then becomes

$$\hat{\phi}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) \tag{9}$$

Depending on the training data, different kernels can be used. Well-known kernels include the *gaussian kernel*, *polynomial kernel*, *laplacian kernel* or *sigmoid kernel*. The *gaussian* and *laplacian* kernel belong to the family of Radial Basis Functions and thus behave similarly.

B. Goodness-of-Fit Measures

Given an estimated function, it is crucial to know, how *good* the function describes the training data (and even better: is able to estimate function values of new input data). There are several known measures for that, called *goodness-of-fit measures*.

1) k-fold cross-validation: One measure that is often used to prohibit overfitting is a *k-fold crossvalidation*. We will use this measure later on to categorise kernel-regressions. A k-fold cross-validation divides the training data in k subsets, uses $k - 1$ for estimating the function and the kth for testing, i.e.

computes $\sum_i |y_i - \phi(\boldsymbol{x}_i)|$ for $i \in \{(k-1)\cdot n/k+1, ..., n\}$. This is repeated k times, i.e. each subset is used once for testing. Finally, an average and a standard deviation are computed [9], [10].

2) Confidence and Prediction Intervals: While k-fold cross-validation gives an overall measure of the estimation's goodness-of-fit, it fails to provide information about the confidence in the prediction for a given input. That's where confidence and prediction intervals come in. They draw a lower and upper bound around the estimated output value with a given confidence, which yields information about how precise the estimation is.

Definition 1: The confidence interval for the mean of the dependent variable, conditioned on given values for the independent variables, that is $\mathbb{E}[Y|\boldsymbol{X}=\boldsymbol{x}]$ is (confidence niveau of $1-\alpha$)

$$\hat{\phi}(\boldsymbol{x}) \pm q_{\alpha/2} \cdot SE\left[\hat{\phi}(\boldsymbol{x})\right]$$

where $q_{\alpha/2}$ is the respective quantile and SE denotes the standard error (regardless of its computation).

A confidence interval can be interpreted as follows: When drawing n times, the mean of these results will be in the given interval in $1-\alpha$ % of the cases. Prediction intervals on the other hand focus on a singular prediction and therefore have an increased standard error:

Definition 2: The predicition interval for a new observation of the dependent variable can be computed as follows:

$$\hat{\phi}(\boldsymbol{x}) \pm q_{\alpha/2} \cdot SE\left[\hat{\phi(\boldsymbol{x})} + \epsilon\right]$$

where $q_{\alpha/2}$ is the respective quantile and SE denotes the standard error (regardless of its computation).

Prediction intervals give an estimate of how precise the prediction is in terms of a single observation. Therefore, a prediction interval will always be longer than a confidence interval.

For non-linear regressions it is no trivial task to compute a prediction or confidence interval. Brabanter et al. [6] proposed a technique to compute approximate confidence and prediction intervals for SVR.

Analogously to Definition 2 they construct a pointwise $(1-\alpha)$ prediction interval as follows:

$$\hat{\phi}_c(\boldsymbol{x}) \pm z_{1-\alpha/2}\sqrt{\hat{\sigma}(\boldsymbol{x}) + Var\left[\hat{\phi}(\boldsymbol{x})|X=\boldsymbol{x}\right]} \tag{10}$$

$\hat{\phi}_c(\boldsymbol{x}) = \hat{\phi}(\boldsymbol{x}) - bias\left[\hat{\phi(\boldsymbol{x})}|X=\boldsymbol{x}\right]$ is a bias-corrected estimation of ϕ. $z_{1-\alpha/2}$ is the $(1-\alpha/2)$-quantile of the standard normal distribution.

We will shortly discuss how to compute the estimated bias, the estimated standard deviation as well as the variance. Most of this is based on the assumption that $\hat{\phi}$ can be seen as a linear smoother, i.e. $\hat{\phi}(\boldsymbol{x})$ is a linear combination of training output data y_i. Hence, a matrix L can be constructed, where each column contains the weights for the linear combination of a training point:

$$L = \left(L(\boldsymbol{x}_1) \middle| \cdots \middle| L(\boldsymbol{x}_n) \right)$$

and $L(\boldsymbol{x}_i) = (l_1(\boldsymbol{x}_i), ..., l_n(\boldsymbol{x}_i))$ s.t.

$$\hat{\phi}(\boldsymbol{x}_i) = \sum_{j=1}^{n} l_j(\boldsymbol{x}_i) y_j$$

Given the dual representation from the kernel regression (see Equation 12), L can be computed via

$$L = \left(K \left(Z^{-1} - Z^{-1}\frac{1_{n\times n}}{c} Z^{-1} \right) + \frac{1_{n\times n}}{c} Z^{-1} \right) \tag{11}$$

where $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $Z = K + \frac{1}{\gamma} I_n$ and $c = \sum_{i=1}^{n}\sum_{j=1}^{n} z_{ij}$. Similiarly, for a new observation $\boldsymbol{x}$, one can compute

$$L(x) = \left(K(\boldsymbol{x}) \left(Z^{-1} - Z^{-1}\frac{1_{n\times n}}{c} Z^{-1} \right) + \frac{1_n}{c} Z^{-1} \right) \tag{12}$$

where $K(\boldsymbol{x}) = (k(\boldsymbol{x}, \boldsymbol{x}_1), ..., k(\boldsymbol{x}, \boldsymbol{x}_n))$.

With this linear smoother at hand, we have

$$bias\left[\hat{\phi(\boldsymbol{x})}|X=\boldsymbol{x}\right] = L(\boldsymbol{x})^\top (\hat{\phi}(\boldsymbol{x_1}), ..., \hat{\phi}(\boldsymbol{x_n})) - \hat{\phi}(\boldsymbol{x}) \tag{13}$$

To compute the estimated standard deviation, we make use of L and $L(\boldsymbol{x})$ again. Furthermore, we need the estimation residuals $\epsilon_i = y_i - \hat{\phi}(\boldsymbol{x}_i)$. Then, with $diag(A)$ containing the diagonal entries of a matrix A and $S(\boldsymbol{x})$ a linear smoother for the standard deviation based on the residuals, we have

$$\sigma(\boldsymbol{x}) = \frac{S(\boldsymbol{x})^\top (\epsilon_1^2, ..., \epsilon_n^2)}{1 + S(\boldsymbol{x})^\top \cdot diag(LL^\top - L^\top - L)} \tag{14}$$

to estimate the standard deviation.

$Var\left[\hat{\phi}(\boldsymbol{x})|X=\boldsymbol{x}\right]$ is the only unknown left from Equation 10. We obtain this again with the linear smoother $L(\boldsymbol{x})$:

$$Var\left[\hat{\phi}(\boldsymbol{x})|X=\boldsymbol{x}\right] = L(\boldsymbol{x})^\top \begin{pmatrix} \sigma^2(\boldsymbol{x}_1) & & \\ & \ddots & \\ & & \sigma^2(\boldsymbol{x}_n) \end{pmatrix} L(\boldsymbol{x}) \tag{15}$$

We use this computation in our methodology presented in Section III.

III. Methodology

In a perfect evolutionary world, encoding is simple, and fitness functions are precise, analytically given and easy to compute. Unfortunately, we have to deal with fitness functions that are either entirely unknown, highly noised or not efficiently computable. In this case, one uses surrogate functions to allow the usage of an evolutionary algorithm. However, surrogate functions differ from the true fitness function and may lead the algorithm in the wrong direction when assigning incorrect fitness values to individuals. When the surrogate function depends on training data that does not follow a uniform distribution, the function's preciseness in a given region depends on the training data's density (and clearness) for that region.

We consider the following situation: There is an unknown functional relation φ between a multidimensional input variable $\boldsymbol{X}$ and an output variable Y. The goal of the evolutionary algorithm is to find a realisation of $\boldsymbol{X}$ that corresponds to a pre-determined realisation of Y. This pre-determined realisation is denoted by y^x. Through experiments (or simulations) training data $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_n, Y_n)$ have been obtained and therefrom a prediction technique calculated an estimate $\tilde{\varphi}$ of the relation φ. The prediction technique at this stage is not subject to restrictions (e.g. Support Vector Regression, Gaussian Processes, Neural Networks and others are possible). This gives rise to the natural fitness function

$$f(\boldsymbol{X}) = |\tilde{\varphi}(\boldsymbol{X}) - y^*| \tag{16}$$

where y^* is the given realisation of Y and the objective is to minimise the fitness value (i.e. realising $\tilde{\varphi}(\boldsymbol{X}) = y^*$). Comparing this to the natural fitness function, when φ is known, we obtain

$$\left|\tilde{f}(\boldsymbol{X}) - f(\boldsymbol{X})\right| = |\tilde{\varphi}(\boldsymbol{X}) - y^*| - |\varphi(\boldsymbol{X}) - y^*| \tag{17}$$

$$= \begin{cases} |\tilde{\varphi}(\boldsymbol{X}) - \varphi(\boldsymbol{X})| & \text{if } a \cdot b > 0 \\ |\tilde{\varphi}(\boldsymbol{X}) + \varphi(\boldsymbol{X}) - 2y^*| & \text{if } a \cdot b \leq 0 \end{cases} \tag{18}$$

where $a = \tilde{\varphi}(\boldsymbol{X}) - y^*$ and $b = \varphi(\boldsymbol{X}) - y^*$.

The approximative fitness function $\tilde{f}$ makes an estimation error - the question is whether and how this affects the evolutionary algorithm. We can summarize the possible outcomes in a 2×2 matrix (see Table I):

TABLE I
ESTIMATION EFFECTS ON EVOLUTIONARY ALGORITHM

	good candidate	bad candidate
overestimating ($\tilde{f} - f > 0$)	problematic	unproblematic
underestimating ($\tilde{f} - f < 0$)	unproblematic	problematic

We use the term *good* and *bad* referring to their actual fitness and influence on further population development. For example, a *good* individual would be one, not only with a good fitness value (closer to zero) but also leading to a more promising search space area. Overestimating the fitness of a good individual is problematic, as this decreases the likelihood of keeping its genetic information in the population. Underestimating, on the other hand, is unproblematic, as it only strengthens a positive effect. Overestimating a bad individual is unproblematic for the same reason: It only strengthens an intended effect. However, underestimating a bad individual's fitness is problematic, as it might lead to a prolonged age in the population or even a genetic influence. Generally speaking, in two out of four cases, the use of an approximative fitness function (surrogate function) does hinder the evolutionary algorithm, and in two, it does not. However, it is unlikely to find a solution that fixes both problematic situations and leaves unproblematic ones intact. This problem is similar to hypothesis testing: The restriction of both errors is impossible. Therefore, one has to decide on one. As more fit individuals usually have a more decisive influence on a population (are more likely to be chosen for recombination, are more likely to have a higher age) than *bad* individuals, we will prioritise them. The idea is that the relative influence of a *good* individual on the further evolution of the population is higher than the relative influence of *bad* individual. Hence, we will focus on lessening the negative impact of the overestimation of a *good* candidate and consider that any changes might negatively affect the underestimation of a *bad* candidate. However, we will make sure that the positive effects of approximative fitness functions for *good* candidates do not vanish.

Returning to the error analysis, we notice the following: In general, approximative functions do not have constant prediction errors. Instead, these errors depend on the approximated value. Hence, we can write:

$$\tilde{\varphi}(\boldsymbol{X}) = \varphi(\boldsymbol{X}) + \epsilon(\boldsymbol{X}) \tag{19}$$

Using this in Equation 18, we obtain

$$\left|\tilde{f}(\boldsymbol{X}) - f(\boldsymbol{X})\right| = \begin{cases} |\tilde{\varphi}(\boldsymbol{X}) - \varphi(\boldsymbol{X})| & \text{if } a \cdot b > 0 \\ |\tilde{\varphi}(\boldsymbol{X}) + \varphi(\boldsymbol{X}) - 2y^*| & \text{if } a \cdot b \leq 0 \end{cases} \tag{20}$$

$$= \begin{cases} |\epsilon(\boldsymbol{X})| & \text{if } a \cdot b > 0 \\ |2\tilde{\varphi}(\boldsymbol{X}) - 2y^* - \epsilon(\boldsymbol{X})| & \text{if } a \cdot b \leq 0 \end{cases} \tag{21}$$

$$= \begin{cases} |\epsilon(\boldsymbol{X})| & \text{if } a \cdot b > 0 \\ \left|2\tilde{f} - \epsilon(\boldsymbol{X})\right| & \text{if } a \cdot b \leq 0 \end{cases} \tag{22}$$

Generally, the higher $|\epsilon(X)|$, the higher the effect described in Table I. In our approach, we adapt the fitness function such that high fitness values get smaller when caused by an individual $\boldsymbol{X}$ with a high $\tilde{f}(\boldsymbol{X})$. Roughly: When the approximation is poor, high distances $\tilde{f}(\boldsymbol{X})$ are less penalized. When the approximation is good, fitness values should only change marginally. Please keep in mind that this will not solve both problems simultaneously, hence our decision to focus on a better assessment for *good* candidates. As we cannot compute $\epsilon(\boldsymbol{X})$—because if we could, we would use φ instead—we need other means of quantifying the goodness of the approximation in a particular point. At this point, prediction intervals come to mind. The length of a prediction interval reflects the goodness of the prediction at that point. The shorter the interval, the more precise the approximation.

Factoring in the confidence level of the prediction interval, we can bound the actual fitness value $\varphi(\boldsymbol{X})$:

$$\tilde{\varphi}(\boldsymbol{X}) - y^* - \frac{L}{2} \leq \varphi(\boldsymbol{X}) - y^* \leq \tilde{\varphi}(\boldsymbol{X}) - y^* + \frac{L}{2} \tag{23}$$

with probability $1 - \alpha$, where α is the confidence level for constructing a prediction interval for $\boldsymbol{X}$ with length $\frac{L}{2}$. With this in mind and the analysis from Equation 22, we propose the following adjusted fitness function:

$$\tilde{f}_{adj}(\boldsymbol{X}) = \frac{|\tilde{\varphi}(\boldsymbol{X}) - y^*|}{\frac{L(\boldsymbol{X})}{2}} \cdot |\tilde{\varphi}(\boldsymbol{X}) - y^*| \tag{24}$$

We can interpret Equation 24 as follows: If the approximate distance is smaller than half the length of the prediction interval in that point, we decrease the approximate distance (the factor is smaller than 1). If it is worse than that, we increase it (the factor is greater than 1). This logic applies to our objective mentioned above: If we have a higher approximative distance, but in a poorly fitted area, i.e. a larger prediction interval, we decrease the distance such that the individual's evaluation will not be too bad. Additionally, when we have a higher approximative distance in a well-fitted area, we increase the fitness such that medium and large distances in well-fitted areas face a penalisation.

IV. Implementation

We created a Java-based implementation of our proposed approach for the evaluation. The implementation employs Jenetics and SMILE and is written in Java. Jenetics is a highly extensible evolutionary algorithm framework [11] which we extended for our work. We implemented different encodings and fitness functions. One of the fitness functions takes the prediction intervals into account. As we focus on the prediction method of Support Vector Regression for this paper, we calculate the prediction intervals based on the algorithm by Brabanter et al. [6]. SMILE, a Java-based statistic library supporting different regression algorithms [12], helps us calculate the regressions and the corresponding prediction intervals.

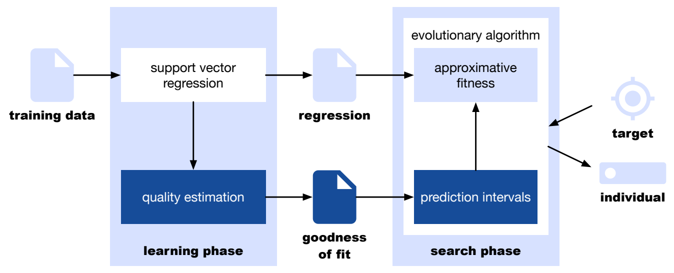


Fig. 1. Details on the Implementation

Figure 1 depicts a sketch of our approach's implementation. All steps or data shown in light blue or white symbolise the original evolutionary algorithm's steps using an approximative fitness function. Dark blue elements represent extensions made to the fitness function to take the prediction intervals into account. The learning phase, which takes place before the actual evolutionary algorithm, takes the training data and trains a least square support vector regression, which we store for later use. In this step, we additionally do a quality estimation of the determined regression. This calculation results in extra goodness of fit values which we store for later use, as well. The search phase comprises the evolutionary algorithm. The approximative fitness function takes a target specification as its optimisation goal and finds an individual whose predicted value is close to the target. Therefore, the approximative fitness function uses the calculated regression to determine the fitness of individuals. We added a factor using the prediction interval length, which the algorithm calculates for each individual. The calculation uses the goodness-of-fit values stored earlier.

To reduce the calculation time during the evolutionary algorithm, we compute as much information necessary for the prediction intervals as possible beforehand.

Algorithm 1 Computation during Learning Phase

```
1: Compute $\hat{y}_i = \hat{\phi}(\boldsymbol{x}_i)\ \forall i \in \{1,..,n\}$
2: Compute $\epsilon_i = \hat{y}_i - y_i\ \forall i \in \{1,..,n\}$
3: Compute $L$ following Equation 11 and $diag(LL^\top - L^\top - L)$
4: Store $IM = Z^{-1} - Z^{-1}\frac{1_{n\times n}}{c}Z^{-1}$ and $IV = \frac{1_{n\times n}}{c}Z^{-1}$
5: Compute $\hat{\sigma}(\boldsymbol{x}_i)\ \forall i \in \{1,..,n\}$
```

This leaves only the following for online computation:

Algorithm 2 Computation during Search Phase

```
Require: $\boldsymbol{x}, \alpha$
1: Compute $K(\boldsymbol{x})$ following Equation 12
2: $L(\boldsymbol{x}) = K(\boldsymbol{x}) * IM + IV$
3: Compute $\sigma(\boldsymbol{x})$ following Equation 14
4: Compute $l/2 = z_{1-\alpha/2}\sqrt{\sigma_{\boldsymbol{x}} + \sum_{i=1}^{n} l_i^2(\boldsymbol{x}) \cdot \sigma^2(\boldsymbol{x})}$
```

The precomputation of IM and IV (see Algorithm 1, line 4) allows to have the most costly step (matrix inversion) performed only once. The online computation then has only n kernel computations and three matrix multiplications to perform, making it more efficient.

V. Evaluation

This section is structured as follows: We first present the research questions to be investigated. We evaluate our approach on four benchmark functions and three test cases from a real-world scenario. For the analytical (benchmark) as well as the application domain, we first describe the evaluation setup, second show and describe the obtained results and third discuss our results.

We aim to answer the central question:

RQ 1: Does the proposed adjusted fitness function perform better as the normal approximative fitness function?

Because we identify the chosen target, the prediction's goodness-of-fit, the training data's distribution and the evolutionary algorithm's configuration as influencing factors, we additionally plan to investigate their influence on the performance of our proposed adjustment of approximative fitness functions. This results in the following research questions:

RQ 2: Does the encoding have a substantial influence on the performance of the adjusted fitness function?

RQ 3: Does the distribution of training data have a substantial influence on the performance of the adjusted fitness function?

TABLE II
EVALUATED BENCHMARK FUNCTIONS WITH PARAMETERS, RANGE ($i \in \{1, ..., m\}$), AND DIMENSION

Function	Formula	Range	Dimension
WeightedSphere	$f(X) = \sum_{i=1}^{m} i^2 X_i^2$	$X_i \in [-5.12, 5.12]$	$m = 10$
Rastrigin	$f(X) = 10m + \sum_{i=1}^{m} \left(X_i^2 - 10\cos 2\pi X_i\right)$	$X_i \in [-5.12, 5.12]$	$m = 10$
Rosenbrock	$f(X) = \sum_{i=1}^{m-1} \left(100\left(X_i^2 - X_{i+1}\right)^2 + (1 - X_i)^2\right)$	$X_i \in [-2.048, 2.048]$	$m = 10$
Ackley	$f(X) = 20 + \exp(1) - 20\exp(-\sqrt{\frac{1}{5m}\sum_{i=1}^{m} X_i^2}) - \exp(\frac{1}{m}\sum_{i=1}^{m}\cos 2\pi X_i)$	$X_i \in [-20, 30]$	$m = 10$

RQ 4: Does the estimation quality of the regression influence the performance of the adjusted fitness function?

We first evaluate our approach on a set of four benchmark functions, and second on a real-world application from material sciences.

A. Benchmark-Functions

1) Description of Setup: The benchmark functions differ from the real-world scenario in one relevant aspect: We know the true function. To simulate an analogue situation as in the real-world scenario, we use a three-level approach: First, we construct training data: According to a specified distribution, we generate input values and calculate corresponding output values - according to the given function. Second, we noise these output data using Gaussian noise: $\delta_i \sim \mathcal{N}(0, 1)$), as experimental data is seldomly perfect. Third, we use different SVR to obtain the approximative functions. These approximative functions then serve as input for the different evolutionary algorithms and different targets. As many factors influence the performance of our approach, we used a combinatorial approach using four categories. The first category contains the different functions. The second category handles the distribution of training data. The third category considers different goodness-of-fit of the respective SVR and the fourth consists of various evolutionary algorithms (see Table III).

One setup specification (consisting of function, distribution, goodness-of-fit, evolutionary algorithm and target) is optimised ten times for each fitness function. We then compare the mean of the true (benchmark) fitness over these ten runs for the ones that followed an optimisation with the adjusted fitness function to the ones that followed an optimisation with the approximative fitness function, i.e. $\mu(f(bestInd_{adj}))$ and $\mu(f(bestInd_{app}))$. Please note the index at *bestInd* describes the fitness function used for the evolutionary algorithm, this *bestInd*ividual resulted from, while an index on f would denote the fitness function used to evaluate the fitness of this given individual. Additionally, we compare the corresponding best results of these ten runs. Thus, we use the true fitness function to obtain an objective comparison but use the modified one for the evolutionary process.

For each setup, we perform Welsh's t-test [13] with Satterwaite's degree of freedom):

TABLE III
ENCODING, SELECTORS AND ALTERERS FOR EVOLUTIONARY ALGORITHM (EA)

Encoding	Selectors	Mutator	Recombinator
real-valued	Elite/Tournament, StochasticUniversal	Gaussian	Mean, Arithmetic-Crossover
naive bit	Elite/Tournament, StochasticUniversal	Swap	SinglePointCrossover, UniversalCrossover
gray bit	Elite/Tournament, StochasticUniversal	Swap	SinglePointCrossover, UniversalCrossover

$$t = \frac{\mu(f(bestInd_{adj})) - \mu(f(bestInd_{app}))}{\sqrt{\frac{\sigma^2(f(bestInd_{adj}))}{n(runs)} + \frac{\sigma^2(f(bestInd_{app}))}{n(runs)}}} \tag{25}$$

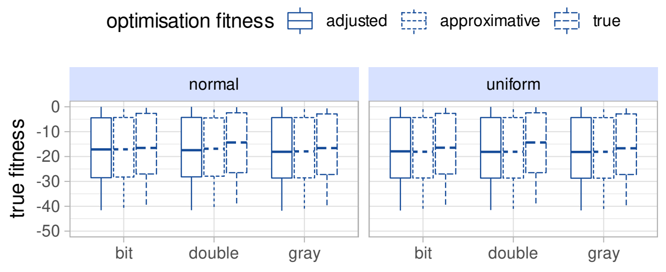
$$\nu = \frac{\left(\frac{\sigma^2(f(bestInd_{adj}))}{n(runs)} + \frac{\sigma^2(f(bestInd_{app}))}{n(runs)}\right)^2}{\frac{\left(\frac{\sigma^2(f(bestInd_{adj}))}{n(runs)}\right)^2}{n(runs)-1} + \frac{\left(\frac{\sigma^2(f(bestInd_{app}))}{n(runs)}\right)^2}{n(runs)-1}} \tag{26}$$

ν denotes the degrees of freedom we will use for the hypothesis test. $n(runs) = 10$ for our setup. These distances and test statistics, and corresponding p-values are the foundation of all later analysis.

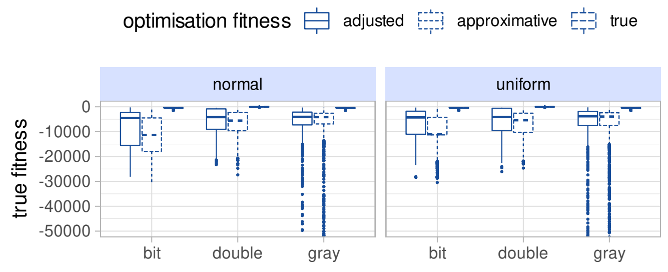
a) Functions: A classification of benchmark functions for EA follows their main properties: modality and separability. We choose four functions, each for one combination of these properties. We chose the WeightedSphere Function for unimodality and separability, and for multimodality and separability, we chose the Rastrigin Function. We chose the Rosenbrock Function for unimodality and inseparability and the Ackley Function for multimodality and inseparability. Table II lists all functions formula expression, chosen parameters (if necessary) and ranges for input variables.

b) Distributions: As the distribution of training data affects the local goodness-of-fit of the regression made on these training data, we used two different distributions for our evaluation. First, we used an independent multivariate normal distribution with a mean at the centre of the proposed range for the given function. For example, when using the Ackley-Function with range: $-20 \leq X_i \leq 30$, we set $\mu(X_i) = 5.0$. We set the standard deviation such that the ranges are in a 3σ interval of the distribution.

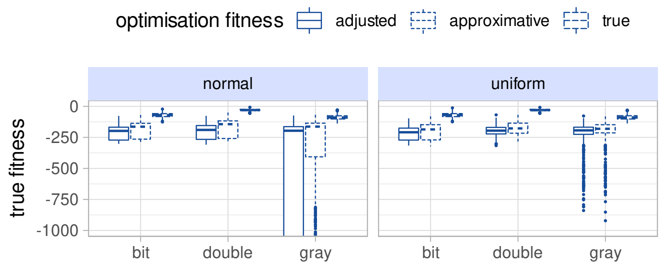
Second, we used an independent multivariate uniform distribution to mirror an experimental design without any expert knowledge beforehand. We choose lower and upper bound according to the ranges of the respective function.



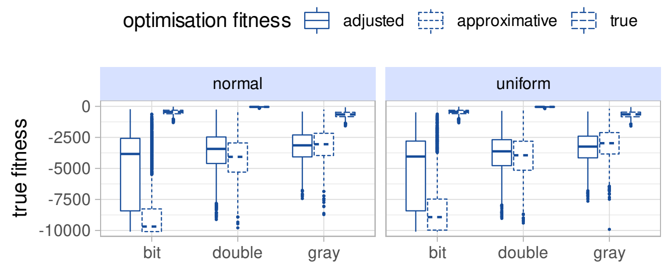
(a) Results of Ackley-function with m=10, n=1000



(b) Results of Rosenbrock-function with m=10, n=1000



(c) Results of Rastrigin-function with m=10, n=1000



(d) Results of WeightedSphere-function with m=10, n=1000

Fig. 2. Evaluation results for four benchmark functions comparing three optimisation modi through the true fitness value (m is number of dimensions, n is number of training points

c) Goodness-of-fit of SVR: We expect different results of our approach depending on the goodness-of-fit of the approximative function. To compare like with like, we compute three approximative functions for one setting (function, distribution) and divide them into three groups: bad, average and good. We base this classification on the results of a 10-fold cross-validation. As we use different benchmark functions, we obtain different areas of goodness-of-fit values. For example, the WeightedSphere function has very small cross-validation results, whereas the Rosenbrock function has much higher cross-validation results. We address this with the use of a *relative* classification: We, therefore, classify the best obtained result as *good*, classify the worst obtained result, which is still in a range of 100% of the best obtained result, as *poor* and chose a result around the average of both values as *average*. We use different kernels with different settings to obtain approximations in these different classes. The kernels we use are: Gaussian kernel, laplacian kernel and polynomial kernel [14]. The meta-parameters ϵ, soft margin and tolerance are kept identical to ensure comparability.

d) Evolutionary Algorithm: We use several different evolutionary algorithms to obtain a more valid result. We encode the problem domain with one chromosome for each dimension, i.e. a genotype consists of m chromosomes. A chromosome can hold the value information in three different encodings: One, as a real-valued encoding (thus, only one gene per chromosome), second as a naive bit-encoding (thus, 64 genes per chromosome), third as a gray-bit-encoding (again, 64 genes per chromosome). We use a gaussian mutator and either the mean recombinator or an arithmetic crossover as recombination operator for real-valued encodings. For bit-valued encodings, we use a swap mutator together with a single-point crossover or a uniform crossover. We either use an elitism approach, where 40% of the population are chosen with an elite selector, and the rest is chosen by a tournament-selector, or a stochastical approach, where only a small portion is chosen with an elite-selector, and the rest are chosen with a stochastic-universal-selector. The combination of all variations leads to a variety of configurations that cover many possible characteristics of EAs. For an overview, see Table III. Please note that we evaluated each of these algorithms with each of the fitness functions. We used a population size of 100 and a fixed number of 100 generations.

All in all, we ran 144 setups, each of them for 50 targets. We ran each setup once for each modus to compare the three optimisation modi true, approximative and adjusted. These 21.600 runs were repeated ten times for statistical reasons, making a total of 216.000 runs.

2) Results: First, to give the reader an overview, Figure 2 shows the distribution of $f(bestInd)$ for all employed fitness functions. The boxplots from left to right are always $f(bestInd_{adj}), f(bestInd_{app}), f(bestInd_{true})$. We can see that the adjusted fitness function outperforms the approximative function for the Rosenbrock function and the WeightedSphere function as the median and both quartiles are closer to zero. In contrast, for the Ackley Function and the Rastrigin function, the approximative function is slightly better or does not differ much from the adjusted function. Please

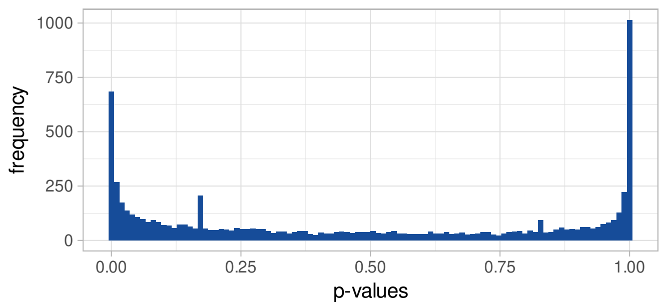


Fig. 3. Distribution of p-values for all 7200 specified runs.

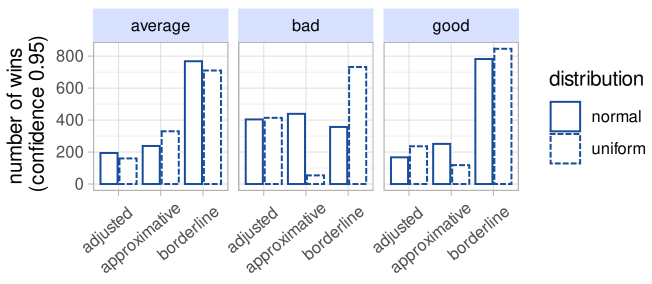


Fig. 4. Statistically significant wins split between distribution and level of goodness-of-fit

TABLE IV
RATIO OF STATISTICALLY SIGNIFICANT ADJUSTED WINS TO APPROXIMATIVE WINS

Significance	Ratio(adj:app)	result
$\alpha = 0.05$	1575:1430	1.1014
$\alpha = 0.01$	1141:832	1.3714
$\alpha = 0.005$	1012:683	1.4817
$\alpha = 0.001$	794: 383	2.0731

note the different y-axes for each function and the fact, that we employed the dual problem to Section III, i.e. calculation the negative fitness value and optimising towards zero.

Figure 3 shows the result of pairwise two-sample hypothesis-test following Equation 25. We computed the p-value for each pairing (a pairing is one distinct setup with approximatve and adjusted fitness functions) and plotted the p-values frequency. For all pairings with a p-value higher than $1 - \alpha$ the proposed method outperforms the approximative fitness function. For all pairing with a p-value smaller than α, the approximative fitness function outperforms the proposed methodology. Please note, that we performed 7200 different t-tests (number of target times number of setups) whose p-values are displayed in Figure 3. When setting $\alpha = 0.05$ to achieve statistical significance, we obtain a 1575:1430 ratio, i.e. 1575 setups were *won* by the adjusted fitness function and 1430 setupt were *won* by the approximative fitness function. *Won* means the hypothesis test decided statistically significant for this option. For $\alpha = 0.01$ we even obtain 1141:832. The results for $\alpha = 0.005, \alpha = 0.001$ can be found in Table IV.

We then used the p-values computed above to investigate the influence of the training data's distribution on the performance of the adjusted fitness function. Figure 4 shows the number of wins in each category (if neither *approximativ* nor *adjusted* performed significantly better at $1 - \alpha = 0.95$, we assigned to the category borderline). The solid bars show those for normal distribution, the dashed lines for uniform distribution. Each subgraph represents one level of goodness-of-fit. We can see, that in setups with a uniform distribution, *adjusted* outperforms *approximative* for levels bad and good, while in setups with a normal distribution, this is not the case. On the other hand, for the average case, the normal distribution performs less poor than the uniform distribution (*uniform* has a higher tendency towards *approximative* than *normal*).

We repeated this procedure to compare the performance based on levels of goodness-of-fit. Figure 5 depicts the outcome. For poorly fitted fitness functions *adjusted* outperforms *approximative*, for well fitted, there is a tendency towards *adjusted*, while for average *approximative* has higher results.

Finally, we investigated the encoding's influence and thus the algorithm's configuration's influence. Figure 6 depicts the results. We see an interesting result that fits the data depicted in Figure 2: The bit-encoding significantly favours the *adjusted* optimisation, while the gray-encoding favours the *approximative*. Interestingly, the choice of selector seems to have little to no influence.

3) Discussion: Now, we can answer our research questions from the beginning of the chapter: The evaluation on benchmark functions showed that our method, in general, outperforms the standard *approximative* technique. While there are setups where our method fails to produce better results (most seen with the Rastrigin Function), our method *wins* more setups. Interestingly, when it outperforms, it does so overwhelmingly (see Table IV).

Second, the encoding does have an influence. Setups with bit-encoding favour the *adjusted* fitness function. This observation might stem from the fact that small changes in bit-encoding can lead to huge changes in the domain. This aligns with the main idea behind the proposed method: To allow more exploration, even when the fitness values are poor (if it is due to a poor fitting). We plan to investigate this relation in future work. Third, the distribution of training data does seem to have an influence, though no significant one. For well and insufficiently trained data, setups with uniform distribution tend towards *adjusted*, while the normal distribution performs better with average fittings. This is surprising, as we expected the normal distribution to have a better influence on our method than the uniform. It might correlate with the distribution of the targets in the output space. We assume that for targets on the outskirts of our distribution, our method performs better. We continue to research in this direction. Fourth, the estimation quality is an influencing factor. Our method has the highest positive impact for good and bad levels, as we expected. Another detail the data show is that our method seemingly works better on unimodal functions (WeightedSphere and

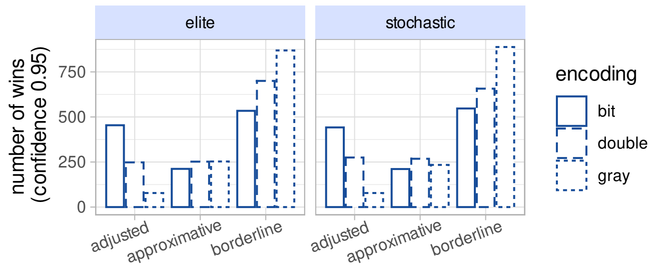


Fig. 6. Statistically significant wins for different encodings and selectors

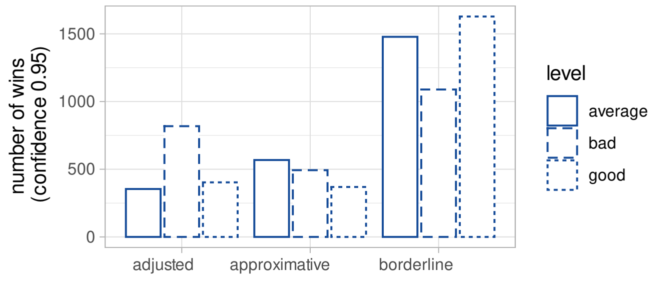


Fig. 5. Statistically significant wins for different levels of goodness-of-fit

Rosenbrock Function). Whether this is due to the modality or due to their goodness-of-fit (WeightedSphere had an overall high goodness-of-fit, Rosenbrock a relatively weak one) is still an open question which we plan to investigate further.

B. Application from material sciences

We additionally tested our approach on an application from material sciences. Ellendt et al. proposed a high-throughput method to identify the ingredients (alloy specification, thermal and mechanical treatment) for structural materials (for example, steel) that match a given requirement profile [15]. As there is no functional relation known between alloy specification/treatments and the resulting requirement profile, a predictive function was developed, mapping so-called micro-descriptors (characteristic values on micro samples of the same material) onto the requirement profile. This enables a backwards search for a given requirement profile, yielding a set of micro descriptors.

1) Description of Setup: We evaluated three different cases, all mapping micro-descriptors to requirement profiles. We first trained a predictive function (the *approximative* function) and employed our technique of prediction intervals. As there was no gold standard (for no true function is known), we had to change our evaluation such, that we compare both the *approximative* fitness values for both optimisation modi and the *adjusted* instead of the true fitness values, i.e. we used $f_{app,adj}(bestInd_{adj})$ and $f_{app,adj}(bestInd_{app})$.

2) Results: We compared the ratios $ratio(f_{app}) = \frac{wins(Adjusted)}{wins(Approximative)}$ and $ratio(f_{adj}) = \frac{wins(Approximative)}{wins(Adjusted)}$ to see, whether one of both performs comparatively better. We obtain $ratio(f_{app}) = 12/161 = 0.745$ for the approximative optimisation modus, and $ratio(f_{adj}) = 8/148 = 0.541$ for the adjusted optimization. That is, when comparing approximative fitness values, the adjusted optimisation mode performed relatively better than the approximative did, when comparing adjusted values.

3) Discussion: We see that our method produces positive results for an application example from material sciences. As the next step of evaluation, we plan to verify our results experimentally.

VI. Conclusion and Further Research

We proposed incorporating statistical knowledge about the goodness-of-fit into an approximative fitness function, given that the actual fitness function is unavailable. We implemented our approach for kernel-based SVR with an approximative technique to compute prediction intervals. We evaluated our approach using 144 different setups, customising benchmark function, training data distribution, goodness-of-fit and configuration of the evolutionary algorithm. Our results show that our approach is significantly better in more cases than the approximative fitness function. We see a better performance in unimodal benchmark functions. We plan to develop analogue adjustments for different learning techniques, such as gaussian processes and neural networks, in further work. Finally, we hope to investigate the influence of the target's position on the performance of our method.

References

[1] K. De Jong, D. Fogel, and H.-P. Schwefel, *A history of evolutionary computation*, 01 1997, pp. A2.3:1–12.
[2] D.-P. Yu and Y.-H. Kim, “Predictability on performance of surrogate-assisted evolutionary algorithm according to problem dimension,” 07 2019, pp. 91–92.
[3] T. Chugh, C. Sun, H. Wang, and Y. Jin, *Surrogate-Assisted Evolutionary Optimization of Large Problems*, 01 2020, pp. 165–187.
[4] L. Shi and K. Rasheed, *A Survey of Fitness Approximation Methods Applied in Evolutionary Algorithms*, 01 2010, vol. 2, pp. 3–28.
[5] Y. Jin, “A comprehensive survey of fitness approximation in evolutionary computation,” *Soft Computing*, vol. 9, pp. 3–12, 10 2005.
[6] K. Brabanter, J. De Brabanter, J. Suykens, and B. De Moor, “Approximate confidence and prediction intervals for least squares support vector regression,” *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 22, pp. 110–20, 11 2010.
[7] A. Smola and B. Schölkopf, “A tutorial on support vector regression,” *Statistics and Computing*, vol. 14, pp. 199–222, 08 2004.
[8] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 2000.
[9] T. Fushiki, “Estimation of prediction error by using k-fold cross-validation,” *Statistics and Computing*, vol. 21, pp. 137–146, 2011.
[10] J. D. Rodriguez, A. Perez, and J. A. Lozano, “Sensitivity analysis of k-fold cross validation in prediction error estimation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 32, no. 3, pp. 569–575, 2010.
[11] F. Wilhelmstötter, “Jenetics,” https://jenetics.io, 2021.
[12] H. Li, “Smile,” https://haifengl.github.io, 2021.
[13] B. L. Welch, “The generalization of ‘student’s’ problem when several different population variances are involved,” *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947. [Online]. Available: http://www.jstor.org/stable/2332510
[14] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and Z. Ghahramani, “Structure discovery in nonparametric regression through compositional kernel search,” *Proceedings of the 30th International Conference on Machine Learning*, 02 2013.
[15] N. Ellendt and L. Mädler, “High-throughput exploration of evolutionary structural materials,” *HTM Journal of Heat Treatment and Materials*, vol. 73, pp. 3–12, 2018.