

École polytechnique de Louvain

Augmented data in Machine Learning : virtual sample generation

Author: **Thomas RENIERS**

Supervisors: **John LEE, Michel VERLEYSEN**

Reader: **Antoine VANDERSCHUEREN**

Academic year 2020–2021

Master [120] in Computer Science and Engineering

Abstract

There is a growing interest from the private and public sectors in machine learning. These sectors will often face the problem of small data: they don't have enough data to properly feed and use most of the machine learning models. However, some workarounds exist such as transfer learning, which will use part of a model well trained to adapt it, or virtual sample generation, that will populate the small data set by generating new ones. These new samples are generated to correspond to the knowledge extracted from the small original data set, as if they were gathered by the same means. These generated samples, combined with the original ones, will increase the accuracy of the machine learning algorithms, allowing them to work in the same way as if the data set was large enough. Two different techniques, particle swarm optimisation virtual sample generation (PSOVSG) and differential evolution virtual sample generation (DEVSG), will be investigated, compared with each other as well as a baseline, to assess whether the model has improved thanks to these generated samples.

Acknowledgements

I would like to thank my two supervisors John Lee and Michel Verleysen for their continuous feedback and precious help throughout the realisation of this master thesis.

My gratitude goes to my family as well as to Manon David for the continuous love and support.

Special thanks to my parents and Alexandre Moueddene and Maxime Clément for their reviews and comments that helped me improve this work.

Finally I would like to thank all of my friends for their presence all along.

Contents

List of abbreviations	5
1 Introduction	6
1.1 The problem of small data	7
1.2 A solution to work with small data : data generation	7
1.3 Structure of this work	7
2 Dealing with small data : state of the art	9
2.1 Transfer Learning and VSG	9
2.2 Focusing on VSG	10
3 Domain range expansion	12
3.1 Multi-distribution mega-trend-diffusion	12
3.2 Triangular membership information-expanded	13
3.3 Comparison of TMIE and MD-MTD	15
4 Virtual sample generation	17
4.1 Particle swarm optimisation generation	17
4.2 VSG based on differential evolution	20
4.3 Differences and similarities	23
4.4 A baseline	23
5 Regression models used	25
5.1 Random forest as the predictive model	25
5.2 K-nearest neighbors based regression	27
6 Results	28
6.1 Data set used	29
6.2 The impact of VSG	30
6.3 Normalization	35

7	Conclusion	37
7.1	Summary of this work	37
7.2	Small change in the fitness	38
8	Appendix	39
8.1	The data used	39
8.1.1	Simulated data	39
8.1.2	Real data	39
8.2	Complementary figures	41
8.2.1	Boxplots of the results	41
8.2.2	KNN model	42
8.2.3	Plots of the output function	45
8.2.4	Boxplots of normalized outputs	46
	Annexes	48
	Bibliography	53

List of abbreviations

AI		Artificial Intelligence
ANN		Artificial Neural Network
CNN		Convolutional Neural Network
DE		Differential Evolution
DEVSG		Differential Evolution Virtual Sample Generation
DT		Decision Tree
GA		Genetic Algorithm
KNN		K-Nearest Neighbors
LI		Latent Information
MAE		Mean Absolute Error
MAPE		Mean Absolute Percentage Error
MD-MTD		Multi Distribution-Mega Trend Diffusion
ML		Machine Learning
MTD		Mega Trend Diffusion
PSO		Particle Swarm Optimisation
PSOVSG		Particle Swarm Optimisation Virtual Sample Generation
RF		Random Forest
RMSE		Root Mean Squared Error
TL		Transfer Learningg
TMIE		Triangular Membership Information Expanded
VSG		Virtual Sample Generation

Chapter 1

Introduction

In every application of machine learning, the data is really important, it is the core of the application, on which the entirety of the algorithm is dependant. In medicine as well as in industry, the data is often small, which causes the algorithms to not be as effective as they could be with larger data sets [1], [2], [3].

Machine learning consists in studying algorithms that automatically improve over time through experience [4], and is part of Artificial Intelligence (AI). In this context, AI is the study of intelligent agents, programs that attempt at making the best decision given a context, in order to achieve a certain goal. More precisely, machine learning (in short, ML) algorithms try to build a model that, trained on some sample data, will aim at approximating the target output function of which the sample data is a small instance. Such a function is often a prediction of a value or an ensemble of decisions to make. The key aspect is that these algorithms are not explicitly programmed to compute the function, instead, they try to fit the best approximation possible. This allows for more complex tasks to be solved, where a human would fail to manually create an algorithm. It also plays a role in the lack of interpretability we sometimes face with ML models; and which is one of their drawbacks [5].

We can categorize the task of ML algorithms into supervised or unsupervised learning. There exist other categories such as semi-supervised but those are left at the curiosity of the reader. Supervised learning means that the model will be trained with both the inputs, called features, and the corresponding outputs, while unsupervised only has access to the set of inputs, and try to identify some structure in the data, typically to cluster them based on some similarity metric.

In this thesis, we use supervised machine learning to make predictions. There are two types of models, depending on the target output: classification and regression. In classification, the output is discrete and called a class, it represents a set, and the model will predict at which set the current data is supposed to belong. Regression models are models that will predict a continuous output value, typically the return

value of a certain unknown function. The latter is the type of model we worked with in this thesis.

1.1 The problem of small data

With the growing interest in machine learning, many companies, as well as public services, now aim at exploiting the benefits machine learning algorithms can provide. However such advantages are not free of constraints. In most cases, we can get the best out of ML algorithms provided that we can feed them a sufficient amount of data, which is not always the case in practice for various reasons, would it be the previous unnecessary or the intrinsic difficulty of capturing such data. This results in a small data set.

We usually say that a data set is small if it contains about thirty to fifty samples [2], [3] which are generally not enough for standard machine learning algorithms to produce a good model, as the data set cannot unveil the whole relations between the features of the population it aims to represent. The case of small sample is not just theoretical; to this day, many medical applications suffer from the lack of big data, such as in early detection of diseases [1], but this problem is also present in industry [2], [3] whether it takes time or resources to generate simulations and acquire data.

1.2 A solution to work with small data : data generation

This lack of data can be compensated for by generating new data, called virtual data, that should be close enough to the real data, but different enough so that it would have a positive impact on the accuracy of the forecasting of the model. This thesis will investigate two techniques of generation: PSOVSG [2] and DEVSG [3], comparing them and hopefully providing some clues on how they work and why or why not they helped in the forecasting. This work aims at providing the reader with an insight of the possibilities of virtual sample generation, more specifically the two techniques that are PSOVSG and DEVSG, and giving some subjects for potential follow-up works.

1.3 Structure of this work

The way to generate this virtual data is the main topic of this thesis, but first, a brief overview of the currently existing techniques to work with small data will be

presented. Then two virtual sample generation algorithms will be detailed; Chapter 3 presents the preliminary step needed for the two techniques detailed in Chapter 4. A brief reminder and discussion about the models used is given in Chapter 5, to follow with the results in Chapter 6. These results are summarized in Chapter 7, concluding this thesis. Complementary figures and details can be found in the Appendix, and the code to reproduce some of the figures is given in the Annexes. All the code for this thesis was written in the Python language [6].

Chapter 2

Dealing with small data : state of the art

Since small data tends to be problematic, it has been under study for some time and solutions were developed. We present here a brief overview of the current techniques, transfer learning and virtual sample generation, and put more focus on the virtual sample generation. More precisely, the aim of this work is to assess how effective virtual sample generation can be, specifically looking at two generation techniques, explaining them and comparing them.

2.1 Transfer Learning and VSG

Even though small data sets are quite common, there exist ways to overcome their flaws. They are currently divided in two families of approaches: virtual sample generation (VSG) and transfer learning (TL). Transfer learning considers that it is feasible to use some previously acquired knowledge from another domain to help in the domain of interest. Take someone who has already learned to play piano, it will probably be easier for him to learn a new instrument because he already knows some theory, how to read a partition, than someone who starts from scratch. This concept has already been proven efficient in the context of convolutional neural networks (CNN). A convolutional neural network is an artificial neural network (ANN) more specifically designed to work with images, by using intrinsic properties of images in its architecture. The architecture of an ANN is supposed to mimic that of the brain [7], [8]; and is depicted in Fig. (2.1), coming from [9]. In such a context, transfer learning would consist, for example, in freezing the first hidden layer of a CNN having been trained on a quite large data set, supposing thus that it is able to capture well the key features of an image; and train the second hidden layer on the desired task, consequently reducing the difficulty of said task. A more

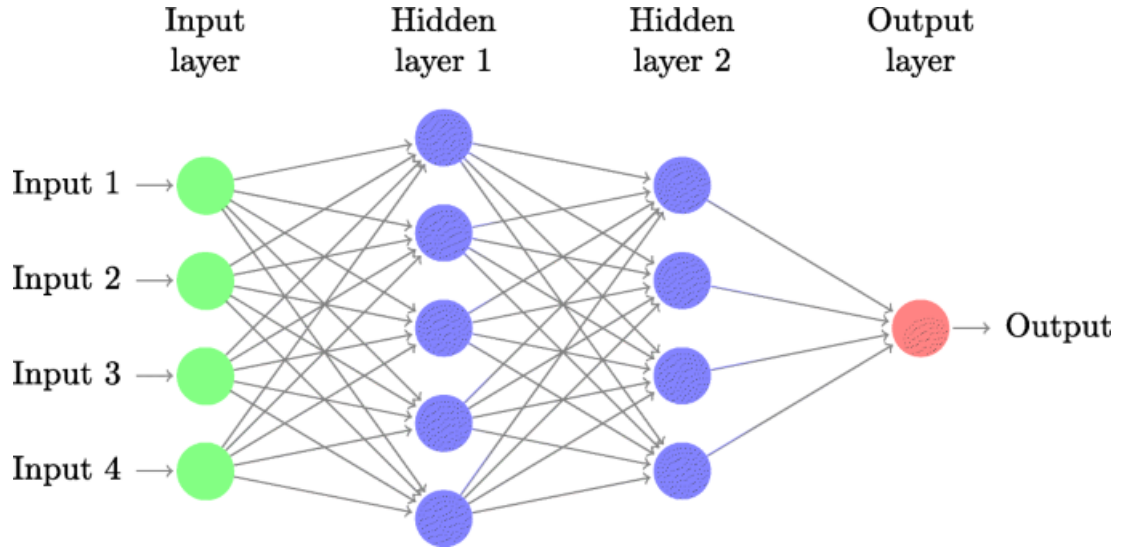


Figure 2.1: A example of an ANN structure

formal definition is that, given a source domain $\mathcal{D}_s$ and a corresponding task $\mathcal{T}_s$, a target domain $\mathcal{D}_t$ and its task $\mathcal{T}_t$, transfer learning tries to improve the target predictive function $f_t()$ based on the information from $\mathcal{D}_s$ and $\mathcal{T}_s$, where either $\mathcal{D}_s \neq \mathcal{D}_t$ or $\mathcal{T}_s \neq \mathcal{T}_t$. A domain $\mathcal{D}$ is defined by a feature space $\mathcal{X}$, along with its marginal probability distribution $P(X)$ where $X = \{x_1, x_2, \dots, x_n\} \in \mathcal{X}$. A task $\mathcal{T}$ is defined by a label space $\mathcal{Y}$ and a predictive function $f()$ learned from the training data $\{x_i, y_i\}$, where $x_i \in X$ and $y_i \in \mathcal{Y}$ [10].

Virtual sample generation, on its part, tries to determine the overall tendency of a small data set in order to generate new samples that would best fit this tendency, meaning that they would likely come from the exact same source and could be actually observed. This is done either by attempting to extract some knowledge from the current data, or perturbing with some noise the original data.

2.2 Focusing on VSG

Transfer learning, while potentially powerful, is too dependant either of the task or the domain. According to the definition, at least one of these have to remain the same between the source and the target. It is thus constraining in the application and generalization. VSG however, could be applied to two different domains, with different tasks, for example, predicting the value of a simple known function and predicting the relative permittivity of a ceramic capacitor, and still provide a meaningful comparison in their performance. This is the reason of the focus of this thesis on virtual sample generation, more precisely, with methods designed

to extract some knowledge and relations from the original data. There exist an handful of such methods, but the choice was ported towards particle swarm optimisation VSG and differential evolution VSG, as they are inspired from known and efficient optimization processes. We will see that these two techniques have a lot in common in the concepts and idea behind them. Therefore they are well suited for a comparison.

Chapter 3

Domain range expansion

When working with small data sets, we can make the assumption that the data we have at our disposal is not big enough, is not a full representation of the reality, we need to complete it with data we will generate. It is then fathomable that we may need to expand the current observed domain boundaries, given that the less data we have, the less sure we are these boundaries are representative enough. There are different techniques of domain range expansions, more or less based on the same concepts. In this thesis, the focus was made on MD-MTD (multi-distribution mega-trend-diffusion) and TMIE (triangular membership information-expanded). While the concepts behind those are similar, we will see they differ in some quite important matters. In order to simplify the notations of our equations, they were written considering only one dimension. When there are many, we only have to apply these equations to each of the dimensions. Following the same idea, the data used to produce Fig. (3.1) and Fig. (3.2) only has one dimension.

3.1 Multi-distribution mega-trend-diffusion

As the name infers, MD-MTD is based on mega-trend-diffusion (MTD) but is able to better describe characteristics of data with complex distribution, and lessens the impact of the central point [3], [11]. The expanded boundaries are, respectively upper (UB) and lower (LB) :

$$UB = \begin{cases} CL + Sk_U \times \sqrt{-2 \frac{\hat{S}_x}{N_U} \times \ln 10^{-20}} & UB \geq max \\ max & UB < max \end{cases} \quad (3.1)$$

$$LB = \begin{cases} CL - Sk_L \times \sqrt{-2 \frac{\hat{S}_x}{N_L} \times \ln 10^{-20}} & LB \leq min \\ min & LB > min \end{cases} \quad (3.2)$$

where CL is the central location of the data; $\hat{S}_x$ is the variance of the data; Sk_U and Sk_L the right and left skewness, accordingly, N_U and N_L , the number of samples at the right or left of CL ; and, of course, max and min the maximum and minimum of the data. In [3], CL is defined as the median, which is an odd choice as it would imply that N_U and N_L would be very close to each other. In other works, CL is defined as the mean, or similar such as $CL = (max + min)/2$. That is,

$$CL = \begin{cases} x_{[(n+1)/2]} & \text{if } n \text{ is odd} \\ \frac{1}{2}(x_{[n/2]} + x_{[n/2+1]}) & \text{if } n \text{ is even} \end{cases} \quad (3.3)$$

n being the size of the data set, and assuming that the x_i 's are sorted, for $i = 1, \dots, n$. The expression of the variance is well known :

$$\hat{S}_x = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1} \quad (3.4)$$

$\bar{x}$ is the mean value of the x_i 's. The right and left skewness are defined as :

$$Sk_U = \frac{N_U}{N_U + N_L + m} \quad (3.5)$$

$$Sk_L = \frac{N_L}{N_U + N_L + m} \quad (3.6)$$

where N_U and N_L are respectively the number of x_i 's greater and lower than the median CL , and m is a correction factor which is set to $m = n/10$ [3]. The skewness represents the proportion of the data that is either right or left to the CL . It measures the asymmetry of the current data thus indicating whether the data points are concentrated near the CL or not, giving then an order of magnitude of the boundaries expansion needed though regulated by the division by N_U or N_L . A detailed explanation of the square root of Eq. (3.1) and Eq. (3.2) is given in [12], however it is important to point out that due to the presence of N_U or N_L in the denominator, the larger the data at our disposal, the lower the value of the square root and consequently, the computed bounds would result in a shrinkage instead of an expansion, hence the need of the second case in Eq. (3.1) and Eq. (3.2). In other words, the more data we have, the more precise our sample of the reality is, and the less we need to expand the boundaries. This is depicted in Fig. (3.1) where the blues dots are the data points, the green one the central location, and the vertical bars the boundaries.

3.2 Triangular membership information-expanded

TMIE is inspired by MTD and expands the domain in a similar fashion. The measure of skewness takes also part in the expansion but is slightly modified from

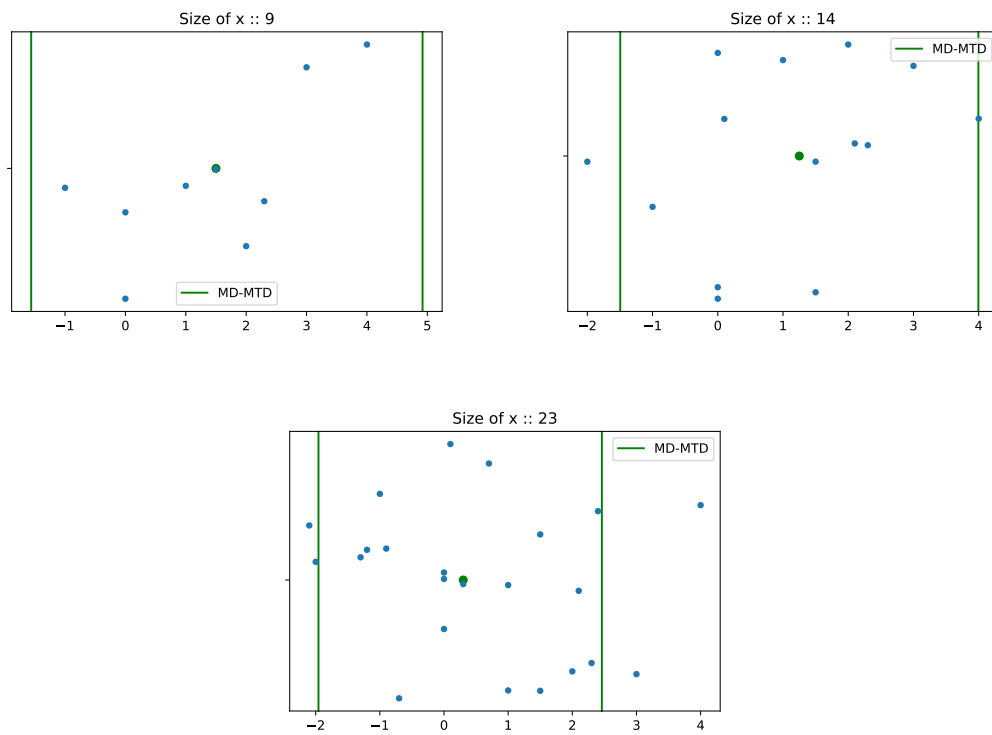


Figure 3.1: The shrinkage effect of MD-MTD with increasing number of data

Eq. (3.5) and Eq. (3.6) with the correction factor being now equal to $m = 1$, as a default value. There exist many other measures of skewness however they generally require a quite large amount of data [13]. This slight modification made, the new boundaries are expressed as follows :

$$UB = CL + \frac{1}{Sk_L} \times (max - CL) \quad (3.7)$$

$$LB = CL - \frac{1}{Sk_U} \times (CL - min) \quad (3.8)$$

with CL being the mean of the observations and not the median as in MD-MTD. The idea behind equations Eq. (3.7) and Eq. (3.8) is to extend the current domain based on the inverse of the proportions of distribution of the data points. Since Sk_U and Sk_L are proportions which sums to $Sk_L + Sk_U \approx 1$, expanding UB proportionally to Sk_U (as in MTD and Eq. (3.1)) is somewhat equivalent to expanding it proportionally to $1/Sk_L$ as the bigger Sk_U , the lower Sk_L thus the bigger $1/Sk_L$. Eq. (3.9) shows us that, since $Sk_L + Sk_U \approx 1$:

$$\frac{1}{Sk_L} \approx \frac{1}{1 - Sk_U} \quad (3.9)$$

and by the first order of Taylor's Series development for $\frac{1}{1-x}$, $|x| < 1$:

$$\frac{1}{1 - Sk_U} \approx 1 + Sk_U$$

So $1/Sk_L$ is indeed proportional to Sk_U .

3.3 Comparison of TMIE and MD-MTD

However, MD-MTD takes into account, on one hand, the variance of the data $\hat{S}_x$ and on the other hand, the number of samples in each part of the domain, regulating even more the expansion with respect to the distribution of the data, possibly to the point where we don't need to expand anymore. A contrario, TMIE expands the boundaries no matter the number of points we have or their distribution as Eq. (3.7) and Eq. (3.8) show an expansion only based on the inverse of skewness and the difference between the actual boundaries (min , max) and the mean CL . Fig. (3.2) shows the comparison of the two methods and illustrates clearly the continuous boundaries expansion of TMIE whereas MD-MTD produces a shrinkage when a sufficient amount of data is reached. The notion of unnecessary to further expand the current boundaries when the number of samples is sufficient is recurrent in many other domain expansion techniques, such as in [14] and their Latent Information (LI) function, and is a key factor showing the inefficacy of TMIE domain expansion.

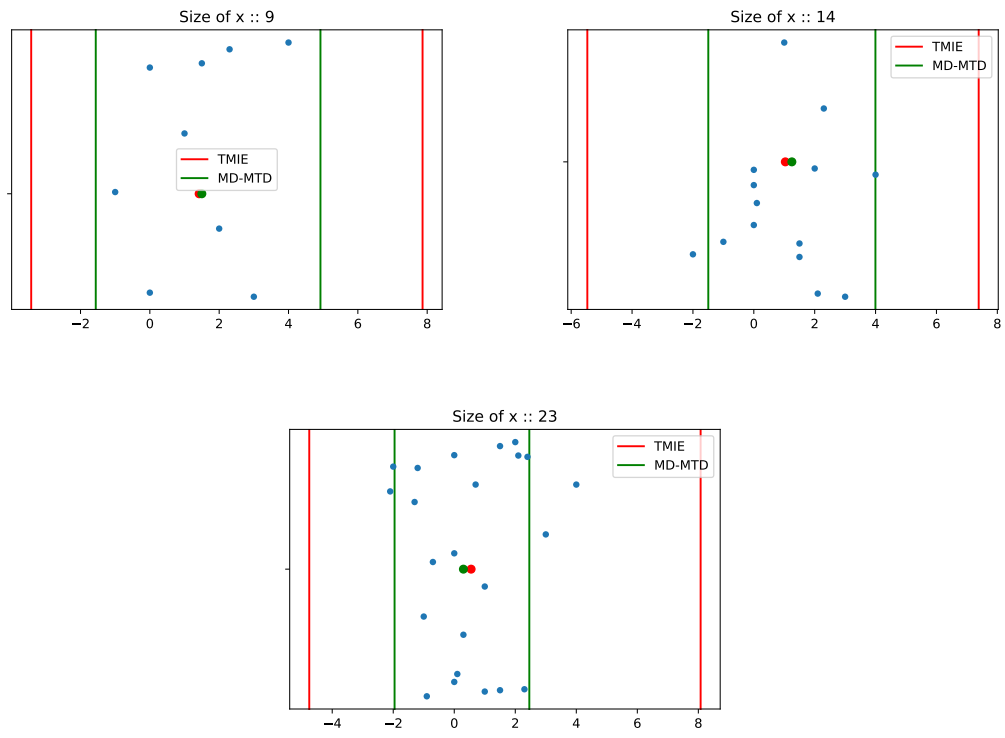


Figure 3.2: Comparison of the boundaries with increasing number of data

Chapter 4

Virtual sample generation

When working with small data, we sometimes may need to expand the observed domain since the few samples we have are more than likely not fully representative of the reality; although this step is not always needed as Eq. (3.1) and Eq. (3.2) show. Once we have defined the boundaries of the space which should represent the reality the best according to Chapter 3, we can begin to generate new samples in that space, samples that should further help the ML model in its prediction. Again, two different techniques are investigated, as well as a "simple" one which will serve as a baseline. For the sake of simplicity, the number of new virtual samples we wish to generate will be called n_{vir} in each of the virtual sample generation (VSG) techniques.

4.1 Particle swarm optimisation generation

Particle swarm optimisation (PSO) is a process of optimisation inspired by the swarm effects witnessed in nature, a flock of birds, or a school of fish; but is related to evolutionary algorithms such as genetic ones (GA). It was developed in 1995 by Kennedy and Eberhart [15] and presented as a new method for optimization, requiring no complex operations. It does not use the genetic operators of GA's that are crossover and mutation [16] but is based on the hypothesis that the sharing of information among a population offers an evolutionary advantage [15]. The process of optimisation can be represented with randomly generated points called particles, having a velocity. These points will explore the space, updating their position based on their velocity and information from the current swarm. The fitness of each particle will be evaluated at each iteration, defining the personal best position for each particle $pbest_i(t)$ as well as the global best position so far $gbest(t)$ for t the current iteration. PSO is currently widely used, for various application, such as weight optimisation of a neural network [16], multi-objective function optimisation

[17], function optimisation in a noisy environment [18] and many others. As it is an iterative process, PSO needs one or more stopping criteria, in our case, once a specific value of fitness is reached, or when the maximum number of iteration is attained. The main equations of PSO, defining the update of position and velocity of the particles, are the following :

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (4.1)$$

$$v_i(t+1) = w(t)v_i(t) + c_1r_1(pbest_i(t) - x_i(t)) + c_2r_2(gbest(t) - x_i(t)) \quad (4.2)$$

$x_i(t)$ is the position of the i -th particle at time t , $v_i(t)$ its velocity, $w(t)$ is called the inertia weight and serves as a regulation of the effect of the velocity over time, typically, a decrease. c_1 and c_2 are two constants called the cognitive and social factor, both set to 2 [19]. Cognitive as it is related to the personal best position so far, $pbest_i(t)$, and social because of the global best position $gbest(t)$ which can be seen as information shared among the population, the "society". r_1 and r_2 are two random vectors consisting of d random numbers independently distributed between 0 and 1; d being the dimension of the particles, the number of features in the data set, for more of a machine learning approach. The weight is adapted with respect to the number of iterations in the following manner :

$$w(t) = w_{end} + (w_{start} - w_{end})\frac{T-t}{T} \quad (4.3)$$

The values w_{start} from which $w(t)$ will linearly decrease until reaching w_{end} are set to, respectively, 0.9 and 0.4 in accordance with [20]. T is the maximum number of iteration, set to 500. In order to update $pbest_i(t)$ and $gbest(t)$, at each iteration, the fitness of each particle will be computed, and $pbest_i(t)$ and $gbest(t)$ will be set accordingly. The fitness function is based on the MAPE (mean absolute percentage error) and defined as follows :

$$f(x) = \min_{y_j} 100 \times \left| \frac{y_j - \hat{y}}{y_j} \right| \quad (4.4)$$

where y_j is one of the y in the original data set, that is, j is an integer between 0 and n , the size of the real data; and $\hat{y}$ is the predicted value for x by the model, $\hat{y} = Model(x)$, where *Model* will be discussed in Chapter 5. The use of the model in the fitness function, thus in the virtual sample generation process, is justified by the supposition that the model, previously trained on a fraction of the original data, has already acquired part of the relationships between the features of the input x and the output y [2]. Now that the details of the equations of PSO are unveiled, the complete process is described in Alg. (4.1).

Algorithm 4.1: PSO virtual sample generation (PSOVSG) process

Result: $new_samples$: a set of n_{vir} new generated samples
let $Model$ a pre-trained ML model such that $\hat{y} = Model(x)$;
let UB and LB the expanded boundaries of the space according to Eq. (3.1) and Eq. (3.2) ;
 $T \leftarrow 500$;
 $c_1 \leftarrow 2$;
 $c_2 \leftarrow 2$;
 $w_{start} \leftarrow 0.9$;
 $w_{end} \leftarrow 0.4$;
 $\epsilon_{tol} \leftarrow 10^{-6}$;
 $new_samples \leftarrow \{\}$;
while $size(new_samples) < n_{vir}$ **do**
 generate 50 particles randomly in the space defined by UB and LB ,
 with a random velocity;
 $gbest \leftarrow +\infty$;
 $t \leftarrow 0$;
 repeat
 $fit_t \leftarrow 0$;
 for $particle \leftarrow 0$ **to** 49 **do**
 compute $f(x_{particle})$ according to Eq. (4.4) ;
 $fit_t \leftarrow fit_t + f(x_{particle})$;
 $pbest_{particle} \leftarrow f(x_{particle})$;
 $gbest \leftarrow \min(gbest, pbest_{particle})$;
 end
 compute w according to Eq. (4.3) ;
 for $particle \leftarrow 0$ **to** 49 **do**
 update $v_{particle}$ according to Eq. (4.2) ;
 update $x_{particle}$ according to Eq. (4.1) ;
 end
 $t \leftarrow t + 1$;
 until $(t \geq T) \vee (fit_t \leq \epsilon_{tol}) \vee (|fit_t - fit_{t-50}| < \epsilon_{tol})$;
 $new_data \leftarrow \operatorname{argmin}_{x_{particle}} f(x_{particle})$;
 $new_samples \leftarrow new_samples \cup new_data$;
end

4.2 VSG based on differential evolution

Differential Evolution (DE) is another evolutionary optimisation algorithm, this time based on genetic algorithms (GA), and proposed by Storn and Price [21]. It is a stochastic process designed to resemble the evolution as seen in nature with concepts like mutation and crossover, following the natural selection concept by preserving the individuals that adapted the best. At each iteration and for every candidate, it will select three parents among the population, different from the current candidate, and produce a new individual with a linear combination of them. The new individual and the current candidate will be inspected allele by allele in a biological approach, feature by feature in a ML one, and the allele of the new individual will be replaced by that of the candidate according to some probability. The final new individual is thus a mixture of the current candidate and the linear combination of the three parents, and will be compared to the current candidate; the algorithm keeping the one with the best fitness among them, a greedy selection. DE is an iterative process that will need to be stopped either when the maximum number of iterations T is attained, or once it reached a sufficient level of fitness. The fitness function, for the sake of comparison, will be the same as for PSO in Eq. (4.4), as [3] did not provide an understandable definition of the one used. The four main steps of the optimisation process are Initialisation, Mutation, Crossover and Selection, and are detailed as follows.

Initialisation : n_{vir} new virtual samples will be generated at random in the domain of interest, according to the following :

$$x_i(t = 0) = LB + r_i \times (UB - LB) \quad (4.5)$$

where x_i are the n_{vir} newly generated virtual samples, UB and LB the expanded boundaries from Eq. (3.1) and Eq. (3.2) respectively, r_i are random vectors of independently distributed random numbers between 0 and 1, t is the current iteration, that is, 0 for the initialisation.

Mutation : At each iteration t and for each individual $x_i(t)$, we randomly select from the other $n_{vir} - 1$ individuals three parents, namely, $x_{p1}(t)$, $x_{p2}(t)$ and $x_{p3}(t)$, with $i \neq p1 \neq p2 \neq p3$. Then, we combine the three parents in the following manner :

$$h_i(t) = x_{p1}(t) + F \times (x_{p2}(t) - x_{p3}(t)) \quad (4.6)$$

with F a scaling factor between 0 and 2, taken as 1.2, that regulates the global locating ability. In other words, a small F will be more exploitative, we will converge faster to the local optimum, while a larger F will make the search more explorative, better capable to jump out of local minima to find the global one, at the cost of a slower convergence.

Crossover : In order to improve the diversity of the population, we apply crossing on the individual $x_i(t)$ and the combination $h_i(t)$ from Eq. (4.6). Each

allele will be set according to some probability, preferring the corresponding allele of $x_i(t)$ or that of $h_i(t)$. This is translated as follows :

$$v_{i,j}(t) = \begin{cases} h_{i,j}(t) & \text{if } rand(0, 1) \leq P_{cr} \\ x_{i,j}(t) & \text{otherwise} \end{cases} \quad (4.7)$$

where $rand(0, 1)$ is a random number uniformly distributed between 0 and 1, and P_{cr} is the probability of crossover, equal to $P_{cr} = 0.7$ here [3]. $v_{i,j}(t)$, $h_{i,j}(t)$ and $x_{i,j}(t)$ mean the j -th allele of the individual $v_i(t)$, $h_i(t)$ and $x_i(t)$, j being between 0 and d , the total number of features.

Selection : We will greedily select the individual to keep between the original $x_i(t)$ and the crossover $v_i(t)$ based on their fitness, computed with Eq. (4.4). This makes sure that the new individual $x_i(t + 1)$ is better than its predecessor $x_i(t)$, resulting, when applied to each i from 0 to n_{vir} , in a set closer to the optimal solution as the mutation and crossover steps help jumping out of local extrema. That is,

$$x_i(t + 1) = \begin{cases} v_i(t) & \text{if } f(v_i(t)) < f(x_i(t)) \\ x_i(t) & \text{otherwise} \end{cases} \quad (4.8)$$

We can furthermore translate all these steps in Alg. (4.2) to show the entire process.

Algorithm 4.2: DE virtual sample generation (DEVSG) process

Result: *new_samples* : a set of n_{vir} new generated samples
let *Model* a pre-trained ML model such that $\hat{y} = Model(x)$;
let *UB* and *LB* the expanded boundaries of the space according to Eq. (3.1) and Eq. (3.2) ;
let *d* the number of dimension of the problem, the number of features in the original data ;
 $T \leftarrow 500$;
 $F \leftarrow 1.2$;
 $P_{cr} \leftarrow 0.7$;
 $\epsilon_{tol} \leftarrow 10^{-6}$;
 $x \leftarrow \{\}$;
for $i \leftarrow 0$ **to** n_{vir} **do**
 $x_i = LB + r_i \times (UB - LB)$ from Eq. (4.5) ;
 $x \leftarrow x \cup x_i$;
end
 $t \leftarrow 0$;
repeat
 $fit \leftarrow 0$;
 for $i \leftarrow 0$ **to** n_{vir} **do**
 do
 $p1 = rand(0, n_{vir})$;
 while $p1 = i$;
 do
 $p2 = rand(0, n_{vir})$;
 while $(p2 = p1) \vee (p2 = i)$;
 do
 $p3 = rand(0, n_{vir})$;
 while $(p3 = p2) \vee (p3 = p1) \vee (p3 = i)$;
 $h_i \leftarrow x_{p1} + F \times (x_{p2} - x_{p3})$ from Eq. (4.6) ;
 $v_i \leftarrow h_i$;
 for $j \leftarrow 0$ **to** d **do**
 if $rand(0, 1) \leq P_{cr}$ **then**
 $v_{i,j} \leftarrow h_{i,j}$;
 else
 $v_{i,j} \leftarrow x_{i,j}$;
 end
 from Eq. (4.7) ;
 end
 if $f(v_i) < f(x_i)$ **then**
 $x_i \leftarrow v_i$;
 end
 from Eq. (4.8) ;
 $fit \leftarrow fit + f(x_i)$;
 end
 $t \leftarrow t + 1$;
until $(t \geq T) \vee (fit \leq \epsilon_{tol})$;
new_samples $\leftarrow x$;

4.3 Differences and similarities

Both PSO and DE are evolutionary optimisation algorithms that are widely used. However, they differ quite fundamentally in their process. PSO will create each new sample one by one, where DE will start with an initial random population and breed it to end up with a better set of virtual samples, through the four steps that are initialisation, mutation, crossover and selection. Their behaviour of their own has an impact on the termination criteria assessing that the virtual samples are good enough; for PSO, we can only analyse one newly generated sample at a time, stopping further attempts at improvement once it reached the proper level of fitness, and that has to be repeated for each of the new samples we try to create. In DE however, we assess the measure of fitness on all our set of new samples, stopping once the global fitness is below a certain threshold. They both use a fitness function to estimate how good the new sample they're trying to produce is, PSO in a hope of improving it more, DE to determine whether we keep the newly bred one or discard it. It very well may happen, for DE, that we end up with a set containing few very good samples (with low fitness) and many intermediate ones, simply because the comparison of a new breed and the older samples is one-to-one. The purpose of the mutation and crossover steps is in fact to try to prevent such cases where we are stuck in local extrema. Each of the new samples of PSO however is, among 50 particles, the lowest in fitness (a comparison in one-to-fifty). Thus, depending on the fitness function, we may end up with a set of almost identical newly generated samples, inducing a certain bias in the data. To prevent that, the fitness function should have many global minima or many local minima that are close to the global one.

Of course, the process of generation of virtual samples, would it be by PSO or DE, is only valid for the features of the data, not the output we try to predict. To get the best target value to then re-train the model with more data, we will need to make use of our pre-trained model on the small data we had at our disposal. Let x_{vir} be the set of virtual generated samples, whether by PSO in Alg. (4.1) or DE in Alg. (4.2), then, their output y_{vir} will be computed according to $y_{vir} = Model(x_{vir})$, for a certain *Model* discussed in Chapter 5.

4.4 A baseline

In order to compare the performance of the two methods of VSG described earlier, a third one, simpler in concept, has been investigated. The idea is that we take randomly three points in the original data and we create a new one that will be the centre of the defined triangle. This is depicted in Fig. (4.1) with a two-dimensional data set. Such a method will ensure that we remain within the boundaries of the

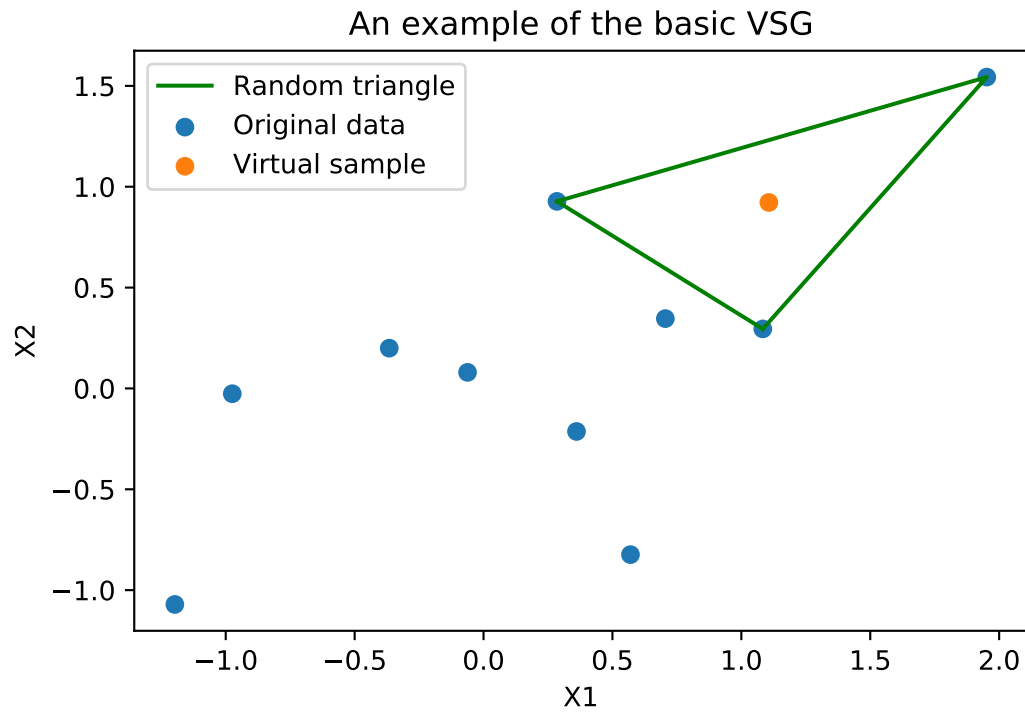


Figure 4.1: Visual representation of the basic VSG technique

original data, not needing the domain expansion from Chapter 3.

Chapter 5

Regression models used

These virtual sample generation techniques from Chapter 4 need a model to help them judge the measure of quality of their new samples. This model plays a key factor in the fitness function of Eq. (4.4), and for it to work best, it should have acquired a good representation of the relations between the input features x and the output y .

The model is really the main piece of any ML application, alongside the preliminary data analysis part. Model selection is a careful decision, for which many tools have been developed such as bootstrap and cross-validation [22]. Also, in typical ML applications, a good practice is to divide the data into three parts: training data, validation data, and test data, where the validation data set is used for the model selection, and the test set to evaluate the final performances of the selected model. In our case, and given the few amount of data available, the initial data set was only separated in train and validation, or train and test, as the main objective is to compare virtual sample generation techniques among them; the final result of the prediction is not the purpose. Model selection was therefore not needed, but nonetheless, we conducted experiments with two different models in order to have a grasp of how the model affects the VSG process. These two models are Random Forest (RF) and K-Nearest Neighbour (KNN) and will be detailed in the following sections.

5.1 Random forest as the predictive model

A random forest (RF) is an ensemble model, a model that combines multiple others to obtain better performances, made of several decision trees. Decision trees (DT) are simple models presenting a tree-like structure, where at each node a selected feature is used for comparison, branching in the left or right sub-tree according to the result of the comparison. The leaves are thus the final values to be output

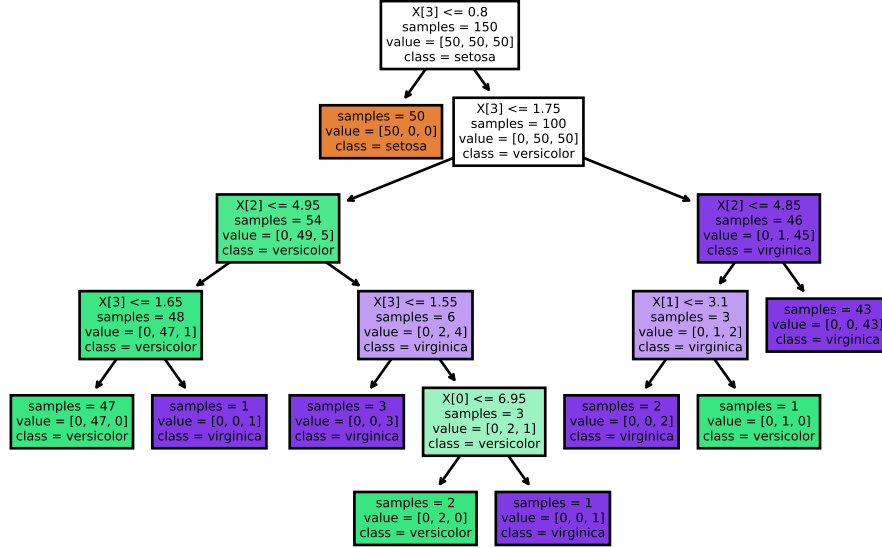


Figure 5.1: A decision tree trained on the iris dataset

as the result of the classification or regression. This is illustrated in Fig. (5.1), in the root of each subtree, we have first the comparison being made selecting one of the four features of the data set, the number of samples being compared, as well as their distribution according to their class, and the target class to be predicted, according to the majority. In the leaves, we only have the number of samples that fall in the category, their distribution, along with the target class. The main disadvantages of DT are the possibility to generate too complex a tree that would result in overfitting, and the fact that they can be unstable, a small variation in the data can result in a completely different prediction, they are of high variance. These problems are addressed in the random forest.

The original definition from L. Breiman states that “a random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(x, \Theta_k), k = 1, \dots\}$ where the Θ_k are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x ” [23]. It means that for each decision tree, a random vector Θ_k is generated which will be used to determine its structure; that is, the split variables, the comparison values at each node, and the final outputs. These trees are then used together, individually predicting their

output, and determining the final output thanks to a majority vote. This definition considers only classification but they can be used also for regression, however, the output will be based on an average of the predictions [24]. The randomness involved in the selection of the features allows to decrease the high variance of decision trees, and to overcome their tendency to overfit the training data [25]. However, such a decrease in variance will sometimes come with a slight increase in bias, though the overall performances are still retained [23]. It is still possible for random forests, as any model, to overfit the data; if the trees are very close to each other in their predictions, the averaging effect will have less impact, and overfitting may happen.

This model was used because of its tendency to work rather well without requiring much fine-tuning of its parameters [25], allowing more time to be put on the VSG process.

5.2 K-nearest neighbors based regression

In an attempt to extract information about the VSG process somewhat independent of the model, another model was used for comparison. K-nearest neighbors (KNN) is a classification or regression model that will predict the output of the target based on its K nearest neighbours. It is based on the assumption that the function we are trying to approximate is somewhat constant in the neighborhood [26]. In case of regression, the model will predict the output from an average of the neighbours, while in classification, a majority vote is used. KNN does not really need training, it only stores the training set points in order to compute the distances with the new point to predict. The distance metric to use is a parameter of KNN and was considered as the Minkowski with parameter $p = 2$, that is, the Euclidean distance. It is also possible to weight the effect of each neighbor; commonly with a uniform one, or inversely proportional to the distance.

This model was used because it requires little time as there is no actual training step, and its simplicity in design makes it easier to understand how it affects the VSG process.

Chapter 6

Results

This chapter will present the results obtained during this thesis along with some hopefully useful insights on them. The models used are those described in Chapter 5, more precisely as implemented by the python module SKLEARN [27], with no parameter tuning at all. In general, the absence of parameter tuning might lead to some overfitting, and indeed, it has, according to Fig. (6.1). In hope to prevent this overfitting, some parameter tuning was investigated, but could unfortunately not solve the issue. In fact, as we only have up to thirty training samples, it is very fathomable that we will always tend to overfit.

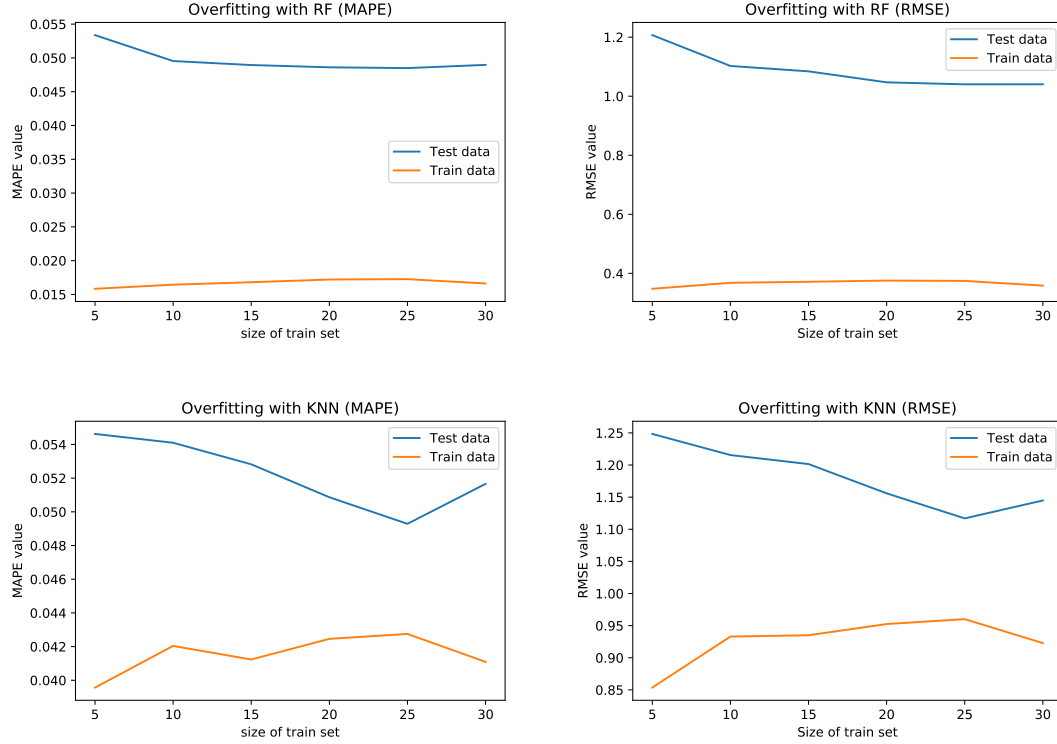


Figure 6.1: The overfitting effect of our models

6.1 Data set used

The data set considered in this thesis comes from [28] and can be found in Table (8.1) in the appendix. This data was used in [2] to assess the performance of their PSOVSG and is representative of a real case industry. It is, of course, a small data set as it contains only 44 samples for 12 features and one output.

For some figures and results, as the twelve features of the said data set don't allow for a clear visual representation, a two-dimensional data set was generated according to the code in Appendix 8.1.1. We generated two correlated features using a bivariate normal distribution with high covariance, and took as target function the sinus of their sum. All graphs and figures that imply a 3D plot used this data instead of the real case one. Although the number of data generated this way is not small anymore (1000), it is still representative of how the algorithms work and can be seen as a generalisation of their behaviour.

6.2 The impact of VSG

Putting altogether the domain expansion technique of MD-MTD discussed in Chapter 3 with the two VSG process PSO and DE from Chapter 4, we can analyse the impact that these virtual samples generation techniques had on the performances of each of our two models. Since we are in a regression model, typical metrics such as precision and recall lose their meaning; instead, we considered the root-mean-squared error (RMSE) and mean absolute percentage error (MAPE) as metrics of comparison. A brief reminder of these two metrics can be found in Eq. (6.1) and Eq. (6.2) :

$$\text{RMSE} = \sqrt{\frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2} \quad (6.1)$$

$$\text{MAPE} = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{|y_i - \hat{y}_i|}{\max(\epsilon, |y_i|)} \quad (6.2)$$

where n_{samples} is the size of the test set, y_i the real value, and $\hat{y}_i$ the predicted value; ϵ is a very small constant to avoid dividing by 0. The comparison is made for each of the VSG techniques described in Chapter 4 : PSO, DE, and the basic one, and also compared to the raw predictions. By raw predictions, we mean the predictions made by the model trained only on the small train set, without any VSG implied. Each experiment was conducted fifteen times and averaged in order to obtain the graphs in Fig. (6.2). These figures show the average evolution of our two measures with respect to the size of the training set; from 5 to 30 samples drawn randomly from the data in Table (8.1). The random drawing is seeded by the run number, that is, the five samples drawn at the very first iteration will be included in the first iteration of drawing the 10 samples, and so on.

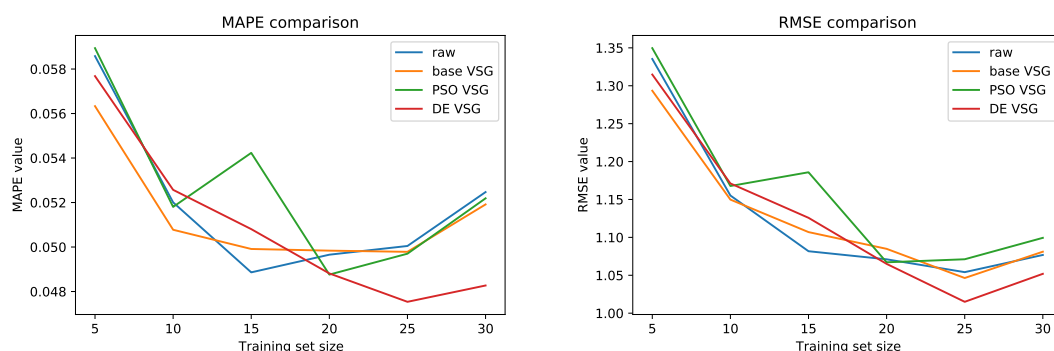


Figure 6.2: Average of 15 predictions for different training set sizes (RF)

As we can see both metrics decrease with the growing number of initial training samples. The small tendency to increase at the last iteration can be explained by the test size, which is always $44 - \text{train_size}$, and will be not representative enough of the trend, as the boxplots of Fig. (6.3) show us. The quartiles for size 30 are larger than at previous steps, indicating a larger variance in the RMSE and MAPE values, more uncertainty. The same effect can be observed for the other techniques, in Appendix 8.2.1.

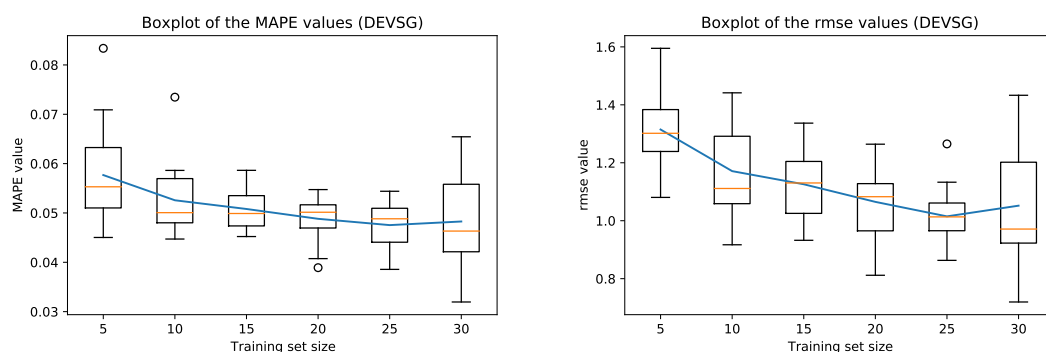


Figure 6.3: Boxplots of the metrics of the 15 experiments of DEVSG, with their average (RF)

Coming back to Fig. (6.2), where the model used was the random forest, but KNN gave similar results (see Fig. (8.4) in the Appendix, along with the corresponding boxplots); we cannot really see an improvement against the raw case, for neither of the VSG methods nor the basic one. This is concerning and in order to find an explanation, we went deeper into the details of the optimisation process, both in PSO and DE.

The fitness function each technique tries to minimize is not easily subject to interpretation, thus we plotted an approximation of it, using a mesh of about 500.000 to 1.000.000 points, equally reparted in the extended boundaries. The resulting is Fig. (6.4) where we can observe some intriguing border effects.

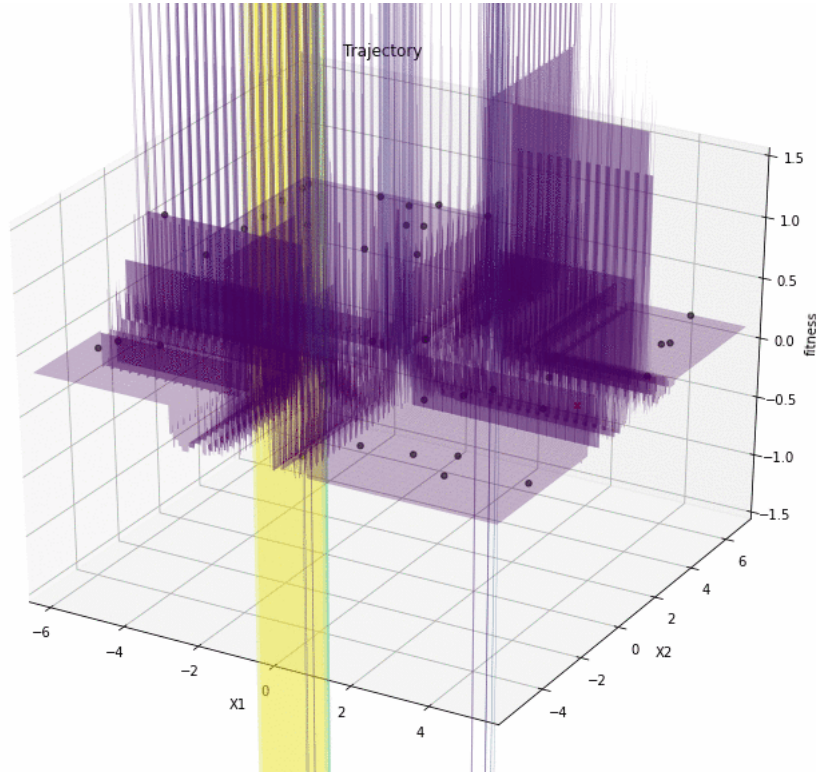


Figure 6.4: Plot of the fitness function described in Eq. (4.4) (RF)

The small oscillations can directly be explained by the formula of the fitness; there exists a finite set of y_j and whether we are close or not to any of these y_j , the resulting fitness will be small or big. The dots on the graph are just the particles of PSO, as it was generated by a built-in function of the PYSWARMS module [29], `plot_surface`¹. The flat ends however cannot really be explained from the fitness function, but are rather consequences of the model chosen, the random forest. Random forests use many decision trees, which work based on ranges; at each split, a certain feature is selected, to be compared to a certain value, branching right or left accordingly. Thus, at some point, the same value will be predicted, passed a certain threshold. The plot of the same mesh, but applied to the predictions of the model in Fig. (6.5) presents the same characteristic.

¹https://pyswarms.readthedocs.io/en/latest/api/pyswarms.utils.plotters.html#pyswarms.utils.plotters.plotters.plot_surface

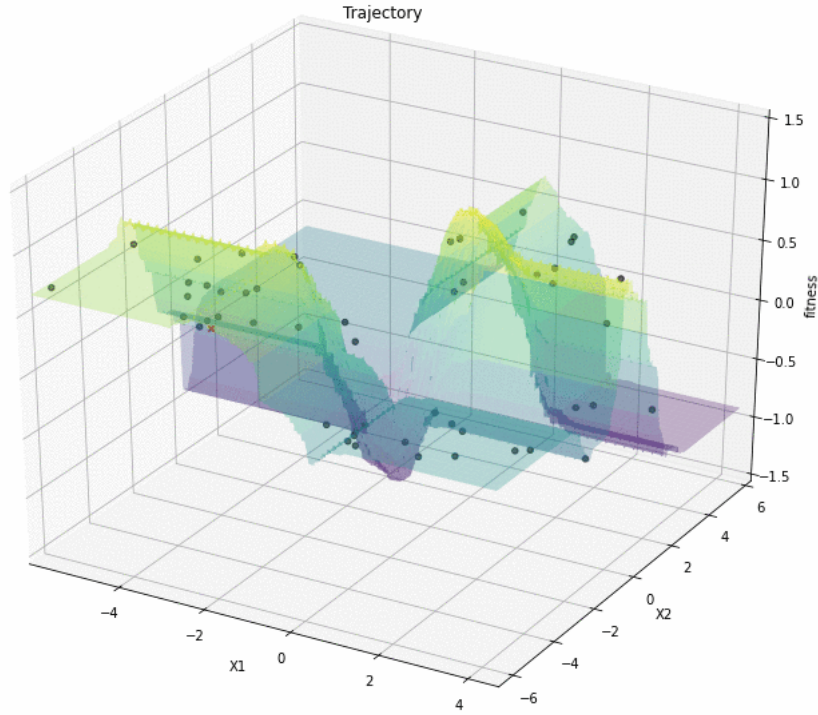


Figure 6.5: Plot of the prediction of RF model ('fitness' should be replaced with 'prediction value')

If we look at the same fitness function, but using the KNN model, we should not see this border effect anymore, instead, as KNN was using Euclidean distances, we should expect the fitness function to be composed of Voronoi regions, each with a center y_j . Despite the absence of clarity of Fig. (6.6) preventing us from clearly seeing the center of the plot, the border effects are quite similar to that of a voronoi diagram (note that the dots are not the centers of the regions, but inherent from the function `plot_surface`), indicating that the two previous assumptions about the impact of the model on the fitness function are correct.

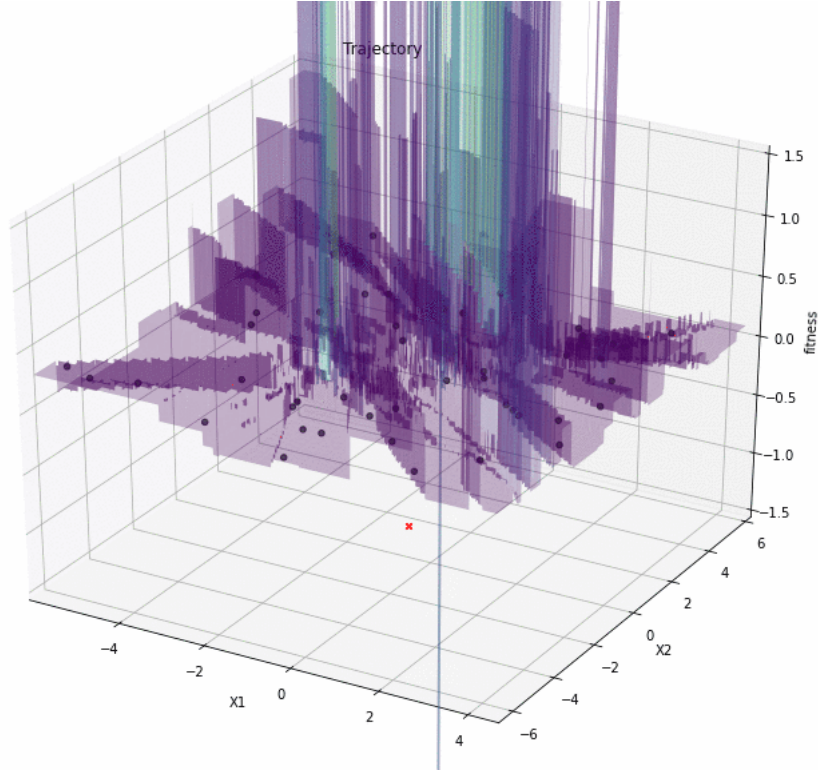


Figure 6.6: Plot of the fitness function described in Eq. (4.4) (KNN)

Even though the estimated output function by RF and by KNN (Fig. (8.9), in the Appendix) differ from one another, and from the exact output function² (Fig. (8.10), in the Appendix), the results on Fig. (6.2) and Fig. (8.4) are similar in value, and the improvement in performance thanks the VSG processes seems underwhelming. This would indicate that the model is not the cause of the bad performances of VSG.

Both particle swarm optimization and differential evolution are optimisation algorithms that have been used multiple times, in various domains, and with success; it would not be reasonable to think their optimisation of the fitness function in Eq. (4.4) is the cause.

Since neither the model nor the optimisation process seems to be defective for the task of virtual sample generation, the reason for poor performances must be inherent to the fitness function. The use of the MAPE in Eq. (4.4) is not really justified; [2] references [30] and [31] in the explanation of the fitness function. In turn, [30] references [31]. Both [2] and [30] explicit that the usage of MAPE is suggested in [31], however, there was no mention of MAPE nor of the concept

²Remember that for these 3D plots, the output function is known and is the sinus of the sum of the two features, $X1$ and $X2$

behind the fitness from Eq. (4.4) in this last paper. Furthermore, one could fathom that using RMSE or another metric would be as justifiable. All these clues would indicate that the main issue preventing the two VSG processes to significantly improve the prediction results is the fitness function itself. Defining a fitness function is a complicated task, but exploring this topic might consist in an interesting follow-up work.

6.3 Normalization

As the fitness function in Eq. (4.4) is adapted from the MAPE, the normalization of the output should have an impact since it plays a role in the denominator. The same experiments as for Fig. (6.2) were conducted, but with beforehand unity-based normalization, depicted in Eq. (6.3). Fig. (6.7) shows us the performance of such normalization, with the RF as the model used; but one can assume similar results with the KNN, as for before.

$$x_{normalized} = \frac{x - x_{min}}{x_{min} - x_{max}} \quad (6.3)$$

We still have a decreasing effect with respect to the size of the training set, as

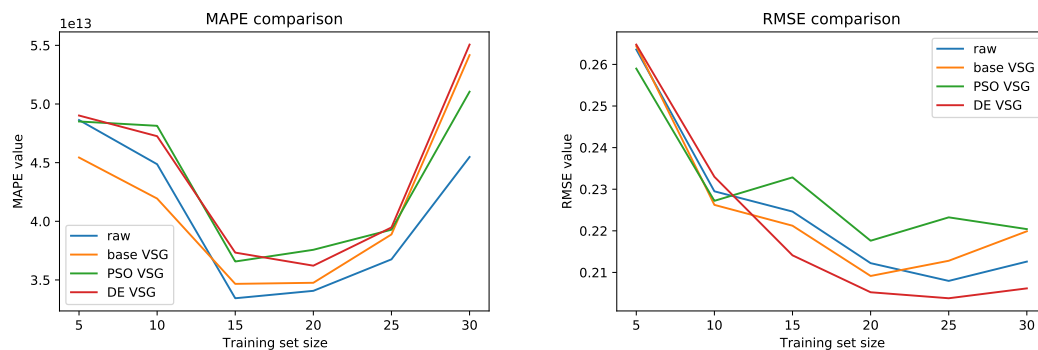


Figure 6.7: Average of 15 predictions for different training set sizes (normalized outputs)

expected for the RMSE. The MAPE however seems to increase much more than previously. Of course, the ranges of the values have changed accordingly to the output, as Eq. (6.1) and Eq. (6.2) suggest. The division by the normalized y_i for the MAPE has indeed had an impact since the values are now past 10^{13} , but this still doesn't explain the increase for larger training size. If we look at the corresponding boxplots in Fig. (6.8) and in Appendix 8.2.4, we see that the division by $y_i \approx 0$ has

had a bigger impact in the quartiles than without normalization, while the effective median is way lower than the mean. Still, Fig. (6.7) shows no improvement between

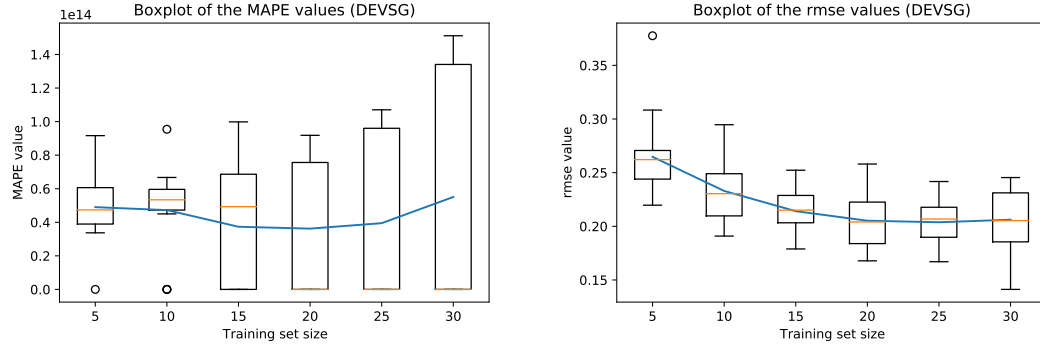


Figure 6.8: Boxplots of the metrics of the 15 experiments of DEVSG, with their average (normalized outputs)

the raw predictions and any of the VSG process, meaning that normalizing the output did not impact the VSG process and assumedly indicating the non-adequacy of the fitness function.

Chapter 7

Conclusion

We detailed in Chapter 4 two processes of virtual sample generation, which sometimes require an expansion of their domain boundaries, as we did in Chapter 3. These VSG techniques used a model that was detailed in Chapter 5. An analysis and comparison of their results was presented in Chapter 6.

7.1 Summary of this work

Although the two VSG processes have not concluded any relevant improvement in the forecasting accuracy of our two models in the case of small data, their procedures have been detailed and the potential causes of failure have been investigated. We have seen that the model, although having an impact on the prediction and the fitness function, did not seem to affect the quality of the generated samples, using two different metrics to assess that. Out of the two domain range expansion techniques, only MD-MTD seemed to be meaningful, but without much impact on the forecasting accuracy. Both optimisation processes were described, without any aberration pointing out. These two processes being widely used in other fields, the more likely cause is the fitness function. A more thorough analysis of the fitness function, as well as of its impacts could be an interesting subsequent work.

Another possible field of search could be to define problem-specific constraints to add to PSOVSG and DEVSG. [2] and [3] mention some constraints $g(x)$ that can be tweaked in the fitness to further improve the quality of the generated samples. These constraints were not given and might be, as well, an explanation of the underwhelming results.

7.2 Small change in the fitness

In an attempt to further investigate the potential improvement in changing the fitness, MAE (mean absolute error) was used in Eq. (4.4) instead of the MAPE, resulting in Eq. (7.1).

$$f(x) = \min_{y_j} 100 \times |y_j - \hat{y}| \quad (7.1)$$

The rest of the process remains the same, for PSOVSG and DEVSG. Again, the same experiments were conducted, with KNN as the model, and the results can be observed in Fig. (7.1).

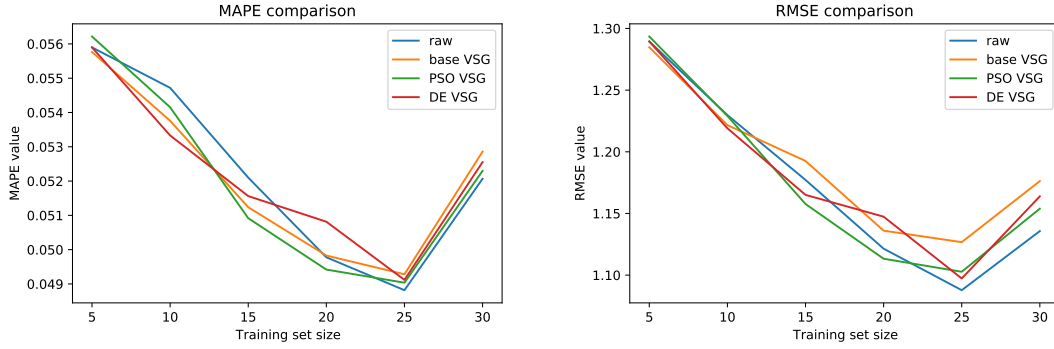


Figure 7.1: Average of 15 predictions for different training set sizes (MAE-based fitness)

The change from MAPE to MAE as the evaluation function in the fitness did not bring any changes, both the ranges of value for our two metrics are the same than from Fig. (6.2) where MAPE was used in the fitness. Furthermore, no improvement between the VSG techniques and the raw or base VSG can be observed. This is an indication that the essence of the fitness is more than likely not efficient. A potential follow-up work could be to analyze why computing the fitness as a certain measure of error between a prediction from a poor model and the closest value in the original data set does not seem to be efficient, and, one step further, define a new fitness appropriate to the virtual sample generation task.

Chapter 8

Appendix

8.1 The data used

8.1.1 Simulated data

```
1 import numpy as np
2 import pandas as pd
3
4 x = np.random.multivariate_normal(mean=[0, 0], cov=[[0.7,
5             0.6],[0.6,0.8]], size=1000)
6 x1 = x[:,0]
7 x2 = x[:,1]
8 y = np.sin(x1 + x2)
9 data = pd.DataFrame(np.array([x1, x2, y]).T, columns=["X1", "X2",
10             "Y"])
```

8.1.2 Real data

	PSD-10	Mois	DF *10 ⁻⁴	TC-min	TC-max	TC peak	4SA	PSD-90	PSD-50	Sinter temp.	K	D-50	RK
1	0.44	0.05	36	-74.3	46.8	-7.3	2.8	2.15	1.0	1297	11.689	0.97	18.092
2	0.44	0.02	40	-73.6	41.8	-7.2	2.9	2.1	0.98	1307	11.142	0.995	17.145
3	0.46	0.02	41	-74.5	42.1	-6.3	2.8	2.05	0.98	1271	10.834	0.94	16.066
4	0.43	0.04	37	-73.8	44.5	-7.2	3.0	2.06	0.95	1292	10.345	0.91	16.188
5	0.45	0.06	28	-73.5	49.3	-9.0	3.0	1.97	0.94	1287	10.286	0.991	16.479
6	0.45	0.06	30	-74.5	47.5	-7.2	3.1	1.97	0.94	1282	10.806	0.905	16.642
7	0.43	0.06	29	-74.3	49.3	-7.4	3.0	1.97	0.92	1286	10.268	0.854	17.153
8	0.47	0.03	29	-74.4	49.3	-7.5	3.2	2.06	0.98	1274	10.904	0.876	15.201
9	0.43	0.05	27	-74.0	48.5	-8.2	3.0	2.11	0.96	1306	10.282	0.958	17.112
10	0.42	0.05	35	-74.2	40.6	-5.5	2.7	2.07	0.94	1268	10.62	0.888	17.577
11	0.42	0.01	30	-73.6	46.5	-6.9	3.1	2.12	0.96	1284	10.539	0.918	17.287
12	0.41	0.02	30	-74.0	43.5	-6.5	3.2	2.15	0.95	1283	10.797	0.843	17.292
13	0.44	0.04	31	-74.2	45.7	-6.8	2.9	2.01	0.95	1272	10.846	0.824	18.302
14	0.45	0.02	30	-73.9	41.9	-6.5	2.9	2.04	0.98	1281	10.533	0.862	16.662
15	0.48	0.01	34	-74.2	41.7	-6.1	3.2	2.0	0.97	1272	11.206	0.879	18.527
16	0.47	0.0	33	-74.3	40.4	-5.6	3.1	1.93	0.94	1268	10.523	0.886	16.606
17	0.5	0.07	36	-74.0	41.1	-5.9	3.3	1.88	0.96	1281	10.232	1.0	18.03
18	0.45	0.01	31	-74.2	45.3	-6.9	3.2	1.89	0.92	1296	10.651	0.883	17.781
19	0.46	0.02	26	-74.0	47.1	-7.7	3.0	1.87	0.94	1300	10.508	1.02	17.537
20	0.46	0.05	29	-74.9	46.1	-6.5	3.0	1.93	0.94	1284	11.003	0.83	16.935
21	0.43	0.08	39	-75.0	43.8	-5.7	3.1	2.01	0.94	1270	11.117	0.908	18.218
22	0.46	0.01	35	-74.0	43.9	-6.8	3.1	1.93	0.94	1292	10.479	0.948	18.181
23	0.5	0.07	23	-74.7	51.3	-7.2	3.1	1.83	0.96	1296	10.875	0.873	17.286
24	0.43	0.04	34	-75.6	49.0	-6.4	3.3	1.98	0.92	1262	11.75	0.917	16.681
25	0.43	0.02	25	-74.8	53.4	-6.7	3.1	2.0	0.93	1270	11.101	0.85	16.603
26	0.47	0.01	40	-74.1	44.3	-5.2	3.0	1.92	0.95	1305	10.493	0.88	18.23
27	0.47	0.05	27	-75.1	52.1	-5.9	2.9	1.92	0.95	1278	11.442	1.03	17.187
28	0.44	0.01	28	-74.5	52.5	-6.8	3.0	1.98	0.94	1269	10.682	0.95	17.047
29	0.5	0.05	22	-74.8	54.0	-6.7	3.0	1.97	0.99	1274	11.008	0.846	17.308
30	0.45	0.05	25	-74.6	54.0	-7.0	3.2	2.04	0.96	1270	10.994	0.914	17.052
31	0.46	0.06	32	-75.0	50.6	-5.8	3.2	1.91	0.93	1273	11.295	0.868	18.624
32	0.5	0.04	28	-74.6	51.4	-6.3	3.1	1.87	0.96	1280	11.018	0.859	17.687
33	0.5	0.02	27	-75.3	58.3	-7.0	3.2	1.82	0.95	1280	11.59	0.955	17.635
34	0.49	0.01	25	-75.6	54.1	-6.4	2.9	1.83	0.95	1281	11.384	0.878	18.623
35	0.49	0.04	24	-75.6	55.2	-6.2	2.6	1.93	0.98	1270	11.513	0.991	20.156
36	0.49	0.02	21	-75.4	55.0	-6.4	2.8	1.91	0.97	1286	10.875	1.062	20.164
37	0.48	0.04	25	-76.4	59.8	-7.0	3.1	1.97	0.97	1293	11.804	0.873	19.84
38	0.48	0.03	26	-76.4	56.1	-6.3	2.9	1.91	0.96	1283	11.388	0.956	20.23
39	0.47	0.03	30	-75.4	55.3	-6.3	2.9	1.94	0.96	1273	12.242	0.874	19.044
40	0.52	0.01	30	-75.1	51.2	-6.1	2.8	1.88	0.99	1288	11.312	0.918	17.427
41	0.51	0.08	36	-74.8	47.3	-5.3	2.8	1.86	0.98	1266	10.858	1.008	17.747
42	0.51	0.01	31	-74.8	50.1	-5.6	3.0	1.87	0.97	1293	11.17	0.929	18.901
43	0.47	0.01	33	-75.2	49.7	-5.2	3.0	1.9	0.94	1292	11.673	0.957	16.64
44	0.48	0.07	45	-75.6	48.2	-5.1	3.1	1.96	0.96	1280	11.647	0.83	17.388

Table 8.1: The data used for sample generation

8.2 Complementary figures

8.2.1 Boxplots of the results

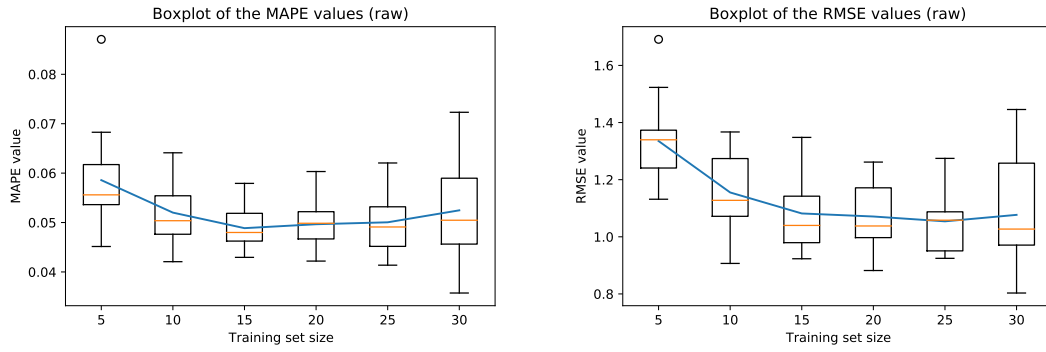


Figure 8.1: Boxplots of the metrics of the 15 raw experiments, with their average (RF)

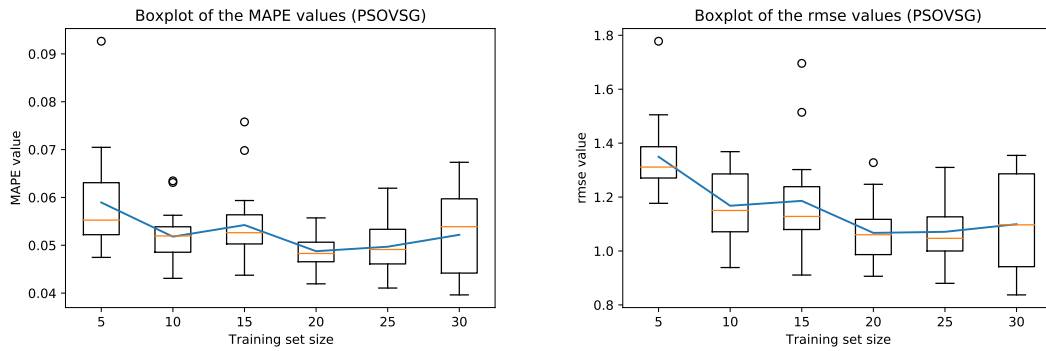


Figure 8.2: Boxplots of the metrics of the 15 experiments of PSOVSG, with their average (RF)

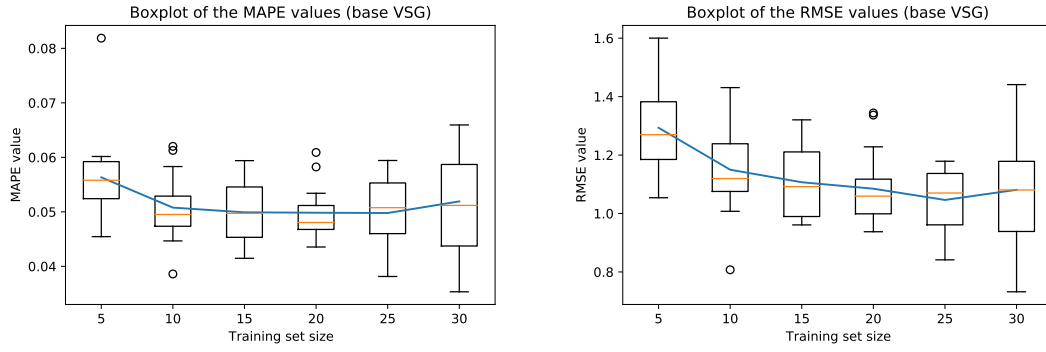


Figure 8.3: Boxplots of the metrics of the 15 experiments of basic VSG, with their average (RF)

8.2.2 KNN model

Results

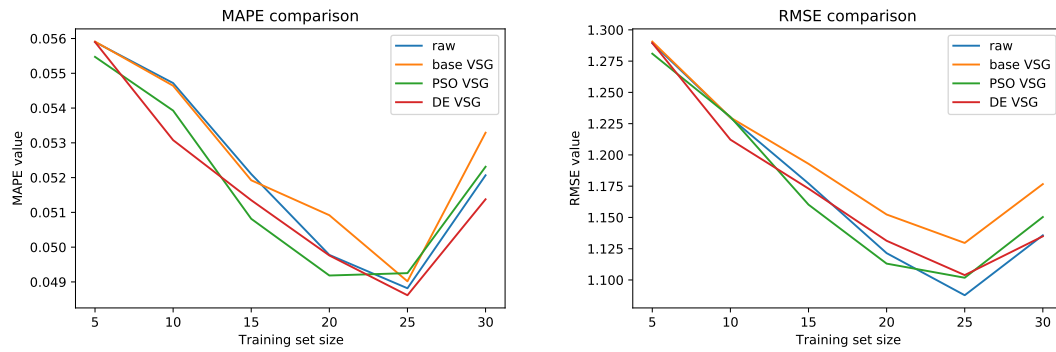


Figure 8.4: Average of 15 predictions for different training set sizes (KNN)

Boxplots

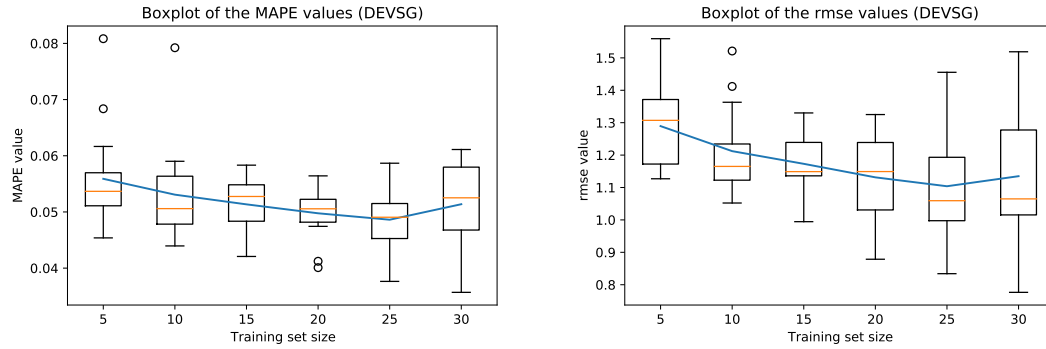


Figure 8.5: Boxplots of the metrics of the 15 experiments of DEVSG, with their average (KNN)

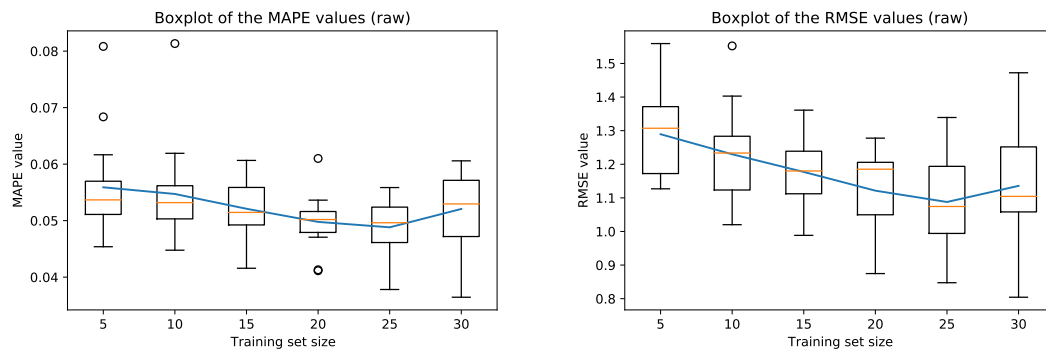


Figure 8.6: Boxplots of the metrics of the 15 raw experiments, with their average (KNN)

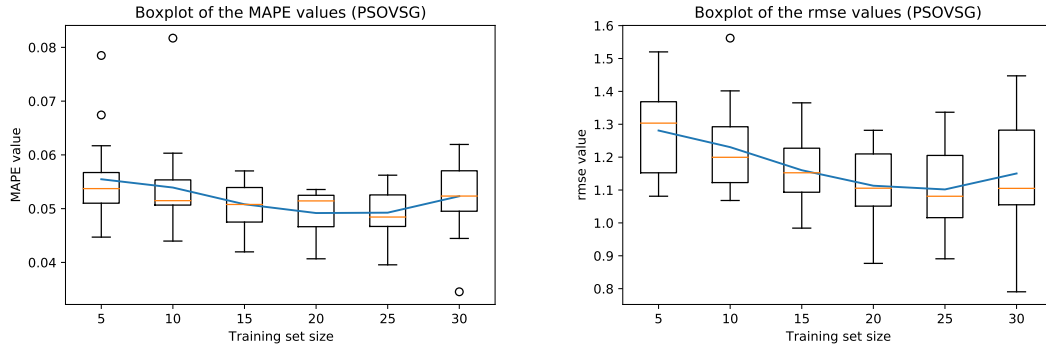


Figure 8.7: Boxplots of the metrics of the 15 experiments of PSOVSG, with their average (KNN)

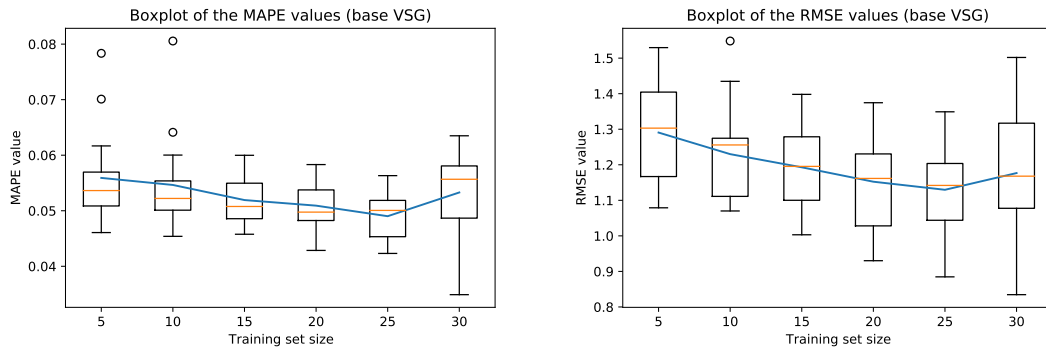


Figure 8.8: Boxplots of the metrics of the 15 experiments of basic VSG, with their average (KNN)

8.2.3 Plots of the output function

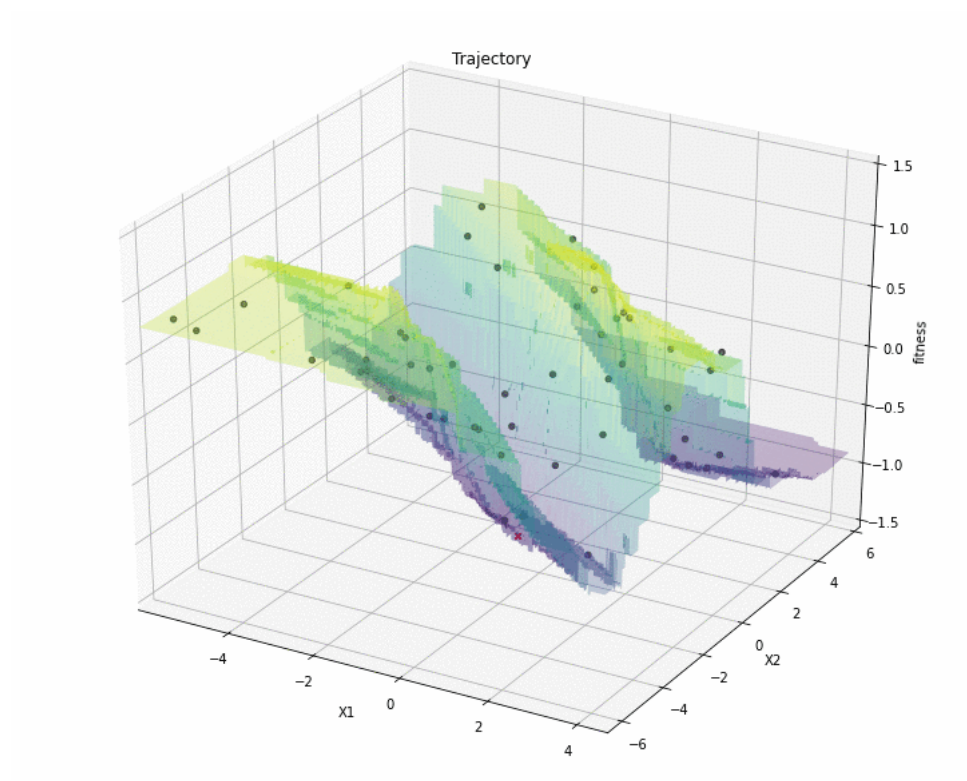


Figure 8.9: Plot of the prediction of KNN model ('fitness' should be replaced with 'prediction value')

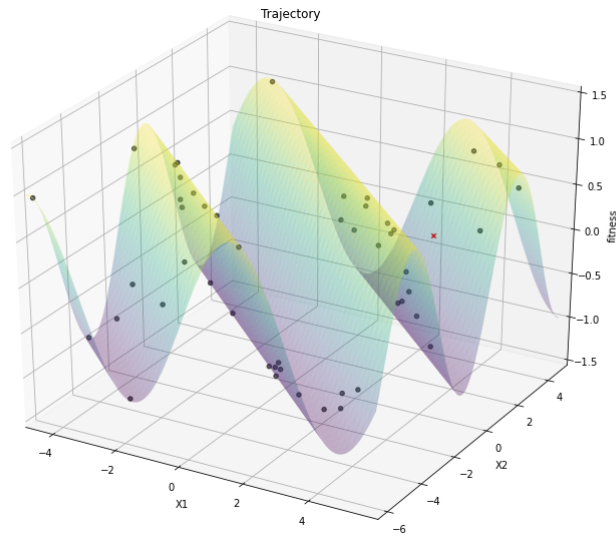


Figure 8.10: Plot of the true output function $Y = \sin(X1 + X2)$ ('fitness' should be replaced with 'true value')

8.2.4 Boxplots of normalized outputs

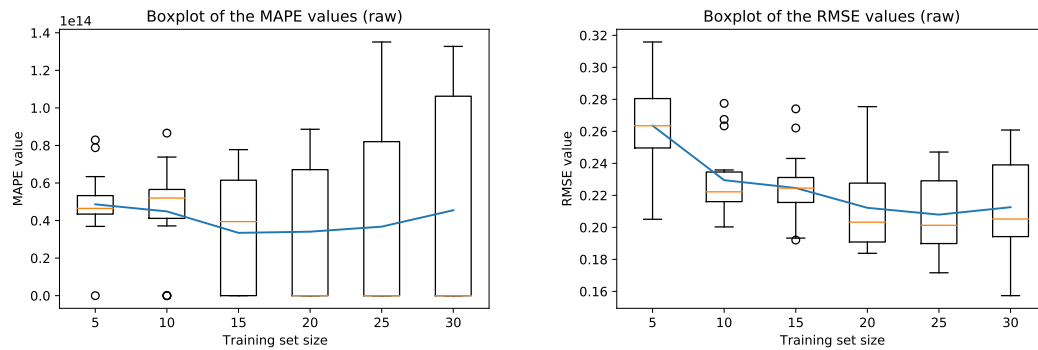


Figure 8.11: Boxplots of the metrics of the 15 raw experiments, with their average (normalized outputs)

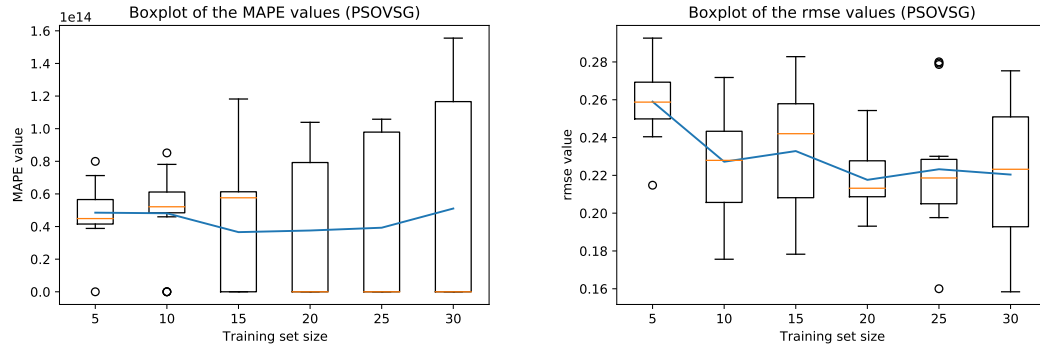


Figure 8.12: Boxplots of the metrics of the 15 experiments of PSOVSG, with their average (normalized outputs)

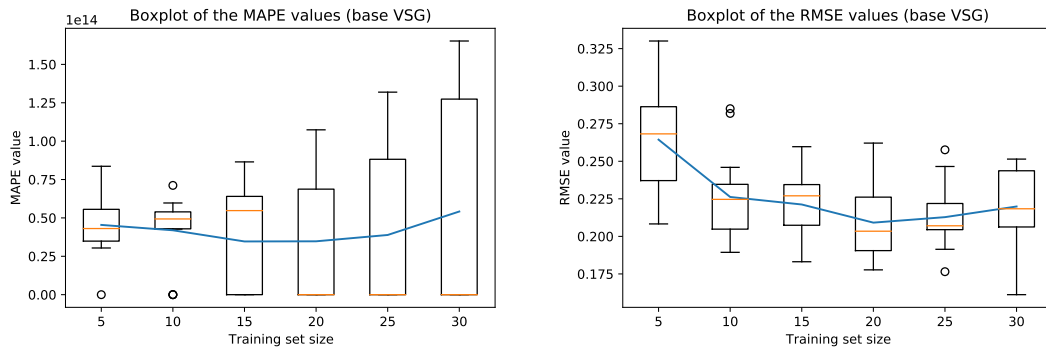


Figure 8.13: Boxplots of the metrics of the 15 experiments of basic VSG, with their average (normalized outputs)

Annexes

Code to produce Fig. (3.1) and Fig. (3.2)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 x = np.array([1,2,-1,0,0,1.5,4,3,2.3])
5 CL = x.mean()
6 N_L = np.count_nonzero(x < CL)
7 N_U = np.count_nonzero(x > CL)
8 sk_L = N_L / (N_L + N_U + 1)
9 sk_U = N_U / (N_L + N_U + 1)
10 LB = CL - 1/sk_U * (CL - x.min())
11 UB = CL + 1/sk_L * (x.max() - CL)
12 CL2 = np.median(x)
13 S_x = np.sum((x - x.mean())**2) / (x.shape[0] - 1)
14 LB2 = CL2 - sk_L * np.sqrt(-2*S_x/N_L * np.log(10**-20))
15 UB2 = CL2 + sk_U * np.sqrt(-2*S_x/N_U * np.log(10**-20))
16 sns.stripplot(x=x)
17 plt.scatter(CL, 0, marker='o', c='r')
18 plt.axvline(LB, c='r', label='TMIE')
19 plt.axvline(UB, c='r')
20 plt.scatter(CL2, 0, marker='o', c='g')
21 plt.axvline(LB2, c='g', label='MD-MTD')
22 plt.axvline(UB2, c='g')
23 plt.title(f"Size of x :: {x.shape[0]}")
24 plt.legend()
25 plt.savefig("small.png")
26 plt.show()
27 #####
28 x = np.array([1,2,-1,0,0,1.5,4,3,2.3, 0.1, 2.1, -2, 1.5,0])
29 CL = x.mean()
30 N_L = np.count_nonzero(x < CL)
31 N_U = np.count_nonzero(x > CL)
32 sk_L = N_L / (N_L + N_U + 1)
33 sk_U = N_U / (N_L + N_U + 1)
34 LB = CL - 1/sk_U * (CL - x.min())
```

```

35 UB = CL + 1/sk_L * (x.max() - CL)
36 CL2 = np.median(x)
37 S_x = np.sum((x - x.mean())**2) / (x.shape[0] - 1)
38 LB2 = CL2 - sk_L * np.sqrt(-2*S_x/N_L * np.log(10**-20))
39 UB2 = CL2 + sk_U * np.sqrt(-2*S_x/N_U * np.log(10**-20))
40 sns.stripplot(x=x)
41 plt.scatter(CL, 0, marker='o', c='r')
42 plt.axvline(LB, c='r', label='TMIE')
43 plt.axvline(UB, c='r')
44 plt.scatter(CL2, 0, marker='o', c='g')
45 plt.axvline(LB2, c='g', label='MD-MTD')
46 plt.axvline(UB2, c='g')
47 plt.title(f"Size of x :: {x.shape[0]}")
48 plt.legend()
49 plt.savefig("medium.png")
50 plt.show()
51 #####
52 x = np.array([1,2,-1,0,0,1.5,4,3,2.3, 0.1, 2.1, -2,
               1.5,0,1,2.4,0.7,-1.2,-1.3,-0.7,0.3,-2.1,-0.9])
53 CL = x.mean()
54 N_L = np.count_nonzero(x < CL)
55 N_U = np.count_nonzero(x > CL)
56 sk_L = N_L / (N_L + N_U + 1)
57 sk_U = N_U / (N_L + N_U + 1)
58 LB = CL - 1/sk_L * (CL - x.min())
59 UB = CL + 1/sk_U * (x.max() - CL)
60 CL2 = np.median(x)
61 S_x = np.sum((x - x.mean())**2) / (x.shape[0] - 1)
62 LB2 = CL2 - sk_L * np.sqrt(-2*S_x/N_L * np.log(10**-20))
63 UB2 = CL2 + sk_U * np.sqrt(-2*S_x/N_U * np.log(10**-20))
64 sns.stripplot(x=x)
65 plt.scatter(CL, 0, marker='o', c='r')
66 plt.axvline(LB, c='r', label='TMIE')
67 plt.axvline(UB, c='r')
68 plt.scatter(CL2, 0, marker='o', c='g')
69 plt.axvline(LB2, c='g', label='MD-MTD')
70 plt.axvline(UB2, c='g')
71 plt.title(f"Size of x :: {x.shape[0]}")
72 plt.legend()
73 plt.savefig("big.png")
74 plt.show()

```

Code to produce Fig. (4.1)

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split

```

```

5
6 x = np.random.multivariate_normal(mean=[0, 0], cov=[[0.7,
    0.6],[0.6,0.8]], size=1000)
7 x1 = x[:,0]
8 x2 = x[:,1]
9 y = np.sin(x1 + x2)
10 data = pd.DataFrame(np.array([x1, x2, y]).T, columns=["X1", "X2",
    "Y"])
11 columns_x = data.columns[data.columns != 'Y']
12 columns_y = data.columns[data.columns == 'Y']
13 data_train, data_test = train_test_split(data, train_size=10)
14 data_new = data_train.copy()
15 index = np.random.choice(data_new.index, size=3, replace=False)
16 nd = data_new.loc[index, columns_x].mean()
17 nd = pd.DataFrame(np.array(nd).reshape(1, -1), columns=columns_x)
18 plt.scatter(data_train['X1'], data_train['X2'], marker='o', label=
    "Original data")
19 plt.plot(data_train.loc[index,'X1'], data_train.loc[index,'X2'],
    color='g', label="Random triangle")
20 plt.plot([data_train.loc[index[2], 'X1'], data_train.loc[index[0], '
    X1']], [data_train.loc[index[2], 'X2'], data_train.loc[index[0],
    'X2']], color='g')
21 plt.scatter(nd['X1'], nd['X2'], marker='o', label="Virtual sample"
    )
22 plt.legend()
23 plt.xlabel("X1")
24 plt.ylabel("X2")
25 plt.title("An example of the basic VSG")
26 plt.savefig("basevsgexample.png")
27 plt.show()

```

Code to produce Fig. (5.1)

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import load_iris
4 from sklearn.tree import DecisionTreeClassifier, plot_tree
5
6 iris = load_iris()
7 clf = DecisionTreeClassifier().fit(iris.data, iris.target)
8 plot_tree(clf, filled=True, impurity=False, class_names=iris.
    target_names)
9 plt.savefig("decisiontree.pdf")
10 plt.show()

```

Code to produce Fig. (6.1)

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from tqdm.notebook import tqdm
5 from sklearn.ensemble import RandomForestRegressor
6 from sklearn.neighbors import KNeighborsRegressor
7 from sklearn.metrics import mean_squared_error,
   mean_absolute_percentage_error
8 from sklearn.model_selection import train_test_split
9
10 data = pd.read_csv("data.csv", decimal='.', index_col=0)
11
12 rmse_test = []
13 mape_test = []
14 rmse_train = []
15 mape_train = []
16 train_sizes = [5, 10, 15, 20, 25, 30]
17 for train_size in train_sizes:
18     r_te = 0
19     m_te = 0
20     r_tr = 0
21     m_tr = 0
22     for run in range(30):
23         data_train, data_test = train_test_split(data, train_size=
train_size, random_state=run)
24         RF = RandomForestRegressor()
25         #RF = KNeighborsRegressor()
26         RF.fit(data_train.loc(axis=1)[columns_x], data_train['RK'])
27         test_prediction = RF.predict(data_test.loc(axis=1)[columns_x])
28         train_prediction = RF.predict(data_train.loc(axis=1)[columns_x
])
29         r_te += np.sqrt(mean_squared_error(data_test.loc(axis=1)[
columns_y], test_prediction))
30         m_te += mean_absolute_percentage_error(data_test.loc(axis=1)[
columns_y], test_prediction)
31         r_tr += np.sqrt(mean_squared_error(data_train.loc(axis=1)[
columns_y], train_prediction))
32         m_tr += mean_absolute_percentage_error(data_train.loc(axis=1)[
columns_y], train_prediction)
33     rmse_test.append(r_te/30)
34     mape_test.append(m_te/30)
35     rmse_train.append(r_tr/30)
36     mape_train.append(m_tr/30)
37
38 plt.plot(train_sizes, rmse_test, label="Test data")
39 plt.plot(train_sizes, rmse_train, label="Train data")
40 plt.legend()
41 plt.title("Overfitting with RF (RMSE)")
42 plt.xlabel("Size of train set")

```

```
43 plt.ylabel("RMSE value")
44 plt.savefig("overfit_rmse_rf.pdf")
45 plt.show()
46 plt.plot(train_sizes, mape_test, label="Test data")
47 plt.plot(train_sizes, mape_train, label="Train data")
48 plt.legend()
49 plt.title("Overfitting with RF (MAPE)")
50 plt.xlabel("size of train set")
51 plt.ylabel("MAPE value")
52 plt.savefig("overfit_mape_rf.pdf")
53 plt.show()
```

Entire code used for the thesis

The full code can be made available on demand to thomasreniers98@gmail.com.

Bibliography

- [1] Mohammad Wedyan, Alessandro Crippa, and Adel Al-Jumaily. A novel virtual sample generation method to overcome the small sample size problem in computer aided medical diagnosing. *Algorithms*, 12(8), 2019.
- [2] Zhong-Sheng Chen, Bao Zhu, Yan-Lin He, and Le-An Yu. A pso based virtual sample generation method for small sample sets: Applications to regression datasets. *Engineering Applications of Artificial Intelligence*, 59:236–243, 2017.
- [3] G. Kang, L. Wu, Y. Guan, and Z. Peng. A virtual sample generation method based on differential evolution algorithm for overall trend of small sample data: Used for lithium-ion battery capacity degradation data. *IEEE Access*, 7:123255–123267, 2019.
- [4] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.
- [5] Christoph Molnar. *Interpretable Machine Learning*, chapter 2.1. No publisher, 2019. <https://christophm.github.io/interpretable-ml-book/>.
- [6] G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995. <https://www.python.org/>.
- [7] Keiron O’Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.
- [8] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5:23, 2017.
- [9] Gustav Holmgren, Peder Andersson, Andreas Jakobsson, and Attila Frigyesi. Artificial neural networks improve and simplify intensive care mortality prognostication: a national cohort study of 217,289 first-time intensive care unit admissions. *Journal of Intensive Care*, 7(1):44, Aug 2019.

- [10] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, May 2016.
- [11] B. Zhu, Zhongsheng Chen, and Lean Yu. A novel mega-trend-diffusion for small sample. *Huagong Xuebao/Journal of Chemical Industry and Engineering (China)*, 67:820–826, 03 2016. (only in Chinese).
- [12] Der-Chiang Li, Chih-Sen Wu, Tung-I Tsai, and Yao-San Lina. Using mega-trend-diffusion and artificial samples in small data set learning for early flexible manufacturing system scheduling knowledge. *Computers & Operations Research*, 34(4):966–982, 2007.
- [13] Magnus Ekström and Sreenivasa Rao Jammalamadaka. A general measure of skewness. *Statistics & Probability Letters*, 82(8):1559–1568, 2012.
- [14] Che-Jung Chang, Der-Chiang Li, Wen-Li Dai, and Chien-Chih Chen. A latent information function to extend domain attributes to improve the accuracy of small-data-set forecasting. *Neurocomputing*, 129:343–349, 2014.
- [15] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN’95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.
- [16] Harish Garg. A hybrid pso-ga algorithm for constrained optimization problems. *Applied Mathematics and Computation*, 274:292–305, 2016.
- [17] R. Brits, A.P. Engelbrecht, and F. van den Bergh. Locating multiple optima using particle swarm optimization. *Applied Mathematics and Computation*, 189(2):1859–1883, 2007.
- [18] Hui Pan, Ling Wang, and Bo Liu. Particle swarm optimization for function optimization in noisy environment. *Applied Mathematics and Computation*, 181(2):908–919, 2006.
- [19] R.E. Perez and K. Behdinan. Particle swarm approach for structural design optimization. *Computers & Structures*, 85(19):1579–1588, 2007.
- [20] Federico Marini and Beata Walczak. Particle swarm optimization (pso). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015.
- [21] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997.

- [22] Walter Zucchini. An introduction to model selection. *Journal of Mathematical Psychology*, 44(1):41–61, 2000.
- [23] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.
- [24] Gérard Biau and Erwan Scornet. A random forest guided tour. *TEST*, 25(2):197–227, Jun 2016.
- [25] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*, pages 587–603. Springer, 2009. <https://web.stanford.edu/~hastie/ElemStatLearn/>.
- [26] Yi Lin and Yongho Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 201/05/09/2006. Full publication date: Jun., 2006.
- [27] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.
- [28] Tung-I Tsai and Der-Chiang Li. Utilize bootstrap in small data set learning for pilot run modeling of manufacturing systems. *Expert Systems with Applications*, 35(3):1293–1300, 2008.
- [29] Lester James V. Miranda. PySwarms, a research-toolkit for Particle Swarm Optimization in Python. *Journal of Open Source Software*, 3, 2018.
- [30] Der-Chiang Li and I-Hsiang Wen. A genetic algorithm-based virtual sample generation technique to improve small data set learning. *Neurocomputing*, 143:222–230, 2014.
- [31] Der-Chang Li and Yao-San Lin. Using virtual sample generation to build up management knowledge in the early manufacturing stages. *European Journal of Operational Research*, 175(1):413–434, 2006.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl