

STRATEGIES FOR REDUCING THE SIZE OF
THE SEARCH SPACE IN
SEMANTIC GENETIC PROGRAMMING

LUIS FERNANDO MIRANDA

STRATEGIES FOR REDUCING THE SIZE OF
THE SEARCH SPACE IN
SEMANTIC GENETIC PROGRAMMING

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADORA: GISELE LOBO PAPPA
COORIENTADOR: LUIZ OTÁVIO VILAS BÔAS OLIVEIRA

Belo Horizonte

Março de 2018

LUIS FERNANDO MIRANDA

STRATEGIES FOR REDUCING THE SIZE OF
THE SEARCH SPACE IN
SEMANTIC GENETIC PROGRAMMING

Dissertation presented to the Graduate
Program in Computer Science of the Fed-
eral University of Minas Gerais in partial
fulfillment of the requirements for the de-
gree of Master in Computer Science.

ADVISOR: GISELE LOBO PAPP
CO-ADVISOR: LUIZ OTÁVIO VILAS BÔAS OLIVEIRA

Belo Horizonte

March 2018

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Miranda, Luis Fernando.

M672s Strategies for reducing the size of the search space in semantic genetic programming. / Luis Fernando Miranda. — Belo Horizonte, 2018.
xxiv, 85 f.: il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Gisele Lobo Pappa.

Coorientador: Luiz Otávio Vilas Bôas Oliveira.

1. Computação – Teses. 2. Programação Genética (Computação) - Teses. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*73(043)



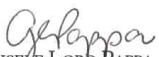
UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

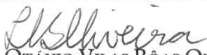
FOLHA DE APROVAÇÃO

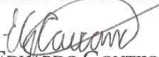
Strategies for reducing the size of the search space in semantic genetic programming

LUIS FERNANDO MIRANDA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:


PROFA. GISELE LOBO PAPP - Orientadora
Departamento de Ciência da Computação - UFMG


DR. LUIZ OTÁVIO VILAS BOAS OLIVEIRA - Coorientador
Pesquisador/Departamento de Ciência da Computação - UFMG


PROF. EDUARDO GONTIJO CARRANO
Departamento de Engenharia Elétrica - UFMG


PROF. FABRICIO MURAI FERREIRA
Departamento de Ciência da Computação - UFMG


PROF. THIAGO FERREIRA DE NORONHA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 12 de Março de 2018.

To my parents.

Agradecimentos

Primeiramente, gostaria de agradecer aos meus pais. Além de todo apoio e carinho, vocês persistiram de forma incessante em pavimentar o caminho que levou à minha formação acadêmica. Eu sei que isso me abriu portas que nunca estiveram visíveis a vocês. Jamais conseguirei retribuir tudo o que vocês fizeram por mim, mas procurarei fazê-lo eternamente. Agradeço, também, à minha família, por terem me apoiado mesmo sem entender o porquê de eu ficar tanto tempo olhando para telas escuras e palavras coloridas.

Agradeço ao meu amor, Helena, por ter escutado tantas vezes as palavras "experimento", "*script*" e "pomodoro" e, ainda assim, não ter desistido de mim. Muito obrigado por toda a compreensão e paciência e por sempre acreditar que eu conseguiria encontrar as soluções para os diversos problemas, mesmo quando eu mesmo não acreditava.

Agradeço de forma especial à minha orientadora, Gisele, pelo excelente trabalho de orientação e pelo exemplo de dedicação incansável aos trabalhos de professor e pesquisador. Muitíssimo obrigado por toda a confiança e paciência. Sua preocupação com a qualidade e a excelência dos projetos em que trabalhamos juntos são marcas na minha formação que buscarei seguir por toda a minha vida.

É difícil expressar em um parágrafo a gratidão que sinto pelo meu coorientador e amigo, Luiz Otávio. Muitíssimo obrigado pela enorme paciência com as minhas infundáveis perguntas. Sem a sua ajuda, o período do mestrado, além de caótico, não teria sido nem de longe tão enriquecedor quanto foi. Além disso, ao ser um profissional tão comprometido e uma pessoa tão organizada, você me inspirou e me ajudou a ser um estudante melhor.

Também tive a sorte de conviver com diversas outras pessoas incríveis ao longo desses dois anos. Em especial, gostaria de agradecer ao meu amigo de *flat*, João, aos amigos e companheiros de laboratório, Alex e Osvaldo e aos meus amigos, Péricles, Thiago, Abraão, Leandro, Douglas, Weverton, Daniela, Eduardo e Jéssica. Muito obrigado por terem feito esses dois anos passarem voando.

*“O correr da vida embrulha tudo;
a vida é assim: esquentada e esfria,
aperta e daí afrouxa,
sossega e depois desinquieta.
O que ela quer da gente é coragem.”*
(Guimarães Rosa)

Resumo

Ao tentar resolver problemas de otimização, algoritmos de programação genética aplicam operações bio-inspiradas (mutação e cruzamento, por exemplo) de forma a explorar o espaço de possíveis soluções em busca de uma solução satisfatória. Normalmente tais algoritmos são utilizados na resolução de um problema conhecido como regressão simbólica, onde o objetivo é encontrar uma expressão matemática cuja curva correspondente se aproxime daquela induzida a partir de um conjunto de instâncias de treinamento.

Operadores canônicos de programação genética não levam em conta aspectos semânticos, o que tende a piorar o desempenho e a robustez dos métodos que os utilizam. Operadores genéticos semânticos, por outro lado, agregam noções de semântica que permitem uma exploração mais consistente do espaço de busca. Outra melhoria parte da exploração de propriedades geométricas que descrevem a relação espacial entre possíveis soluções em um espaço semântico n -dimensional, onde n é igual ao número de instâncias de treinamento. O método normalmente tomado como referência nesse contexto se chama *Geometric Semantic Genetic Programming* (GSGP). No entanto, em problemas onde o valor de n é alto - um cenário comum em aplicações reais - o processo de busca pode se tornar excessivamente complicado, uma vez que o volume do espaço semântico cresce exponencialmente em função do número de dimensões. Esta tese busca reduzir esse problema, focando na redução da dimensionalidade do espaço de busca no contexto de programação genética semântica através de métodos de seleção de instâncias.

Nosso principal objetivo é projetar, implementar e validar métodos que reduzam o tamanho do espaço de busca. Mais precisamente, queremos entender até que ponto o número de dimensões do espaço semântico é capaz de influenciar o processo de busca e qual o impacto da aplicação de métodos de seleção sobre os resultados da busca realizada pelo GSGP. Além disso, buscamos entender o impacto provocado pelo ruído sobre os métodos de programação genética e sobre as estratégias de seleção de instância propostas.

Duas abordagens são consideradas: (i) aplicar métodos de seleção de instâncias como uma etapa de pré-processamento, antes que as instâncias de treinamento sejam fornecidas ao GSGP e (ii) incorporar a seleção de instâncias ao processo evolutivo do GSGP. Um conjunto de experimentos foi realizado em um grupo de bases de dados reais e sintéticas. A análise experimental realizada indica que parte dos métodos propostos podem, de fato, melhorar aspectos relacionados à busca realizada pelo GSGP.

Palavras-chave: Programação Genética Geométrica Semântica, Regressão Simbólica, Seleção de instâncias, Aprendizagem Supervisionada.

Abstract

When trying to solve optimization problems, genetic programming (GP) algorithms apply bio-inspired operations (e.g., crossover and mutation) in order to find a satisfactory solution in the space of possible solutions. Usually, such algorithms are used to solve a problem known as symbolic regression, in which the goal is to find a mathematical expression whose corresponding curve approximates that one induced by a set of training instances.

Canonical GP operators do not take into account semantic aspects, which tends to worsen the performance and robustness of the methods that use them. Semantic genetic operators, on the other hand, aggregate notions of semantics that allow a more consistent exploration of the search space. Another improvement is the exploitation of geometric properties that describe the spatial relationship between possible solutions in an n -dimensional semantic space, in which n is equal to the number of training instances. The method usually taken as a reference in this context is called Geometric Semantic Genetic Programming (GSGP). However, in problems where the value of n is high—a common scenario in real applications—the search process can become excessively complicated, since the size of the semantic space increases exponentially as the number of dimensions increases. In this thesis, we aim to mitigate this problem by focusing on the reduction of the dimensionality of the search space in the context of semantic genetic programming through instance selection methods.

Our main goal is to design, implement and validate methods that reduce the size of the search space. More precisely, we want to understand to what extent the number of semantic space dimensions is capable of influencing the search process and the impact of the application of selection methods on the search performed by GSGP. In addition, we attempt to understand the impact of noise on genetic programming methods and on the proposed instance selection strategies.

Two approaches are considered: (i) applying instance selection methods as a pre-processing step, i.e, before the training instances are provided to the GSGP, and (ii) incorporating the instance selection into the evolutionary process of GSGP. Experi-

ments were performed on a set of real-world and synthetic datasets. The experimental analysis indicates that some of the proposed methods may, in fact, improve aspects related to the search performed by GSGP.

Keywords: Geometric Semantic Genetic Programming, Symbolic Regression, Instance Selection, Supervised Learning.

List of Figures

1.1	Example of a GP syntax tree	2
1.2	Example of fitness evaluation in the semantic space.	3
1.3	Possible effect of instance selection methods in the fitness calculation. . . .	6
2.1	Main steps of an evolutionary algorithm.	10
2.2	Example of fitness landscape.	12
2.3	Example of application of the crossover operator.	13
2.4	Example of application of the mutation operator.	14
2.5	Illustration of the semantic space and the geometric crossover and mutation operators	16
4.1	Median training and test NRSME obtained by GP and GSGP for each dataset.	33
4.2	Median test RIE and EIE obtained by GP and GSGP for each dataset. . .	34
5.1	Example of an unbalanced dataset and the impact of a instance selection method on the regression.	41
5.2	Comparison between the proximity and the surrounding functions.	42
5.3	Illustration of the ranking process applied to the <i>kotanche</i> dataset.	43
5.4	Relative weights obtained by applying the different metrics to a training set.	43
5.5	Projections and embeddings created by applying the dimensionality reduction methods	45
5.6	Results of the instance selection process applied to the <i>kotanche</i> dataset.	47
5.7	Probability of selecting an instance according to its normalized rank. . . .	49
5.8	Evolution of the RMSE values obtained by GP according to the number of instances removed.	56
5.9	Initial weight calculation for one of the instances of the <i>keijzer-6</i> dataset .	57
5.10	Variation of the median execution time for GP runs according to the number of instances removed.	58

5.11	Evolution of the RMSE values obtained by GSGP according to the number of instances removed.	61
5.12	Variation of the median execution time for GSGP runs according to the number of instances removed.	62
5.13	Evolution of the RMSE values obtained by GSGP combined with an embedding method, according to the number of instances removed.	65
5.14	Median RMSE in the training and test sets over the generations for GSGP with and without PSE for <i>yacht</i> and <i>towerData</i> datasets.	66
5.15	Median training and test RMSE obtained by GSGP and PSE for each dataset.	68

List of Tables

4.1	Datasets used in the experiments regarding noise impacts.	29
5.1	Datasets used in the experiments.	50
5.2	Median training and test RMSE and reduction (<i>% red.</i>) achieved by the algorithms for each dataset.	51
5.3	Test RMSE obtained by GP on a training set reduced using the proximity function.	53
5.4	Test RMSE obtained by GP on a training set reduced using the surrounding function.	54
5.5	Test RMSE obtained by GP on a training set reduced using the remoteness function.	54
5.6	Test RMSE obtained by GP on a training set reduced using the nonlinearity function.	55
5.7	Test RMSE obtained by GSGP on a training set reduced using the proximity function.	58
5.8	Test RMSE obtained by GSGP on a training set reduced using the surrounding function.	59
5.9	Test RMSE obtained by GSGP on a training set reduced using the remoteness function.	59
5.10	Test RMSE obtained by GSGP on a training set reduced using the nonlinearity function.	60
5.11	Test RMSE obtained by GSGP on a training set embedded using the <i>t-SNE</i> method and reduced using the proximity function.	62
5.12	Test RMSE obtained by GSGP on a training set embedded using the <i>t-SNE</i> method and reduced using the surrounding function.	63
5.13	Test RMSE obtained by GSGP on a training set embedded using the <i>t-SNE</i> method and reduced using the remoteness function.	63

5.14	Test RMSE obtained by GSGP on a training set embedded using the <i>t-SNE</i> method and reduced using the nonlinearity function.	64
5.15	Median training RMSE of the PSE with different values of λ and ρ for the adopted test bed.	64
5.16	Median training and test RMSE's obtained for each dataset.	66
5.17	Median training and test RMSE's obtained by GSGP and PSE.	67
A.1	Training RMSE obtained by GP on a training set reduced using the proximity function.	71
A.2	Training RMSE obtained by GP on a training set reduced using the surrounding function.	72
A.3	Training RMSE obtained by GP on a training set reduced using the remoteness function.	72
A.4	Training RMSE obtained by GP on a training set reduced using the nonlinearity function.	73
A.5	Training RMSE obtained by GSGP on a training set reduced using the proximity function.	73
A.6	Training RMSE obtained by GSGP on a training set reduced using the surrounding function.	74
A.7	Training RMSE obtained by GSGP on a training set reduced using the remoteness function.	74
A.8	Training RMSE obtained by GSGP on a training set reduced using the nonlinearity function.	75
A.9	Training RMSE obtained by GSGP on a training set embedded using the <i>t-SNE</i> method and reduced using the proximity function.	75
A.10	Training RMSE obtained by GSGP on a training set embedded using the <i>t-SNE</i> method and reduced using the surrounding function.	76
A.11	Training RMSE obtained by GSGP on a training set embedded using the <i>t-SNE</i> method and reduced using the remoteness function.	76

Contents

Agradecimientos	xi
Resumo	xv
Abstract	xvii
List of Figures	xix
List of Tables	xxi
1 Introduction	1
1.1 Motivation	4
1.2 Objectives	6
1.3 Contributions	7
1.4 Thesis Organization	7
2 Concepts and Problem Definition	9
2.1 Genetic Programming	9
2.1.1 Representation and initialization	10
2.1.2 Individual Evaluation and Selection	11
2.1.3 Genetic Operators	13
2.2 Semantic GP	14
2.2.1 GSGP	15
2.3 Instance Selection	17
3 Related Work	21
3.1 Noise Impact	21
3.1.1 Genetic Programming with Noisy Data	21
3.1.2 Quantifying Noise Robustness	23
3.2 Instance Selection	24

4	Noise Impact	27
4.1	Methodology	28
4.1.1	Test Bed	29
4.1.2	Noise Robustness in Regression	30
4.1.3	Experimental Analysis	31
5	Instance Selection for Regression	37
5.1	Pre-Processing Strategies	38
5.1.1	TENN and TCNN	38
5.1.2	Instance Weighting	40
5.2	PSE	48
5.3	Experimental Analysis	49
5.3.1	Experimental Design	50
5.3.2	TCNN and TENN	51
5.3.3	Instance Weighting	52
5.3.4	PSE	62
5.4	Noise Effects on PSE	65
5.4.1	Experimental Analysis	65
6	Conclusions and Future Work	69
A	Training Results for Instance Weighting	71
	Bibliography	77

Chapter 1

Introduction

Evolutionary computation is an umbrella term used to refer to a group of problem-solving techniques which uses computational models inspired by the well-known mechanisms of evolution [Fogel, 2006]. These techniques share a common conceptual base of simulating the evolution of individual structures via processes that mimic biological operations and depend on the perceived performance (fitness) of these structures, defined by the environment that surrounds them.

Evolutionary algorithms maintain a population of structures—representing candidate solutions for the problem being solved—that evolves according to rules of selection and other operations, such as recombination (also called crossover) and mutation. A fitness value is attributed to each individual in the population, reflecting its behavior in the environment. The selection process comprises a probabilistic procedure capable of guiding the search towards regions with high fitness individuals, thus exploiting the available information about the search space. Recombination and mutation perturb these individuals, providing general heuristics for exploration. Although simplistic from a biologist’s viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms [Spears et al., 1993].

In this thesis, we focus on a branch of evolutionary computation called Genetic Programming (GP), which distinguishes from other evolutionary algorithms by representing each individual as an interpretable program, variable in length and usually expressed as a syntax tree (as shown in Figure 1.1), which must be executed in order to evaluate its fitness. During the evolutionary process, GP evolves a population of programs by stochastically transforming, generation by generation, this population into a new, hopefully better, population of programs [Banzhaf et al., 1998; Poli et al., 2008].

GP has been successfully used to solve a supervised learning task called symbolic regression, in which we want to find a mathematical expression, in symbolic form, that

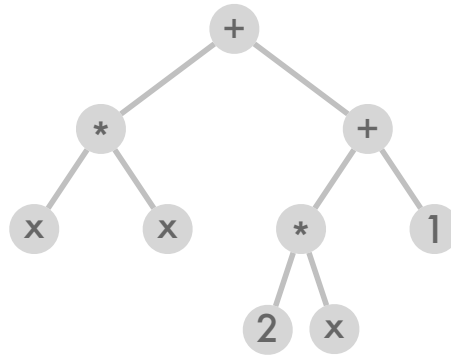


Figure 1.1: GP syntax tree representing the function $x^2 + 2x + 1$

fits (or approximately fits) a collection of instances in a training set. We can measure the quality of a produced model by using it to predict the output value of a set of test instances (for which we know the actual output value). The smallest the average distance between the predicted and the actual output values, the better the model. Once a satisfactory model is found, we can use it to interpolate the output value for unknown instances, which in turn allows us to make predictions, gain insights, and find optimal regions in the underlying response surface.

GP traditionally evolves a population of syntax trees by using two different genetic operators, called crossover and mutation, which in their original form produce offspring by manipulating the syntax, i.e. the structure, of the parents. Both operators ensure that the resulting offspring will represent valid individuals, but they are unable to guarantee that the behavior of the offspring will somehow resemble the behavior of their parents. This fact—that canonical GP operators act on a purely syntactic level—implies that, quite often, individuals on the offspring will represent worse candidate solutions, deteriorating the search process.

With that in mind, researchers focused on the conception of GP methods that take the behavior—or the semantics—of the individuals into account. These new methods incorporate semantic awareness to the evolutionary process, modifying the search operators so that they produce offspring that behave similarly to their parents [Vanneschi et al., 2014a], with promising results obtained so far.

In the supervised learning context, the semantics of a program is described as the vector of outputs the program generates when applied to the set of inputs defined by the set of training instances. This vector can be represented as a point in an n -dimensional metric space $\mathcal{S}$ (called semantic space), where n is the size of the training set. Figure 1.2 illustrates the evaluation of the semantics of a GP individual and its representation in the semantic space. The target output—i.e., the vector of actual

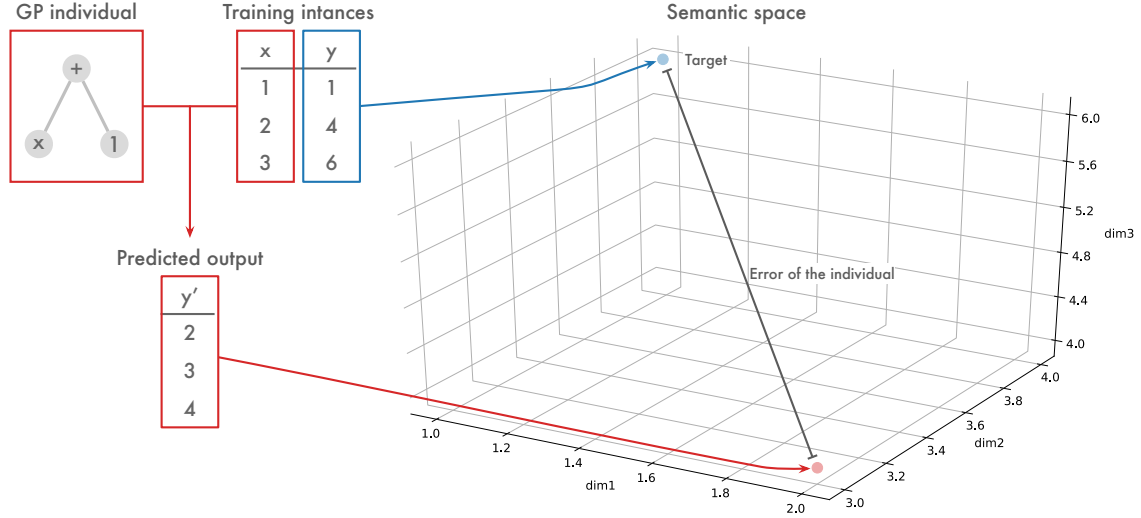


Figure 1.2: Fitness evaluation and semantic space. During the fitness evaluation, GP methods apply the function represented by the individual to the set of inputs x , generating an output vector y' , which can be represented in the semantic space. The target output vector y is also representable in the semantic space and the fitness of the individual is proportional to the distance between y and y' .

values being searched, defined by the outputs given in the training set—can also be represented in the semantic space. The fitness value is proportional to the distance between the target output vector and the output vector generated after applying the function represented by the individual to the set of inputs. In this context, the objective in a semantic GP search can be seen as an attempt to modify the semantics of the individuals in $\mathcal{S}$ by moving them closer to the target output.

Despite the new perspective and superior performance brought by the semantic GP methods, they are indirect: their operators act on the syntax of parents to produce offspring, which are accepted if some semantic criterion is satisfied. [Moraglio et al. \[2012\]](#) describe two drawbacks related to this approach: (i) it is very wasteful as it is heavily based on trial-and-error and (ii) it does not provide insights on how syntactic and semantic searches relate to each other. They also present a new GP framework, capable of manipulating the syntax of individuals with geometric implications on their disposition in the semantic space. This framework, called Geometric Semantic GP (GSGP), replaces the traditional crossover and mutation operators by the so-called geometric semantic operators, which exploit semantic awareness and geometric shapes to describe the spatial relationship between parents and offspring, inducing precise geometric properties into the semantic space. The geometric semantic crossover operator based on Euclidean distances, for example, returns a convex combination of the

parents, generating offspring with output vectors in the segment between the output vectors of the parents. These operators search directly the space of the underlying semantics of the programs, inducing a unimodal fitness landscape—which can be optimized by evolutionary algorithms with good results for virtually any metric [Moraglio, 2011].

By construction, the geometric semantic operators are affected by the increase of the dimensionality of the semantic space. This effect is related to the curse of dimensionality accounted by the exponential increase of the search space with the number of dimensions of $\mathcal{S}$, which depends on the number of training instances. In problems where this number is high—a common scenario in real-world applications—we end up with an extremely large search space, more difficult to search into and that may bring different instances representing the same type of information. In this scenario, by reducing the number of instances we automatically reduce the number of dimensions of the semantic space, which in turn reduces its complexity. Intuitively, with a lower complexity, the number of possible combinations will decrease, which may increase the speed of convergence to the optimum. New strategies to address this problem are the main goal of this thesis.

Apart from all the advantages of using GSGP instead of GP, a few papers in the literature have also claimed GSGP to be more robust to overfitting—which can be caused by noisy data points—when compared to canonical GP techniques [Vanneschi et al., 2013; Castelli et al., 2012, 2013a; Vanneschi et al., 2014b; Vanneschi, 2014]. At first, this might be even counter-intuitive, as the exponential growth of solutions caused by GSGP might even worsen the effects of overfitting. However, no systematic study has been performed to assess whether and/or in which situations this might be true, and how noisy data points affect the performance of GSGP. With that in mind, before diving into issues related specifically to the dimensionality of the semantic space, we study the impact of noise on the regression performed by GSGP.

1.1 Motivation

The number of massive datasets has been growing remarkably in the last few decades, covering a large group of fields, such as e-commerce, financial market, image processing and bioinformatics. The availability of very large databases has constantly challenged the machine learning and data mining community to come up with fast, scalable and accurate approaches [Peteiro-Barral and Guijarro-Berdinas, 2013].

In the symbolic regression context, making massive datasets smaller by removing

part of their instances can reduce the computational effort employed to induce the regression model. In addition, instances may concentrate in some particular regions of the input space, leading the regression method to overspecialize the model to map these regions, while areas with fewer instances are not given the necessary attention. In these cases, removing instances from these dense regions can improve the induced model by leading the regression method to consider the whole input space with similar importance [Cardie and Howe, 1997]. An intuitive strategy to reduce the size of the training set and the dimensionality of the semantic space is by performing what in the machine learning literature is known as data instance selection [Domingos, 2012].

In the GSGP context, as we mentioned, by reducing the size of the training set, the number of dimensions of the semantic space also decreases, leading to a smaller and simpler search space. Figure 1.3 illustrates how the number of dimensions can modify one of the aspects of the search. It illustrates a 2D semantic space (i.e., only two examples are present in the training set) with two candidate solutions to the problem (parents p_1 and p_2). The target output and the set of positions the offspring generated by the geometric crossover between p_1 and p_2 (blue dashed line) can occupy is also depicted, with o_1 representing a specific offspring in that set. Figure 1.3b shows the fitness function for the possible offspring resulting from the crossover (in this case, since the fitness value represents the error between the known and the predicted outputs, lower values indicate better solutions). The green point represents the fitness value for o_1 , which in this scenario has the best fitness among the possible offspring. As we said, each instance in the training set corresponds to one dimension in the semantic space. Now consider, for example, that dim_1 was induced by an instance which is actually an outlier. If we remove that instance, we could see the semantic space in a simplified way, in which only the dim_2 axis matters. In this case, the fitness function will change and become the one represented in Figure 1.3c, and the best possible fitness value will no longer be equal to o_1 's fitness.

The fact that instance selection methods can change the search process does not necessarily mean that the solutions found will be improved or that the convergence speed will be increased. We made the conjecture that these improvements may happen based on the assumption that, on semantic spaces with a lower number of dimensions, if these dimensions are induced by relevant instances, methods that rely on geometric properties and operations, like GSGP, can take advantage of working in a simplified search space. However, there are some possible drawbacks as well. For example, if the instance selection method removes instances that are actually crucial for the search process, an important source of information will be lost and the solution achieved will certainly be flawed. With this scenario in mind, our main research hypothesis is:

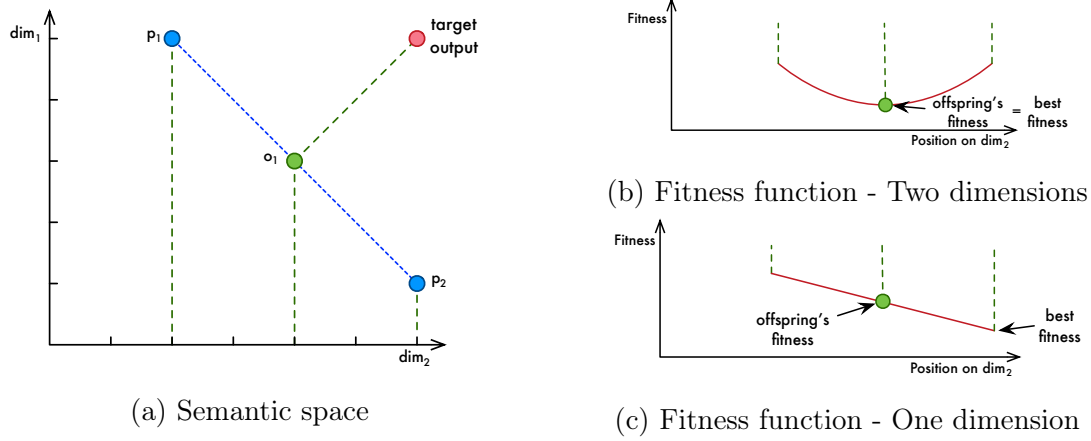


Figure 1.3: Possible effect of instance selection methods in the fitness calculation.

By decreasing the number of training cases, and consequently the number of dimensions of the semantic space, we can improve the search process performed by GSGP, making it simpler and more efficient.

1.2 Objectives

In order to validate our hypothesis, this thesis aim attention on the following research questions:

1. What is the impact of noisy data on the performance of GSGP when compared to GP in symbolic regression problems?
2. To what extent can the number of dimensions reshape the semantic space and help the search process?
3. What is the impact of instance selection methods to the results of the search performed by GSGP?
4. What is the impact of noisy data on the performance of the instance selection methods proposed?

In order to answer Q1, we made a deep analysis of the GSGP performance in the presence of noise. Q2 was tackled by analyzing the impact of instance selection methods on both GP and GSGP. In this way, we were able to verify whether the improvements on the search could be linked to a smaller dimensionality of the semantic space. The answer to Q3 was obtained by analyzing how the predictive capabilities of GSGP are affected by the instance selection performed as a preprocessing step and as an integrated

approach to the GSGP search. Finally, Q4 looked at which instance selection method performed better under the presence of noise.

1.3 Contributions

A study regarding the effects of noisy data on the GSGP search: we performed an analytic study of the impact of noisy instances on the performance of GSGP when compared to GP in symbolic regression problems. Using 15 synthetic datasets, we added different ratios of noise and compared the results obtained with those achieved by a canonical GP. The performance of both methods was measured using a conventional error metric and new robustness metrics adapted from the classification literature. The results of this study were published in [Miranda et al. \[2017\]](#).

A study regarding existing instance selection methods and the introduction of new strategies: we also presented a study about the impact of instance selection methods on GSGP by analyzing and extending existing instance selection approaches, all of which implemented as pre-processing steps. We validate our study by performing an experimental analysis using a diversified collection of real-world and synthetic datasets. We also propose a new method, called Probabilistic Instance Selection Based on the Error, which allows the reduction of the semantic space by selecting instances during the evolutionary process, taking into account the impact of each instance on the search. The initial results of this study were published in [Oliveira et al. \[2016\]](#), and a more complete version submitted to the *Evolutionary Computation Journal*.

1.4 Thesis Organization

The remainder of this document is organized as follows. Chapter 2 describes the main concepts concerning semantic genetic programming and details the problems we are trying to solve. Chapter 3 addresses related work, while 4 presents an analysis regarding the effects of noisy data on GP and GSGP. Chapter 5 describes and evaluates our strategies for reducing the size of the search space in semantic GP. Finally, Chapter 6 concludes the text and addresses future work.

Chapter 2

Concepts and Problem Definition

This chapter addresses, in Section 2.1, the main concepts regarding evolutionary algorithms, together with the specificities behind genetic programming. It also describes, in Section 2.2, fundamental elements related to semantic and geometric semantic genetic programming. In Section 2.3, it discusses key aspects involving instance selection tasks. Along the chapter, we introduce evaluation metrics and detail the problems we are trying to solve.

2.1 Genetic Programming

The three main mechanisms that drive evolution forward are reproduction, mutation, and natural selection (i.e., the Darwinian principle of survival of the fittest) [Raidl, 2005]. Evolutionary algorithms adopt these mechanisms of natural evolution in a simplified way in order to improve, generation by generation, the quality—or fitness—of a population of individuals representing potential solutions to a given optimization problem [Back and Schwefel, 1996].

In genetic programming, individuals in the population are interpretable programs, typically represented as syntax trees. This population is evolved by repeatedly selecting the fittest programs and producing new programs from them [Langdon, 1996]. Furthermore, the population is usually of fixed size and each new program replaces an existing member. The fitness value of each individual is calculated by running it on the input attributes of a set of training instances and verifying how close the outputs produced by them are compared to the actual output values of these instances.

Figure 2.1 illustrates traditional steps of a GP algorithm. It starts by randomly creating an initial population of candidate solutions. Then, at each generation, it evaluates the population members based on a fitness function, assigning them a fitness

value. Some individuals are selected based on a probability proportional to their fitness and submitted to genetic operators. The resulting individuals replace the current population and the generation finishes. The algorithm verifies if any of the stopping criteria is satisfied (usually by reaching a maximum number of generations or finding a satisfactory solution) and, in affirmative case, it stops its execution and returns the best individual found; otherwise, it starts a new generation.

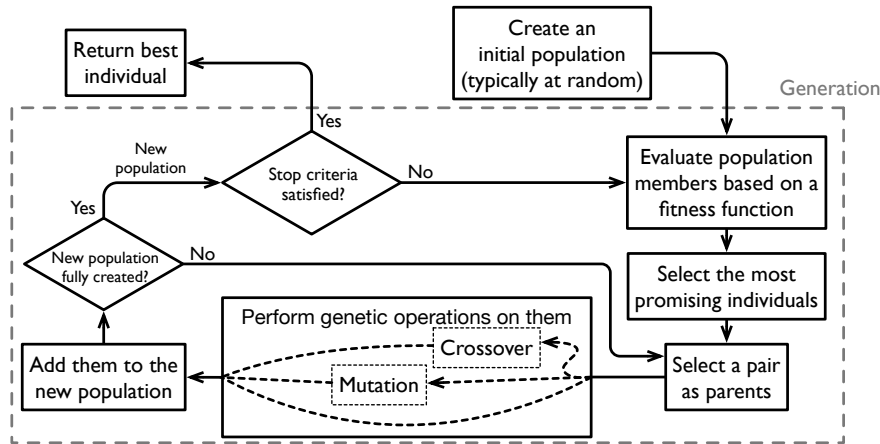


Figure 2.1: Main steps of an evolutionary algorithm.

Genetic programming has been successfully applied to a large number of problems such as automatic design [Nguyen et al., 2014], pattern recognition [Liu et al., 2016], robotic control [Busch et al., 2002], synthesis of artificial neural networks [Ritchie et al., 2003], bioinformatics [Langdon, 2015], music [Kunimatsu et al., 2015] and picture generation [Alsing, 2008]. We use GP in a particular type of supervised learning task called symbolic regression, which involves finding a mathematical expression, in symbolic form, that fits (or approximately fits) a set of training instances. Unlike traditional linear and polynomial regression methods, which fit parameters to an equation of a given form, symbolic regression searches both the parameters and the form of the equation simultaneously [Koza, 1992].

2.1.1 Representation and initialization

The choice of a structure for representing individuals in GP affects execution order, use and locality of memory, and the application of the genetic operators [Banzhaf et al., 1998]. The most common forms of representation are by linear, tree, and graph structures. The way these structures are actually held in memory, however, may differ from the virtual representation in which they are executed and modified, as an effort to improve some performance-related aspects of the algorithm.

From the three fundamental structures, syntax trees are the most common form of representing candidate solutions in GP. The trees evolved are composed of elements from terminal and function sets. Figure 1.1 shows, as an example, a syntax tree representing the function $x^2 + 2x + 1$. The terminal set, consisting of the variables and constants in the program (x , 1 and 2), are the leaves of the tree, while the arithmetic operators ($+$ and $*$) are internal nodes and form the function set.

Similarly to other evolutionary algorithms, in GP the individuals of the initial population are typically randomly generated. There are a number of different approaches to generate this random initial population, but the most used and known were proposed by Koza [1992] and are called *grow*, *full* and *ramped half-and-half*.

The three methods generate trees in a top-down fashion, by selecting one node at time. The *full* method chooses only functions until a node is at the maximum predefined tree depth. Then it chooses only terminals. The result is that every branch of the tree goes to the full maximum depth. In turn, the *grow* method involves growing trees that are variably shaped, with nodes being selected randomly from the function and the terminal sets throughout the entire tree (with exception of the root node, which is always a function). Once a branch contains a terminal node, that branch is ended, even if the maximum depth has not been reached.

The *ramped half-and-half* method incorporates both *full* and *grow* methods and involves creating an equal number of trees using a depth parameter that ranges between 2 and the maximum specified depth. For example, if the maximum specified depth is 6, 20% of the trees will have depth 2, 20% will have depth 3, and so forth up to depth 6. Then, for each value of depth, half of the trees are created via the *full* method and half are produced via the *grow* method.

2.1.2 Individual Evaluation and Selection

The process of deciding which individuals will be selected to undergo genetic operations requires assigning a fitness value to every new individual. The better the solution represented by an individual, the more likely it will survive to compose the next generation. In the symbolic regression context, we are interested in the output produced by each individual, i.e., the value returned when we evaluate its syntax tree starting at the root node. In this case, the fitness function can be seen as a metric that captures the divergence (error) between the program output and some known desired output, typically given by the set of output values of the training instances [Krawiec and Pawlak, 2012].

The representation of a solution, known as genotype, is logically separated from

the use or effects of its application to the problem, known as phenotype. In this sense, the genotype encodes the corresponding phenotype, like in nature, where the DNA encodes an actual look and operation of a human [Pawlak, 2015]. Genetic Programming, like all algorithms which depend on some form of evolutionary adaptation, operates within the context of a fitness landscape, which refers to the mapping from the genotypes of a population of individuals to their fitness, and a visualization of that mapping [Kinnear, 1994]. In its simplest form, a fitness landscape, illustrated in Figure 2.2, can be seen as a plot where each point in the horizontal direction represents the genotype of a specific individual, with its fitness plotted as the height. If the genotypes can be visualized in two dimensions, the plot can be seen as a three-dimensional map, which may contain hills and valleys, with the summit corresponding to the fitness value of the optimal solution.

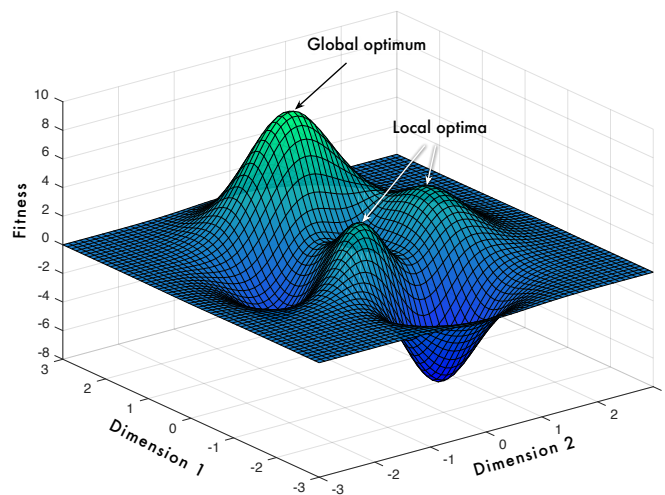


Figure 2.2: Example of fitness landscape.

There are a number of different methods that can be used for deciding which individuals will reproduce and which will be removed from the population. The most commonly employed method for selecting individuals is called tournament selection [Banzhaf et al., 1998]. In essence, the method selects randomly, with uniform probability, a group of k individuals from the current population. The best individual inside this group is selected as a parent (or as one of the parents) for the next genetic operation. The parameter k is called the tournament size and can be used to modify the selection pressure exerted by the method (the higher k the higher the pressure to select above average quality individuals, which typically implies higher convergence speed) [Poli, 2005].

2.1.3 Genetic Operators

The fact that, in GP, the initialization process produces randomly generated individuals typically implies that the average fitness of the initial population is very low. Therefore, GP methods must rely on search operators (also called genetic operators, in this context) in order to expand the search towards high fitness regions of the fitness landscape [Oliveira, 2016]. GP differs considerably from other evolutionary algorithms in the implementation of these operators [Banzhaf et al., 1998]. While there are many of them, usually only three, namely crossover, mutation, and reproduction, are adopted.

In crossover, randomly selected subtrees from each of the two parents are exchanged to form two new individuals (offsprings), as shown in Figure 2.3. The idea is that useful building blocks for the solution of a problem are accumulated in the population and that crossover permits the aggregation of them into even better solutions [Koza, 1992]. The mutation operator, on the other hand, creates only one offspring by picking a random subtree of a parent and replacing it with a new randomly generated subtree, as shown in Figure 2.4. The idea is to bring innovation to GP by introducing new code fragments into the population. The mutation operator, therefore, is used as a workaround for lost of diversity and stagnation, especially in small populations [Pawlak, 2015].

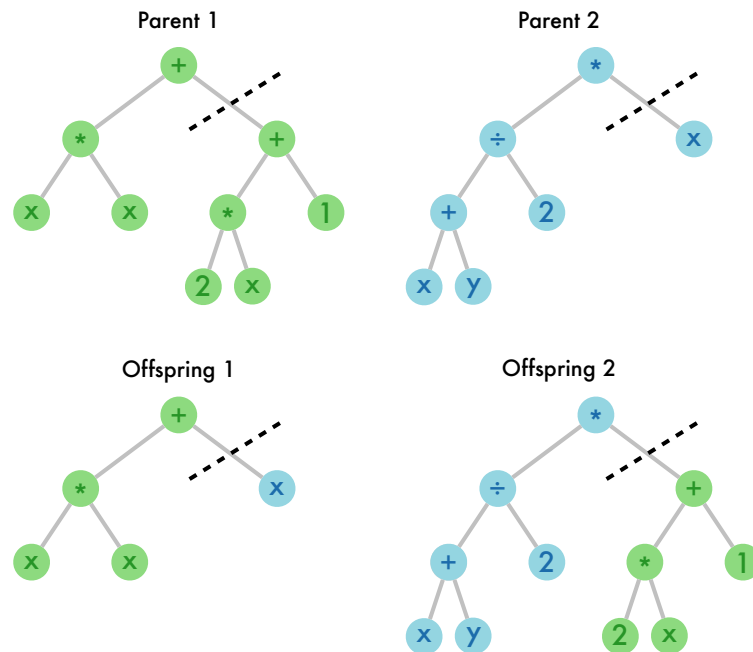


Figure 2.3: Example of application of the crossover operator. The dashed lines indicate the points where the subtrees are swapped.

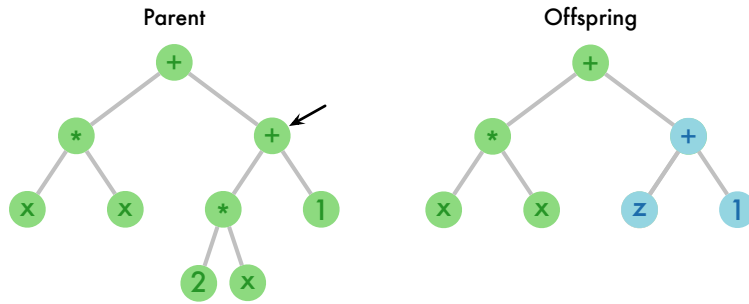


Figure 2.4: Example of application of the mutation operator. The arrow points to the root of the subtree selected for replacement.

The third genetic operator commonly applied in GP is called reproduction. Unlike the other operators, it does not perform any modifications on the selected parents, meaning that it simply copies the parent to the next population with no change.

The probability of applying each of the genetic operators is usually defined via user-defined parameters. While crossover is typically applied with high probability (between 90% and 95%), mutation is usually not applied or applied with very low probability (usually smaller than 5%). The remainder probability, i.e., to complete 100%, corresponds to the probability of applying the reproduction operator [Poli et al., 2008].

2.2 Semantic GP

Semantic genetic programming (SGP) is a relatively new thread in GP research, which originated in the high complexity of the genotype-phenotype mapping in evolutionary program synthesis [Krawiec, 2016]. In its original definition, GP manipulates the population only at a purely syntactic level, abstracting from the semantics (i.e. the behavior) of each individual. This aspect allows it to rely on simple, generic search operators, but the main consequence of this choice is that it is difficult (or even impossible) to predict how modifications in the programs will affect their semantics [Vanneschi et al., 2014a]. As a result, in GP, even minor modifications in the structure of the individuals may result in fundamentally different behavior and, as a consequence, canonical genetic operators are unable to guarantee that the offspring generated by them will share some of the semantic characteristics of their parents.

Recent works in the GP field show that the semantics of the programs can play a crucial role during the evolutionary process [Vanneschi et al., 2014a]. For this reason, researchers have been proposing a variety of methods that employ semantically-aware

operators capable of guiding the search towards more promising regions of the search space, thus improving the chances of reaching better solutions.

There are different definitions of semantics in the GP literature—e.g., Reduced Ordered Binary Decision Diagrams (BDD) [Beadle and Johnson, 2008] and logical formalism [Johnson, 2007]. We adopt a definition of semantics directly related to symbolic regression. Given a training set $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ —where $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$ ($i = 1, 2, \dots, n$)—the semantics of an individual representing a program p , denoted by $s(p)$, is defined as the vector of outputs it produces when applied to the set of inputs defined by T , i.e., $s(p) = [p(\mathbf{x}_1), p(\mathbf{x}_2), \dots, p(\mathbf{x}_n)]^T$.

Using this definition, the semantics of an individual may be seen as a point in a n -dimensional semantic space, where n is the number of training instances (previously shown in Figure 1.2). One of the advantages of this framing is that determining the semantics of an individual comes essentially for free, since each tree has to be evaluated on the training instances to calculate its fitness. Calculating the semantics of a program is then a side-effect of fitness calculation, available at no extra computational cost [Krawiec and Pawlak, 2013]. More importantly, however, such understanding of semantics binds it closely to the fitness function, capturing the divergence between the output of the individual and the desired output.

2.2.1 GSGP

A new perspective was brought by semantic GP methods, leading researches to realize that there are deeper implications of posing a program synthesis task as a search for a program with a certain semantics, rather than for a program with a certain fitness function. The fitness function typically used in GP can be seen as a metric in $\mathcal{S}$, making it possible to formally turn the set $\mathcal{S}$ into a space with certain geometry that can be exploited for the sake of search [Krawiec, 2016].

With $\mathcal{S}$ being a metric space, the fitness of an individual can be determined by measuring the distance between its representation in $\mathcal{S}$ and the target semantics (t), specified by the actual output values of the training instances. This implies that, for the Euclidean distance, for example, the surface of an evaluation function plotted with respect to $\mathcal{S}$ has the form of a cone with the apex corresponding to t , as shown in Figure 2.5a.

Following the concept of semantic space, Moraglio et al. [2012] present a new GP framework, capable of manipulating the syntax of the individuals with geometric implications on their disposition in the semantic space. The framework, called Geometric Semantic GP (GSGP), searches directly in the space of the underlying semantics of the

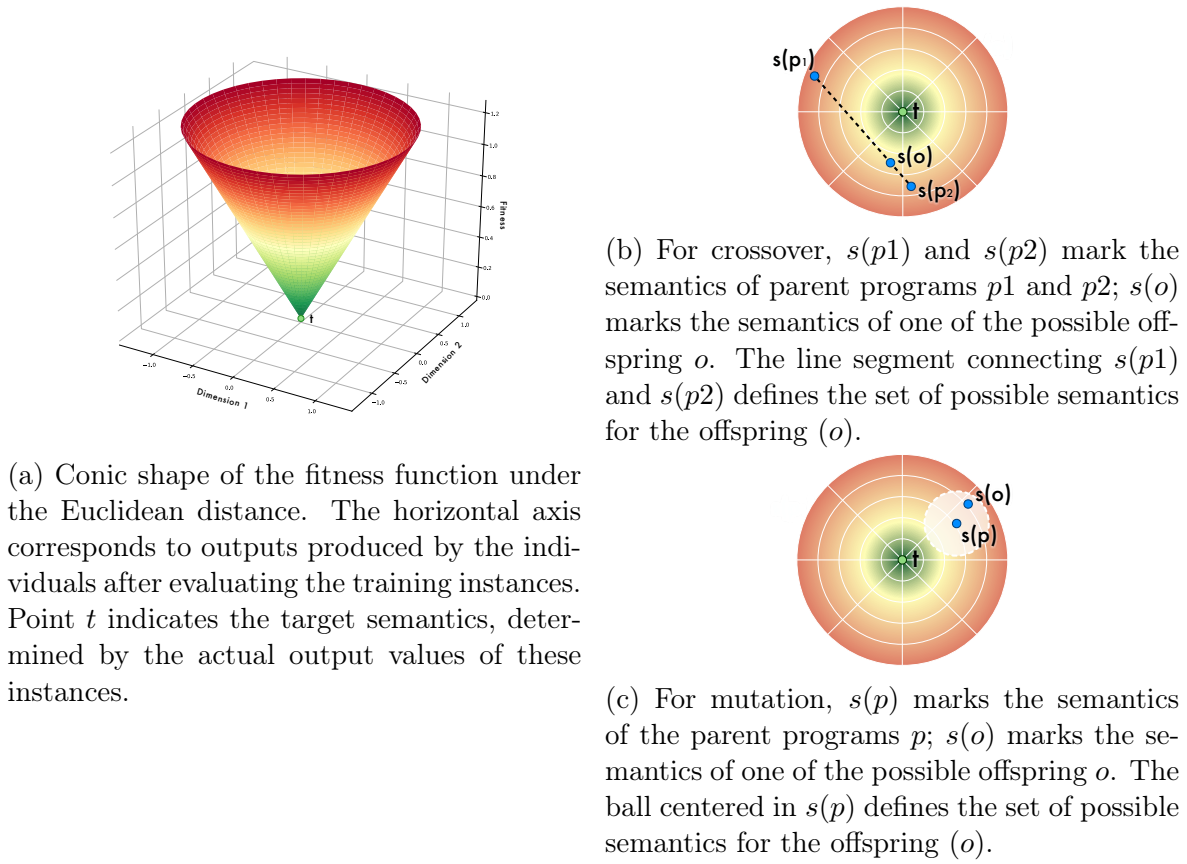


Figure 2.5: Illustration of the geometric crossover and mutation operators, for the Euclidean metric, in a two-dimensional semantic space (i.e. with only two instances present in the training set).

programs, inducing a unimodal fitness landscape as shown in Figure 2.5a. [Moraglio \[2011\]](#) presents formal evidence that evolutionary algorithms with geometric operators can optimize cone landscapes with good results for virtually any metric. In practice, GSGP introduces a new class of genetic operators which, acting on the syntax of the parent programs, produces offspring that are guaranteed to respect some semantic criterion by construction.

For Euclidean spaces, the Geometric Semantic Crossover (GSX) operator combines two parents, resulting in one offspring that behaves as a convex combination of them, i.e., for any input, the offspring is located in the metric segment between the parents. This characteristic guarantees that the offspring error—the divergence between the target output and the output generated by the individual—is lower bounded by the error of the worst of its parents. For spaces based on the Manhattan distance, the offspring resulting from the geometric semantic crossover is placed inside a hyperrectangle delimited by its parents. Figure 2.5b illustrates the representation of the GSX operator in a two-dimensional semantic space defined using the Euclidean distance.

The Geometric Semantic Mutation (GSM) operator generates offspring by applying perturbations to the parents, ensuring that the offspring is placed inside the closed ball $B(p; \varepsilon)$ centered in the parent p and with radius ε [Moraglio et al., 2012], where $\varepsilon \in \mathbb{R}$ is proportional to the mutation step parameter. Figure 2.5c shows the representation of the GSM operator in a two-dimensional semantic space, again defined using the Euclidean distance.

Since GSGP was proposed, it has been successfully applied in different domains, e.g., modelling of the behaviour of different pharmacokinetics parameters [Vanneschi et al., 2013, 2014b], prediction of high performance concrete strength [Castelli et al., 2013c], multiclass classification involving land cover/land use applications [Castelli et al., 2013b], prediction of energy performance of residential buildings [Castelli et al., 2015c], forecasting energy consumption [Castelli et al., 2015b,e], prediction of burned areas resulting from forest fires [Castelli et al., 2015d], and application in maritime awareness [Vanneschi et al., 2015].

2.3 Instance Selection

In this section, we address key aspects regarding instance selection methods. We start by discussing their benefits and drawbacks, then their divisions and basic elements. Although we focus only on the application of these methods to regression problems, the discussion presented here can also be extended to classification contexts. Likewise, most of the general ideas described in this section are not linked to a specific type of learner, meaning that, while in this thesis we put together the instance selection process with the regression performed by GSGP, they can actually be seen as independent topics.

In a nutshell, instance selection methods try to find a subset S of the original training set T such that $|S| < |T|$, and that the predictive capabilities of models induced by S are similar to those induced by T [Arnaiz-González et al., 2016].

In a way, instance selection methods can be thought of as multi-objective problems: on the one hand, they attempt to reduce the size of the resulting dataset and, on the other, to minimize some error metric [Leyva et al., 2015b].

One of the main goals of instance selection methods is to speed up the learning process. A reduction in the size of a dataset typically yields a corresponding reduction in the time required to process all training instances and induce a model. That said, the lure of instance selection methods tends to become increasingly appealing with the growing size of databases, which makes unfeasible getting results in a reasonable time

[Leyva et al., 2015a].

The selection process, however, is not always motivated by performance-related issues. In some datasets, certain regions of the input space may be excessively well-covered, while others may lack representativeness. This can bias regression models induced by learning methods to perform well only on these overrepresented regions, decreasing their generalization capability. In this scenario, removing instances from these dense regions can improve the induced model by leading the regression model to consider the entire input space with similar interest [Cardie and Howe, 1997].

In the context of this work, however, the most important benefit of instance selection methods is related to the regression performed by GSGP. As we mentioned, the semantics in GSGP is defined as a point in a space with dimensionality equivalent to the number of training instances. Therefore, by reducing the number of instances we automatically reduce the number of dimensions of the semantic space, which in turn reduces the complexity of the search space. The smaller the complexity, the smaller the number of possible combinations, which may increase the speed of convergence to the optimum. In this work, we employ two types of strategies to reduce the number of dimensions of the semantic space. The first is applied before data is given as input to GSGP, and depends only on the characteristics of the dataset. The second strategy, in turn, considers the median absolute error of each instance during the GSGP evolution to select the most appropriate instances.

In addition, instance selection methods can be used to reduce storage requirements and improve generalization and accuracy (when it is used to filter the noise out of the original dataset). However, although the expectation is to obtain an accuracy equal to or better than the original dataset, in practice this is not always achieved—i.e., if the selection process removes instances that are actually crucial for the search process, an important source of information will be wasted and a certain loss of accuracy may be inevitable [Calvo-Zaragoza et al., 2015].

Depending on how the selected subset is built, instance selection methods can be classified as incremental, decremental, or batch [Arnaiz-González et al., 2016]. Incremental methods start with an empty set and add instances to it. The order of the instances in the original set is important for these methods and will determine their effectiveness, as the current instance choice depends on instances already added to the set. An opposite approach is followed by the so-called decremental methods, which start with the original dataset and remove the instances that they consider "discardable" according to a certain criterion. Again, the order is important, but not as much as in the case of incremental methods, as the whole sample is available right from the start to help making the decisions. Batch methods, in turn, mark the instances that are

candidates to be eliminated, and once they have all been analyzed, they are removed from the dataset. This technique ensures that the impact on the complete subset after the elimination of one instance is known [Wilson and Martinez, 2000].

Another aspect that distinguishes instance reduction techniques is whether they remove internal or border instances [Wilson and Martinez, 2000]. The idea behind removing internal instances is that they do not affect the learning process as much as border instances, and can be removed with relatively little effect on the regression model produced. We use this idea in conjunction with the intuition that the learner can be more accurate if it considers the whole input space with similar importance, i.e., we seek to retain border instances, while removing internal instances of overrepresented regions of the input space. Methods that focus on eliminating border instances are often used to remove noise.

There are also methods that apply weighting functions to estimate the relative importance of each region of the input space, so that the influence of every instance can be taken into account during the selection process. Since this is a key aspect related to one of the instance selection methods presented in this thesis, we discuss it in more detail in Chapter 5.

Chapter 3

Related Work

This chapter discusses existing strategies for measuring the impact of noise on GP methods and for dealing with instance selection (IS). The first topic—noise impact on GP—is addressed by analyzing two aspects: (i) the impact of noisy data on GP—Section 3.1.1, in which we present existing strategies built in order to reduce the impact of noisy data on the GP search, and (ii) the strategies to quantify noise robustness—Section 3.1.2, in which we present a set of metrics proposed in order to estimate the loss of accuracy caused by noisy instances. Regarding the second topic, in Section 3.2 we present IS methods built to work on classification contexts and the attempts to adapt them to handle regression tasks.

3.1 Noise Impact

This section focuses specifically on works performed to analyze and minimize the effects of noisy data in GP. In addition, to the best of our knowledge, so far there are no measures to quantify the impact of noise in GP-induced models for symbolic regression problems. Thus, we also present an overview of techniques to measure the impact of noisy data on the performance of classification techniques, which we adapted to the regression domain.

3.1.1 Genetic Programming with Noisy Data

Different strategies have been proposed in symbolic regression to investigate and minimize the impact of noisy data on the search performed by GP. On the one hand, one can try to filter out noise data before performing the regression. On the other hand,

one can improve the methods to simply deal with the problem—a much more common approach.

Following the first strategy, [Sivapragasam et al. \[2007\]](#) use Singular Spectrum Analysis (SSA) to filter out the noise components before performing the symbolic regression of a short time series of fortnight river flow. The experimental study indicates that when the stochastic (noise) components are removed from short and noisy time-series, the short-lead forecasts can be improved.

Regarding methods that try to deal with the problem, [Borrelli et al. \[2006\]](#) employ a Pareto multi-objective GP for symbolic regression of time series with additive and multiplicative noise. The authors adopt two different configurations employing statistical metrics for the fitness objectives: (1) the Mean Squared Error (MSE) combined with the first two momenta and (2) the MSE with the skewness added to the kurtosis—all the measures computed regarding the desired and evaluated outputs. An experimental analysis considering time series generated from 50 functions from the literature shows that, although reducing overfitting and bloat, the multi-objective approach does not perform well when the noise level is too high. However, for moderate noise levels, the approach can successfully discover the trend of the series.

[De Falco et al. \[2007\]](#), in turn, present two GP methods guided by context-free grammars with different fitness functions that take parsimony and the simplicity of the solutions into account. The Parsimony-based Fitness Algorithm (PFA) and Solomonoff-based Fitness Algorithm (SFA) adopt fitness functions based, respectively, on parsimony ideas and on Solomonoff probability induction concepts. These methods are compared in four datasets generated from known functions, with five different levels of additive noise. The experimental analysis indicates that the SFA achieves smaller error when compared to PFA for all the datasets and levels of noise.

[Imada and Ross \[2008\]](#) also present a fitness function, alternative to functions based on the sum of errors, in which the scores are determined by the sum of the normalized differences between the target and evaluated values, regarding different statistical features. The experimental analysis of two datasets with two levels of additive noise shows that the proposed fitness function outperforms the fitness based on the sum of errors.

Although the above works handle noise in the symbolic regression context, there is a lack of studies directed to quantify the impact of the noise in GP-based regression methods. The next section presents measures adopted to quantify the influence of noise in classification algorithms from the machine learning literature. In Chapter 4 we select—and adapt—these metrics to regression problems.

3.1.2 Quantifying Noise Robustness

When a machine learning method is capable of inducing models that are not influenced by the presence of noise in data, we say it is robust to noise—i.e., the more robust a method is to noise, the more similar are the models it induces from data with and without noise [Sáez et al., 2016].

Following this premise, works in the classification literature adopt measures that compare the performance of models induced in the presence and absence of noise in the dataset, in order to evaluate the robustness of the learner. Here we introduce three of these metrics: relative risk bias, relative loss of accuracy and equalized loss of accuracy.

The Relative Risk Bias (RRB) [Kharin and Zhuk, 1994] measures the robustness of an optimal decision rule—i.e., the Bayesian Decision rule providing the minimal risk when the training data has no “contaminations”. Sáez et al. [2016] extend the measure to any classifier, given by:

$$RRB_{x\%} = \frac{R_{x\%} - R}{R} , \quad (3.1)$$

where $R_{x\%}$ is the classification error rate obtained by the classifier in a dataset with noise level given by $x\%$ and R is the classification error rate of the Bayesian Decision rule without noise (this is a theoretical decision rule, not learned from the data and depends on the data generating process), which is by definition the minimum expected error that can be achieved by any decision rule.

The Relative Loss of Accuracy (RLA) [Sáez et al., 2011], in turn, quantifies the impact of increasing levels of noise in the accuracy of the classifier model when compared to the case with no noise. The RLA measure, with level of noise equals to $x\%$, is defined by:

$$RLA_{x\%} = \frac{A_{0\%} - A_{x\%}}{A_{0\%}} , \quad (3.2)$$

where $A_{0\%}$ and $A_{x\%}$ are the accuracies of the classifier with a noise level of 0% and $x\%$, respectively. RLA is considered more intuitive than RRB, as methods obtaining high values of accuracy without noise ($A_{0\%}$) will have a low RLA value.

Finally, the Equalized Loss of Accuracy (ELA) [Sáez et al., 2016] was proposed as a correction of the RLA inspired by the measure from Kharin and Zhuk [1994], and overcomes the limitations of RRB and RLA. The initial performance ($A_{0\%}$) has a very low influence in the RLA equation, which can negatively bias the loss of accuracy of methods with high $A_{0\%}$ when compared to methods with low initial accuracy. E.g., let $A_{0\%} = A_{10\%} = 50$ be the accuracies of the method α and $A'_{0\%} = 80$ and $A'_{10\%} = 75$ be

the accuracies of the method β . Although method β has very low loss of accuracy for 10% of noise, the α classifier has a better $RLA_{10\%}$ —equals to 0. The ELA measure is given by:

$$ELA_{x\%} = \frac{100 - A_{x\%}}{A_{0\%}} \quad , \quad (3.3)$$

where $A_{x\%}$ and $A_{0\%}$ are defined as in Equation 3.2. $ELA_{x\%}$ is equivalent to $RLA_{x\%} + f(A_{0\%})$ —see Sáez et al. [2016] for the derivation—where the factor $f(A_{0\%}) = (100 - A_{0\%})/A_{0\%}$ is equivalent to $ELA_{0\%}$ and depends only on the initial accuracy $A_{0\%}$. Thus the $ELA_{x\%}$ value of a method is based on its robustness, measured by the $RLA_{x\%}$, and on the behavior of clean data—i.e., without controlled noise—measured by $ELA_{0\%}$.

3.2 Instance Selection

There is a wide variety of instance selection (IS) methods for classification tasks, as well as various surveys that present the state-of-the-art techniques [Olvera-López et al., 2010]. Similarly to regression, classification is a problem addressed by ML techniques in which the training instances are composed by an input vector and an output, but in classification the outputs are discrete variables known as classes, instead of values of a continuous variable.

IS techniques are usually used as a preprocessing stage, selecting—and sometimes even modifying—a group of instances from the training set to be used as input for a classification algorithm. Although instance selection can be used with different classification algorithms [Grochowski and Jankowski, 2004], it is usually applied to preprocess the training set used as input for the k -Nearest Neighbor (k -NN) algorithm [Cover and Hart, 1967]—a review of these works can be found in [Garcia et al., 2012; Olvera-López et al., 2010; Cano et al., 2003]. This happens because k -NN heavily relies on neighbors instances, and its computational time is closely related to the number of instances in the training set.

When compared to the variety of instance selection techniques for classification tasks, the number of IS methods for regression problems is relatively small. This difference can be explained by the increased complexity of the latter when compared to the former [Kordos and Blachnik, 2012]. As an example, the continuous nature of the outputs of instances defined in regression problems allows an infinite number of possible values predicted by the system, while in classification problems the number of possible outcomes is finite and defined by the number of classes. The dissimilarity between the two tasks also prevents directly applying instance selection methods from

the classification to the regression domain. Nevertheless, there are a few works in the literature that make some adjustments to instance selection techniques for classification problems in order to apply them to the regression domain.

The CNN for Regression (RegCNN) and ENN for Regression (RegENN) [Kordos and Blachnik, 2012] adapt the Condensed Nearest Neighbor (CNN) [Hart, 1968] and Edited Nearest Neighbor (ENN) [Wilson, 1972] methods for instance selection in classification problems to the regression domain. RegCNN and RegENN replace the label comparison used in their classification versions by an error-based comparison. Instead of comparing the label predicted by a k -NN classifier and the expected label to make a decision, RegCNN and RegENN compare the error between the output predicted by a regression method and the expected output to a threshold, in order to make the decisions of removing or keeping an instance.

The same threshold strategy is used to adapt two versions of the Incremental Reduction Optimization Procedure (DROP)—originally applied to instance selection for classification problems—to regression tasks [Arnaiz-González et al., 2016]. The authors also present DROP2 and DROP3 versions, where the number of correctly classified instances by a classification model is replaced by the sum of the absolute errors induced by a regression model.

The RegCNN and RegENN methods were renamed Threshold ENN and Threshold CNN in [Arnaiz-González et al., 2016] and compared with a discretization approach, which converts the continuous outputs of the instances into discrete values representing their labels and then applies the original ENN and CNN to select the instances. They also employ the boosting ensemble technique to combine the output of several instance selection algorithms to select the final training set.

The Class Conditional Instance Selection for Regression (CCISR) [Rodríguez-Fdez et al., 2013] extends the Class Conditional Instance Selection method from the classification domain to regression problems. CCISR employs a modified version of the class nearest neighbor relation to compute the instance scoring function used to select a subset from the training set.

The Mutual Information (MI) prototype selection [Guillen et al., 2010], on the other hand, gets inspiration from the information theory field instead of adapting IS methods from classification to regression problems. For each instance in the training set, the method computes the MI of the training set without that instance. If the MI decrease due to the deletion of an instance is not significant when compared to the MI decrease caused by the deletion of one of its neighbors, the method infers the instance is not important and removes it.

The Simple Multidimensional Iterative Technique for Subsampling (SMITS)

[[Vladislavleva et al., 2010](#)] employs one of four different metrics—proximity, surrounding, remoteness and nonlinear deviation—to measure the importance of an instance according to its nearest-in-the-input-space neighbors. These metrics are used in two different approaches: (i) to generate weights used inside the fitness function, giving different importance to each instance on the final fitness value; (ii) to select a subset from the training set, composed by the instances with the highest metric value.

Chapter 4

Noise Impact

The presence of noise in data is an issue recurrently approached in the machine learning field. Noisy data can highly influence the performance of machine learning techniques, leading to overfitting and poor data generalization [Nettleton et al., 2010]. We define noise as anything that obscures the relationship between the predictor variables and the target variable of a problem [Hickey, 1996]. In classification and regression problems, noise can be found in the input (predictor) variables, in the output (target) variable or both, and is usually the result of non-systematic errors during the process of data generation.

Over the past few years, GSGP has shown robustness and high generalization capability. Researchers believe these characteristics may be associated with a lower sensibility to noisy data. However, there is no systematic study on this matter. This chapter performs a deep analysis of the GSGP performance over the presence of noise. Using synthetic datasets where noise can be controlled, we added different ratios of noise to the data and compared the results obtained with those of a canonical GP.

In the context of regression problems, robust regression methods have been proposed to address noisy data points or outliers¹, and also to deal with other data assumptions most regression methods do not respect [Rousseeuw and Leroy, 2005], such as the independence between the input variables. Although not very popular for some time due to its computational cost, robust regression provides an alternative to deal with noise. When modeling Genetic Programming (GP) to solve symbolic regression problems, only a few studies have looked at the impact of noise on the results of data generalization and overfitting [Borrelli et al., 2006; Sivapragasam et al., 2007; De Falco

¹We consider that both noisy points and outliers are out of pattern instances that should be identified. We do not go into the merit of whether a noisy point may be actually useful to the task and represent an outlier.

et al., 2007; Imada and Ross, 2008].

Instead, the community has given great focus to the relations between complexity, overfitting and generalization, and its relation to bloat and parsimony [Fitzgerald and Ryan, 2014; Vanneschi et al., 2010]. While the former refers to a phenomenon characterized by an excess of code growth without a corresponding improvement in fitness, the latter refers to the desired property of using, within the function set, only functions necessary to solve the problem in question. These are indeed close-related issues in GP, but they do not account for problems that are not inherent to the GP search, but intrinsic to the input data. A few works have also investigated this matter considering the behavior of the GP when additive noise is added to the input data [Borrelli et al., 2006; Sivapragasam et al., 2007; De Falco et al., 2007; Imada and Ross, 2008].

As previously mentioned, in this chapter we study the impact of noisy data in GP and GSGP. The main objective is not to look at how canonical GP deals with noise, but rather investigate how GPs that take semantics into account deal with the problem when compared to GP. We start by describing, in Section 4.1.1, the test bed used in our experiments. In Section 4.1.2, we provide an overview concerning the current status of studies involving noise impact on GSGP. In Section 4.1.3, we analyze how GSGP performs in symbolic regression problems with different levels of noise when compared to GP.

We are particularly interested in noise found in the output variable of symbolic regression problems. This is because GSGP operates in a semantic space, guided by the vector of outputs defined by the training set. As a consequence, noise in the output has a much bigger impact in the search process in GSGP than noise in the predicted variables.

4.1 Methodology

This section presents the methodology followed to analyze how GSGP performs in symbolic regression problems with different levels of noise when compared to GP. We present the datasets considered in our study, along with the strategy to incrementally add noise to the data, and the measures we adopt to assess the impact of different levels of noise on the performance of GSGP and GP.

Table 4.1: Datasets used in the experiments regarding noise impacts. Training and test sets are independent. Names highlighted in bold corresponds to datasets also used in the experiments presented in 5.4.1.

Dataset	Objective function	Sampling strategy	
		Training	Test
keijzer-1	$0.3 x \sin(2\pi x)$	$E[-1, 1, 0.1]$	$E[-1, 1, 0.001]$
keijzer-2	$0.3 x \sin(2\pi x)$	$E[-2, 2, 0.1]$	$E[-2, 2, 0.001]$
keijzer-3	$0.3 x \sin(2\pi x)$	$E[-3, 3, 0.1]$	$E[-3, 3, 0.001]$
keijzer-4	$x^3 e^{-x} \cos(x) \sin(x) (\sin^2(x) \cos(x) - 1)$	$E[0, 10, 0.1]$	$E[0.05, 10.05, 0.1]$
keijzer-6	$\sum_i^x \frac{1}{i}$	$E[1, 50, 1]$	$E[1, 120, 1]$
keijzer-7	$\ln x$	$E[1, 100, 1]$	$E[1, 100, 0.1]$
keijzer-8	$\sqrt{x}$	$E[0, 100, 1]$	$E[0, 100, 0.1]$
keijzer-9	$\arcsin(x) \quad i.e., \ln(x + \sqrt{x^2 + 1})$	$E[0, 100, 1]$	$E[0, 100, 0.1]$
vladislavleva-1	$\frac{e^{-(x-1)^2}}{1.2 + (y-2.5)^2}$	$U[0.3, 4, 100]$	$E[-0.2, 4.2, 0.1]$
vladislavleva-2	$e^{-x} x^3 (\cos(x) \sin(x)) (\cos(x) \sin^2(x) - 1)$	$E[0.05, 10, 0.1]$	$E[-0.5, 10.5, 0.05]$
vladislavleva-3	$e^{-x} x^3 (\cos(x) \sin(x)) (\cos(x) \sin^2(x) - 1)(y - 5)$	$x : E[0.05, 10, 0.1]$ $y : E[0.05, 10.05, 2]$	$x : E[-0.5, 10.5, 0.05]$ $y : E[-0.5, 10.5, 0.5]$
vladislavleva-4	$\frac{10}{5 + (x-3)^2 + (y-3)^2 + (z-3)^2 + (v-3)^2 + (w-3)^2}$	$U[0.05, 6.05, 1024]$	$U[-0.25, 6.35, 5000]$
vladislavleva-5	$30 \frac{(x-1)(z-1)}{y^2(x-10)}$	$x : U[0.05, 2, 300]$ $y : U[1, 2, 300]$ $z : U[0.05, 2, 300]$	$x : E[-0.05, 2.1, 0.15]$ $y : E[0.95, 2.05, 0.1]$ $z : E[-0.05, 2.1, 0.15]$
vladislavleva-7	$(x - 3)(y - 3) + 2 \sin((x - 4)(y - 4))$	$U[0.05, 6.05, 300]$	$U[-0.25, 6.35, 1000]$
vladislavleva-8	$\frac{(x-3)^4 + (y-3)^3 - (y-3)}{(y-2)^4 + 10}$	$U[0.05, 6.05, 50]$	$E[-0.25, 6.35, 0.2]$

4.1.1 Test Bed

Since real-world problems have intrinsic noise inserted when the data is acquired and pre-processed from the environment [Nettleton et al., 2010], we adopt a test bed composed of synthetic data, generated from 15 known functions selected from the list of benchmark candidates for symbolic regression GP presented in [McDermott et al., 2012], which in turn enumerates benchmarks from the GP and GSGP literature based on some quality criteria.

Table 4.1 presents the function set and the sampling strategy adopted to build the datasets. The training and test sets are sampled independently, according to two strategies. $U[a, b, c]$ indicates a uniform random sample of size c drawn from the interval $[a, b]$ and $E[a, b, c]$ indicates a grid of points evenly spaced with an interval c , from a to b , inclusive. For the former strategy, we generated five sets of samples and for the latter, since the procedure is deterministic, we generated only one sample.

In order to evaluate the impact of noise on GSGP and GP performances, the re-

sponse variable (desired output) of the training instances was perturbed by an additive Gaussian noise with zero mean and unitary standard deviation, applied with probability given by r . We generated datasets with r varying from 0 to 0.2 with steps equal to 0.02, resulting in 11 different levels of noise, in a total of 165 datasets analyzed.

The performance of the methods in the datasets was measured using the Normalized Root Mean Square Error (NRMSE) [Keijzer, 2003; De Falco et al., 2007], given by²:

$$NRMSE = \frac{RMSE \cdot \sqrt{\frac{n}{n-1}}}{\sigma_t} = \sqrt{\frac{\sum_{i=1}^n (y_i - f(x_i))^2}{\sum_{i=1}^n (y_i - \bar{t})^2}}, \quad (4.1)$$

where $\bar{t}$ and σ_t are, respectively, the mean and standard deviation of the target output vector t and f is the model (function) induced by the regression method. NRMSE is equal to 1 when the model performs equivalently to $\bar{t}$ and equal to 0 when the model perfectly fits the data. We used the normalized version of RMSE to be able to compare results from different levels of noise and datasets in a fair way, as described in the next section.

4.1.2 Noise Robustness in Regression

The performance of GSGP and GP in the same dataset with different levels of noise is assessed by the robustness measures presented in Section 3.1.2, namely RLA and ELA, adapted to the regression domain. Instead of using the accuracy—a performance measure for classification methods—we adopted the NRMSE.

Notice that the accuracy is defined in $[0\%, 100\%]$ —or $[0, 1]$ —with higher values meaning better accuracy and, consequently, smaller error. Thus, the larger the RLA or ELA measured values, the less robust is the method to the respective noise level. The NRMSE, on the other hand, is defined in $[0, +\infty)$ and higher values mean greater error.

In this context, we introduce the Relative Increase in Error (RIE) and Equalized Increase in Error (EIE) measures as alternatives to RLA and ELA, respectively, to quantify the noise robustness in the regression domain. RIE and EIE are given by Equations 4.2 and 4.3, respectively, in which $E_{x\%}$ is the NRMSE obtained by the model in the dataset with $x\%$ of noise, $E_{0\%}$ is the NRMSE obtained by the model in

²The presented NRMSE equation regards the training set. However, the formula is easily extensible to the test set.

the dataset with no noise, and a plus one term is added to both denominators in order to avoid division by zero. The higher the values of both measures, the more sensitive the model is to the respective noise level.

$$RIE_{x\%} = \frac{E_{x\%} - E_{0\%}}{1 + E_{0\%}} \quad (4.2)$$

$$EIE_{x\%} = \frac{E_{x\%}}{1 + E_{0\%}} \quad (4.3)$$

Similarly to ELA, we can derive EIE according to Equation 4.4, such that $EIE_{x\%}$ is equal to $RIE_{x\%}$ plus a term depending only on the model NRMSE with no noise—given by $EIE_{0\%}$.

$$EIE_{x\%} = \frac{E_{x\%}}{1 + E_{0\%}} = \frac{E_{x\%} + E_{0\%} - E_{0\%}}{1 + E_{0\%}} = RIE_{x\%} + EIE_{0\%} \quad (4.4)$$

4.1.3 Experimental Analysis

This section presents the experimental analysis of the performance of GSGP in symbolic regression problems with noisy data. We compare the results with a canonical GP [Banzhaf et al., 1998], using the noise robustness measures introduced in Section 4.1.2 and the 15 datasets presented in Table 4.1 with 11 different noise levels. Given the non-deterministic nature of GSGP and GP, each experiment was repeated 50 times. As explained in Section 4.1.1, we resampled five times the data obtained randomly by the uniform strategy. In datasets with this sampling strategy, the experiments were repeated 10 times for each sample, resulting in a total of 50 repetitions.

All executions used a population of 1,000 individuals evolved for 2000 generations with tournament selection of size 7 and 10 for GP and GSGP, respectively. The grow method was adopted to generate the random functions inside the geometric semantic operators, while the ramped half-and-half method was used to generate the initial population, both with maximum individual depth equal to 6. The terminal set included the input variables of each dataset and constant values randomly picked from the interval $[-1, 1]$. The function set included three binary arithmetic operators ($+$, $-$, $\times$) and the analytic quotient (AQ) [Ni et al., 2013], an alternative to the arithmetic division with similar properties, but without discontinuity, given by:

$$AQ(a, b) = \frac{a}{\sqrt{1 + b^2}} \quad (4.5)$$

For GP, the crossover and mutation probabilities were defined as 0.9 and 0.1, re-

spectively. For GSGP, we employed the crossover for Manhattan-based fitness function and mutation operators from [Castelli et al., 2015a], both with probability 0.5. The mutation step required by the mutation operator was defined as 10% of the standard deviation of the outputs (Y) given by the training data.

Despite the fact that in real scenarios noise is found in both training and test data, in this controlled experiment we added noise only to the training set. Although this decision may seem unreasonable at first, it is motivated by the fact that, to measure whether a model generated by a GP method is effective in the presence of noise, we have to verify how it would work in the absence of noise (i.e., if the model it produces can approximate the curve of the original function regardless of being trained under the presence of noise). Had we inserted noise in the test set, good results could indicate that the model learned noisy regions of the curves represented by both the training and test sets.

As a consequence of this decision, training and test values are not comparable to each other (which is not a problem, since we are concentrating on the comparison between the test results of GP and GSGP). That is why the test error presented is, overall, smaller than the training error. In order to investigate this aspect, we carried out a different experiment, in which we also inserted noise on the test set. For a 10% level of noise, for example, the training error was smaller than the test error for 7 out of 8 datasets.

Figure 4.1 shows how the median training and test NRMSE are affected when increasing the percentage of noisy instances. Regarding the results for data with no noise, GSGP presents better median test NRMSE in all but two datasets, *keijzer-6* and *vladislavleva-5*. However, the opposite behavior is observed for noise levels greater than or equal to 18% in *keijzer-1*, 6% in *keijzer-9*, 2% in *vladislavleva-1* and 14% in *vladislavleva-4*. Moreover, GSGP test NRMSE approximates from GP when the noise level increases in the datasets *keijzer-2*, *keijzer-3*, *keijzer-4*, *keijzer-7*, *keijzer-8*, *vladislavleva-2* and *vladislavleva-8*. This behavior may indicate that, although GSGP outperforms GP in low levels of noise in most of the datasets, its performance deteriorates faster than GP when the level of noise increases. Notice that in all experiments the median training NRMSE of the GSGP is smaller than the one obtained by GP, regardless of the behavior of both methods in the test data, which may indicate that GSGP has a greater tendency to overfit noisy data than GP.

Figure 4.2, in turn, shows the median values for the EIE and RIE measures presented in Section 4.1.2, obtained by GSGP and GP methods for different noise levels considering only the test set. When analyzing RIE values, we verify that GSGP is less robust to noise than GP for all noise levels in 10 datasets—*keijzer-2*, *keijzer-3*,

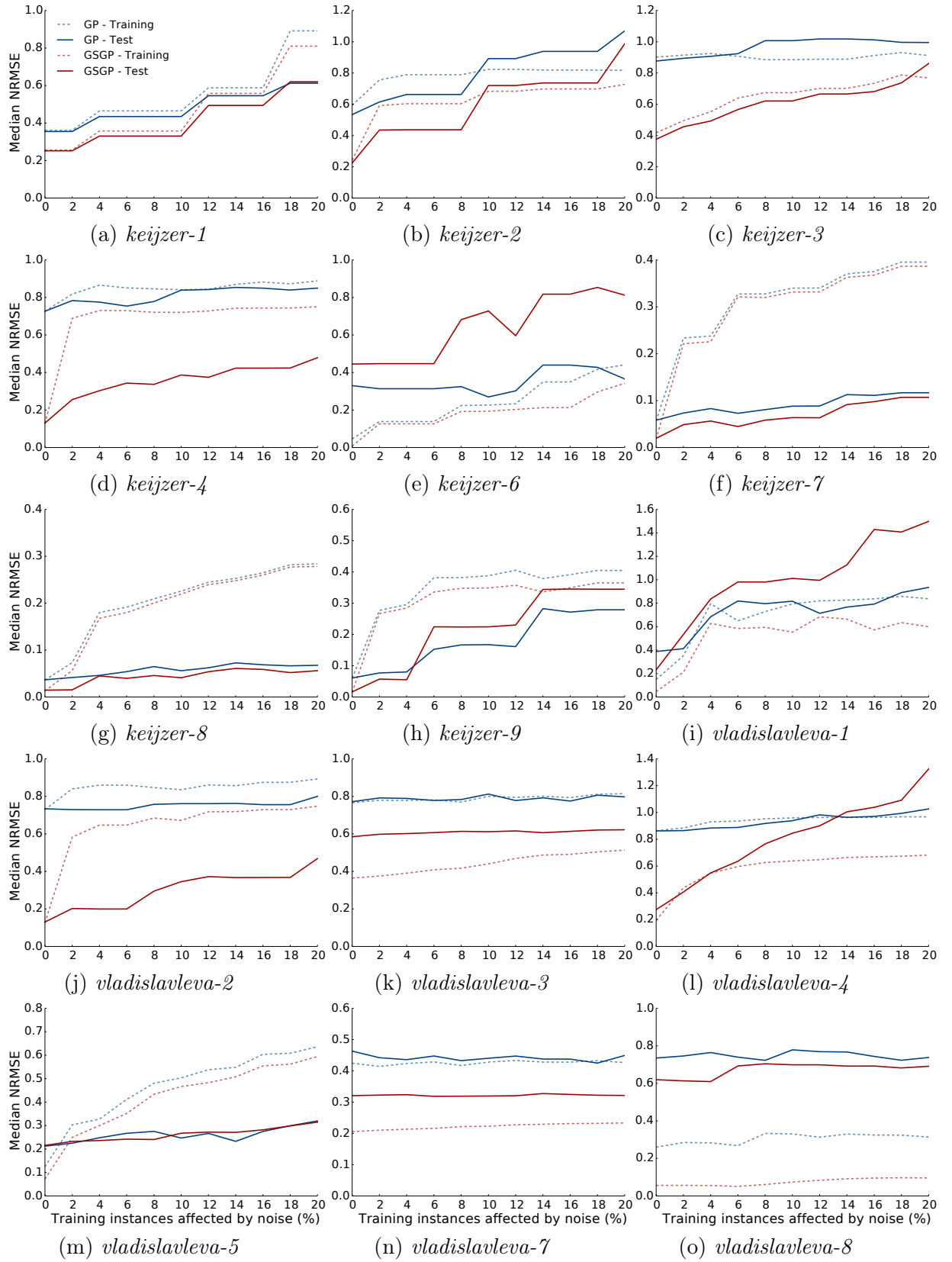


Figure 4.1: Median training and test NRSME obtained by GP and GSGP for each dataset.

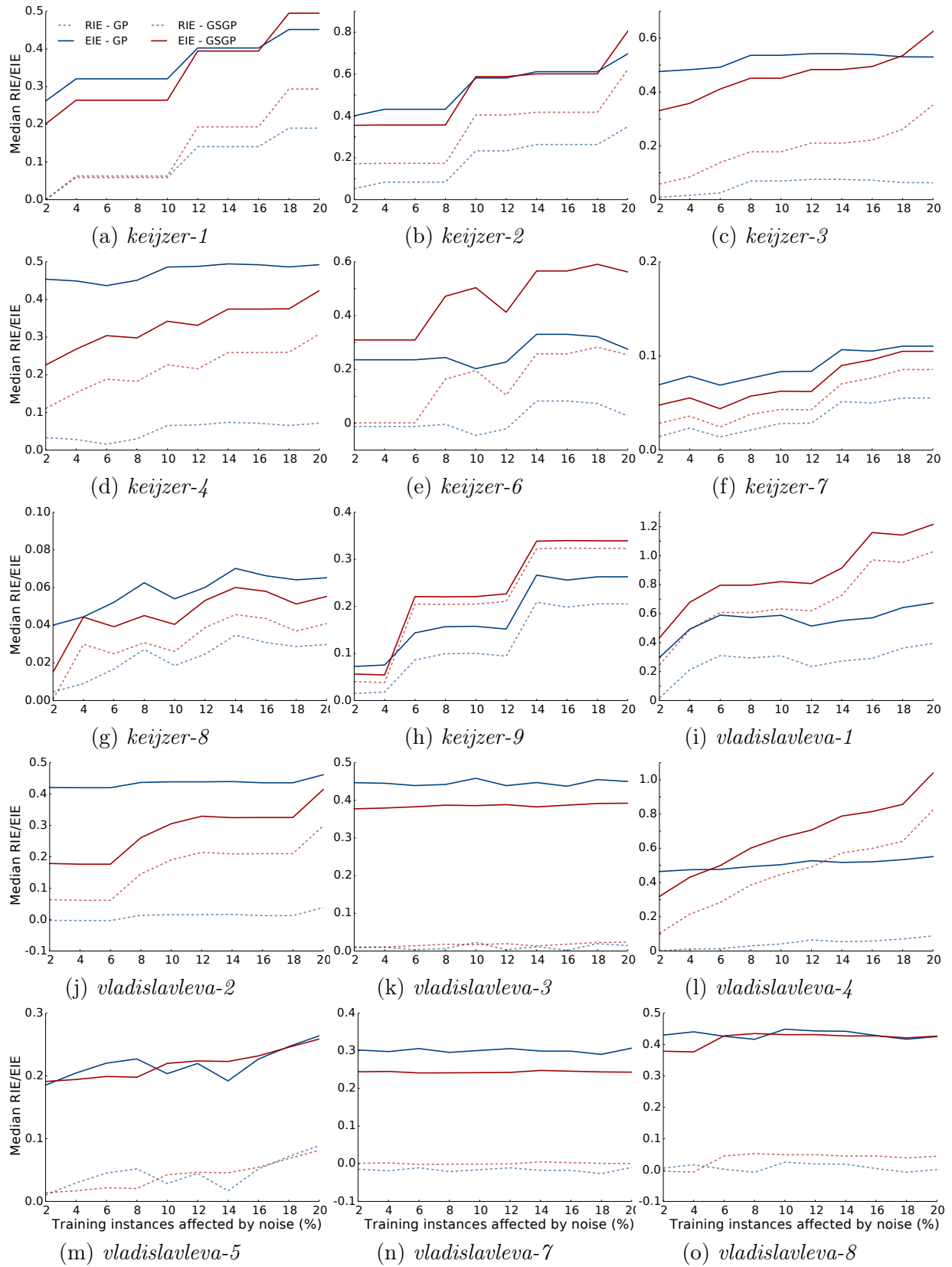


Figure 4.2: Median test RIE and EIE obtained by GP and GSGP for each dataset.

keijzer-4, *keijzer-6*, *keijzer-7*, *keijzer-9*, *vladislavleva-1*, *vladislavleva-2*, *vladislavleva-4* and *vladislavleva-7*—and for noise levels greater than or equal to 4% for *keijzer-1* and *keijzer-8* and 6% in the dataset *vladislavleva-8*.

However, this scenario changes when we look at the values of EIE. GSGP is more robust than GP in all noise levels in six datasets—*keijzer-4*, *keijzer-7*, *keijzer-8*, *vladislavleva-2*, *vladislavleva-3* and *vladislavleva-7*—and the opposite happens in only two datasets—*keijzer-6* and *vladislavleva-1*. Besides, we can observe that GSGP obtains smaller EIE values than GP for noise levels smaller than 18% in the datasets *keijzer-1* and *keijzer-3*. On the other hand, GP outperforms GSGP in terms of EIE for noise levels greater than 4% in the datasets *keijzer-9* and *vladislavleva-4*. These analyses indicate that, overall, GSGP is more robust to noise than GP according to the EIE measure.

The main reason for these contradicting results lies on what these measures regard as important to quantify noise robustness. As presented in Section 4.1.2, the method performance in the dataset with no noise has very low influence in the RLA measure—and consequently in its regression counterpart (RIE). ELA and EIE, on the other hand, add a term to their respective equations to represent the behavior of the model in the data without controlled noise. As GSGP performs better than GP in the majority of scenarios when no noise is present, it is natural that EIE considers it more robust to noise than RIE.

In order to compare the results presented in Figures 4.1 and 4.2, we conducted three paired one-tailed Wilcoxon tests comparing GP and GSGP under the null hypothesis that their median performance—measured by their median test NRMSE, RIE and EIE in all datasets—are equal. The adopted alternative hypotheses differ according to the overall results presented in Figures 4.1 and 4.2: GSGP outperforms GP in terms of NRMSE and EIE and GP outperforms GSGP in terms of RIE. The p -values reported by the tests are presented in Table 4.2. Considering a confidence level of 95%, the symbol $\blacklozenge$ indicates the null hypothesis was not discarded and the symbol $\blacktriangle(\blacktriangledown)$ indicates that GSGP is statistically better (worse) than GP. For the NRMSE measure, GSGP outperforms GP in datasets with 0%, 2%, 4%, 6%, 8% and 12% of noise. However, there are no statistical differences when the noise level is greater than 12%, which indicates that GSGP performance approximates from GP. When analyzing the robustness measures, RIE indicates that GP is more robust than GSGP in all noise levels. However, the same is not true for the EIE measure, which indicates GSGP is more robust than GP with low levels of noise (2% and 4%) and have no significant differences for noise levels greater than 4%.

In conclusion, these results indicate that GP is more robust to all levels of noise

Table 4.2: P -values obtained by the statistical analysis of the performances of GP and GSGP. The symbol $\blacklozenge$ indicates the null hypothesis was not discarded and the symbol $\blacktriangle(\blacktriangledown)$ indicates that GSGP is statistically better (worse) than GP with 95% confidence.

	Training instances affected by noise (%)									
	2	4	6	8	10	12	14	16	18	20
NRMSE	0.006 $\blacktriangle$	0.004 $\blacktriangle$	0.015 $\blacktriangle$	0.032 $\blacktriangle$	0.053 $\blacklozenge$	0.042 $\blacktriangle$	0.115 $\blacklozenge$	0.151 $\blacklozenge$	0.195 $\blacklozenge$	0.262 $\blacklozenge$
RIE	0.003 $\blacktriangledown$	0.002 $\blacktriangledown$	0.001 $\blacktriangledown$	0.000 $\blacktriangledown$	0.000 $\blacktriangledown$	0.000 $\blacktriangledown$	0.000 $\blacktriangledown$	0.000 $\blacktriangledown$	0.000 $\blacktriangledown$	0.000 $\blacktriangledown$
EIE	0.015 $\blacktriangle$	0.021 $\blacktriangle$	0.126 $\blacklozenge$	0.300 $\blacklozenge$	0.381 $\blacklozenge$	0.402 $\blacklozenge$	0.467 $\blacklozenge$	0.381 $\blacklozenge$	0.598 $\blacklozenge$	0.885 $\blacklozenge$

than GSGP when the RIE measure is employed to analyze the outcomes. On the other hand, when the NRMSE or EIE values are analyzed, GSGP outperforms GP in terms of robustness to lower levels of noise and presents no significant differences regarding GP in higher levels of noise. Therefore, although GSGP performs better than GP in low levels of noise, the methods tend to perform equivalently for larger levels of noise.

Chapter 5

Instance Selection for Regression

As we mentioned, when GSGP is used for symbolic regression, the semantics of any individual can be represented as a point in an n -dimensional semantic space. Most datasets used in real-world applications contain a fairly large number of instances, which means that the search takes place in a high-dimensional semantic space, complex and usually difficult to handle. In this context, by reducing the number of input instances we automatically reduce the number of dimensions and the complexity of the search space. Intuitively, the lower complexity associated with the search space can lead to an enhanced regression model.

For the canonical GP framework, this argument regarding the complexity of the semantic space does not apply—canonical genetic operators employed by GP do not ensure any geometric property in the semantic space. However, depending on the dataset, many instances may be representing the same piece of information—a particular region of the input space, for example. Therefore, by removing instances from overrepresented areas of that space, we can reduce the execution time without significantly worsening the quality of the regression model. Likewise, this potential benefit also extends to GSGP.

This chapter presents our approach to explore and evaluate the impact of reducing the number of dimensions of the semantic space through the application of instance selection methods.

We first introduce notations for data instances used throughout the remainder of this thesis. Given the input training set $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ —with $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$ and $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{id}]$, for $i = \{1, 2, \dots, n\}$ —we define $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T$ and $Y = [y_1, y_2, \dots, y_n]^T$ as the matrix $n \times d$ of inputs and the n -element output vector,

respectively. The distance in the input space between two instances, I_i, I_j , is given by:

$$\text{dist}_p(I_i, I_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_p \quad (5.1)$$

where $\|\mathbf{u}\|_p$ denotes the p -norm in the input space of a given a vector $\mathbf{u} \in \mathbb{R}^d$, defined as:

$$\|\mathbf{u}\|_p = \left(\sum_{i=1}^d |u_i|^p \right)^{1/p}. \quad (5.2)$$

The distance is used to compute the set of k nearest neighbors of a given instance $I_i = (\mathbf{x}_i, y_i)$ — $N_i = \{n_{i1}, n_{i2}, \dots, n_{ik}\}$ —and its set of associates $A_i = \{I_j | I_i \in N_j \text{ for } j \in \{1, 2, \dots, n\} \text{ and } i \neq j\}$ —i.e., instances that have I_i among their k -nearest neighbors.

This chapter is organized as follows: we begin by describing, in Section 5.1, our lines of thought and the particular way we exploit existing instance selection preprocessing strategies. In Section 5.2, we introduce a new and integrated IS method, called Probabilistic instance Selection based on the Error (PSE), which selects instances based on their median absolute error. In Section 5.3, we present an experimental analysis of the IS strategies discussed along the chapter, applying them to a collection of real-world and synthetic datasets and discussing the results obtained. Finally, in Section 5.4, we verify if the PSE method can provide good results also in noisy scenarios.

5.1 Pre-Processing Strategies

In this section, we explore IS strategies applied as preprocessing steps. We start by describing, in Section 5.1.1, the methods TCNN and TENN, presented in Section 3.2. These methods follow the same idea used in IS methods built for classification domains, but were adapted so that they are able to handle regression tasks. In Section 5.1.2, we explore the idea of instance weighting as a way of distinguishing instances according to their importance. We also explore dimensionality reduction techniques concerning the input space, producing low dimensional embeddings as an attempt to enhance the notion of closeness between instances.

5.1.1 TENN and TCNN

As previously mentioned, the Threshold Edited Nearest Neighbor (TENN) and Threshold Condensed Nearest Neighbor (TCNN) [Kordos and Blachnik, 2012] adapt instance selection algorithms for classification problems—Edited Nearest Neighbor (ENN) [Wil-

Algorithm 1: TENN

Input: $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^n, k, \alpha$
Output: Instance set $S \subset T$

```

1 Shuffle  $T$ ;
2  $S \leftarrow T$ ;
3 for  $i \leftarrow 1$  to  $n$  do
4    $\hat{y} \leftarrow \text{regr}(\mathbf{x}_i, S \setminus (\mathbf{x}_i, y_i))$ ;
5    $N \leftarrow \text{knn}(k, T)$ ;
6    $\theta \leftarrow \alpha \cdot \text{sd}(N)$ ;
7   if  $\theta = 0$  then
8      $\theta \leftarrow \alpha$ 
9   if  $|y_i - \hat{y}| > \theta$  then
10     $S \leftarrow S \setminus (\mathbf{x}_i, y_i)$ 
11 return  $S$ ;
```

Algorithm 2: TCNN

Input: $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^n, k, \alpha$
Output: Instance set $S \subset T$

```

1 Shuffle  $T$ ;
2  $S \leftarrow (\mathbf{x}_1, y_1)$ ;
3 for  $i \leftarrow 2$  to  $n$  do
4    $\hat{y} \leftarrow \text{regr}(\mathbf{x}_i, S)$ ;
5    $N \leftarrow \text{knn}(k, T)$ ;
6    $\theta \leftarrow \alpha \cdot \text{sd}(N)$ ;
7   if  $\theta = 0$  then
8      $\theta \leftarrow \alpha$ 
9   if  $|y_i - \hat{y}| > \theta$  then
10     $S \leftarrow S \cup (\mathbf{x}_i, y_i)$ 
11 return  $S$ ;
```

son, 1972] and Condensed Nearest Neighbor (CNN) [Hart, 1968], respectively—to the regression domain. They are presented in Algorithms 1 and 2.

These algorithms employ an internal regression method to evaluate the instances according to a similarity-based error. The decision of keeping or removing the i -th instance from the training set is based on the deviation of the instance prediction $\hat{y}_i$ and the expected output y_i , given by $|\hat{y}_i - y_i|$. If this difference is smaller than a threshold θ , $\hat{y}_i$ and y_i are considered similar and the instance is accepted or rejected, depending on the algorithm. The threshold θ is computed based on the local properties of the dataset, given by $\alpha \cdot \text{sd}(N)$, where α is a parameter controlling the sensitivity and $\text{sd}(N)$ returns the standard deviation of the outputs of the set N , composed by the k nearest neighbors of the instance.

The internal regression method adopted by TENN and TCNN—the procedure *regr* presented in Algorithms 1 and 2—can be replaced by any regression method. Our implementation uses a version of the k NN (k -nearest neighbor) algorithm for regression to infer the value of $\hat{y}$ (by taking a weighted average of the regression function in a local space, i.e., the k nearest points to a given point). Besides the training set T , these algorithms receive as input the number of neighbors to be considered and a parameter α , which controls how the threshold is calculated. At the end, the set S of instances selected to be used to train the external regression method is returned.

TENN is a decremental method, starting with all training cases in the set S and iteratively removing the instances diverging from their neighbors. An instance $(\mathbf{x}_i, y_i)$ is considered divergent if the output $\hat{y}$ inferred by the model learned without the instance is dissimilar from its output (y_i). TCNN, on the other hand, is an incremental method,

beginning with only one instance from the training set in S and iteratively adding only those instances that can improve the search. The instance $(\mathbf{x}_i, y_i)$ is added only if the output $\hat{y}$ inferred by the model learned with S diverges from y_i .

5.1.2 Instance Weighting

This section introduces a method based on the hypothesis that the level of concentration of instances in the input space has a crucial impact on the training stage performed by regression methods, including GP-based. This hypothesis comes from the fact that error metrics used to guide the search performed by GP give the same weight to every instance. Consequently, regions of the input space with a higher concentration of instances bias the search, in the sense that the fitness function gives a better reward to individuals with good performance on these dense regions.

Figure 5.1 gives an example of such a behavior. Consider a training set T composed by 60 points evenly distributed in the input space in $[-1.5, 4.5]$, and output defined by Equation 5.3—represented as circles (filled and empty) in the figure. Notice that the intervals $[-1.5, -0.5]$ and $[1, 4.5]$, although with the same distribution of instances in the input space, have a denser distribution in the output space when compared to the interval $(-0.5, 1)$. In order to make the distribution in the output space more balanced, we selected the subset S from T , represented by the filled circles. The red and blue curves represent functions induced by a GP using T and S as training sets, respectively. The red curve converges to a constant that minimizes the error in the dense region, while the blue curve is able to capture the tendency of the original function.

$$f(x) = \frac{\sin(\pi x)^{10} + \sin(\pi x)}{\pi x} \quad (5.3)$$

In such cases, instance selection methods can be quite useful, since they can prevent the creation of a model that is inclined to perform well only on regions with a higher instance density and, in addition, they have the potential to reduce running time without worsening accuracy. We are interested in a method capable of identifying instances with low variation in the output space w.r.t. cases in the same region. These instances could be removed from the training set, resulting in a smaller set in which the distribution of the instances in the input space better reflects the variations in the output space.

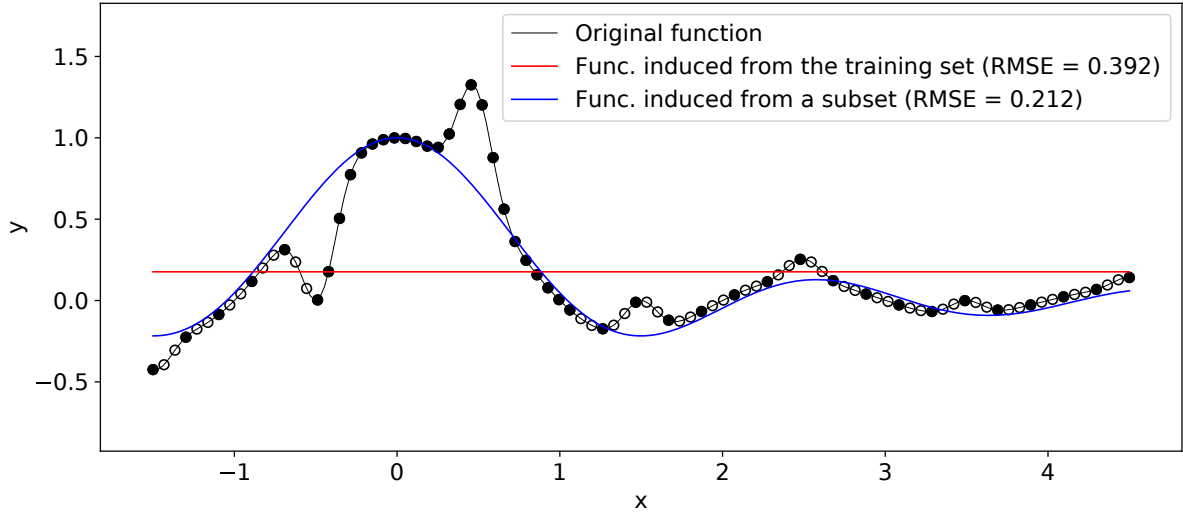


Figure 5.1: Example of an unbalanced dataset and the impact of a instance selection method on the regression.

5.1.2.1 Weighting Process

Vladislavleva et al. [2010] present four different metrics to estimate the importance of each instance in the training dataset. The proximity and surrounding metrics, introduced by Harmeling et al. [2006], measure the degree of isolation and the degree of spread of the neighbors of a given instance, respectively. In turn, the nonlinearity metric is based on the deviation from a least-squares hyperplane passing through the k nearest neighbors. Finally, the remoteness metric is a combination of the proximity and surrounding metrics.

The proximity function γ tries to estimate how isolated an instance is by measuring the average distance to its k nearest neighbors:

$$\gamma(I_i, N_i, k) = \frac{1}{k} \sum_{j=1}^k \text{dist}_p(I_i, n_{ij}) . \quad (5.4)$$

However, the proximity function does not take into account the relative directions between an instance and its neighbors. Therefore, a situation like the one depicted in Figure 5.2 would be indistinguishable for it. This kind of information can be captured using the surrounding function δ , which tries to identify instances on the edge of the response surface—i.e., instances that are not uniformly surrounded by its neighbors—by measuring the average length of the vectors pointing from I_i to its k nearest neighbors:

$$\delta(I_i, N_i, k) = \left\| \frac{1}{k} \sum_{j=1}^k (I_i - n_{ij}) \right\|_p . \quad (5.5)$$

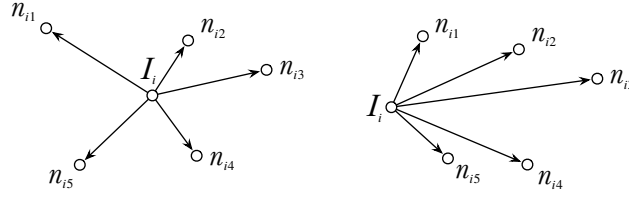


Figure 5.2: Since the surrounding function also takes the directions to the neighbors into account, it assigns higher weight values when the neighbors are located in the same direction.

The choice of k influences the perception of the data: smaller values for k increase the local impact on the metrics while larger values lead to a more global influence. When determining the neighbors and associates of the instances, only the input space was considered.

Another way to detect edge regions is by looking at the shape of the underlying response surface in the neighborhood of each instance. The nonlinearity function ν tries to emphasize areas of nonlinear changes and is defined as the distance from an instance I_i to the least-squares hyperplane passing through its k nearest neighbors:

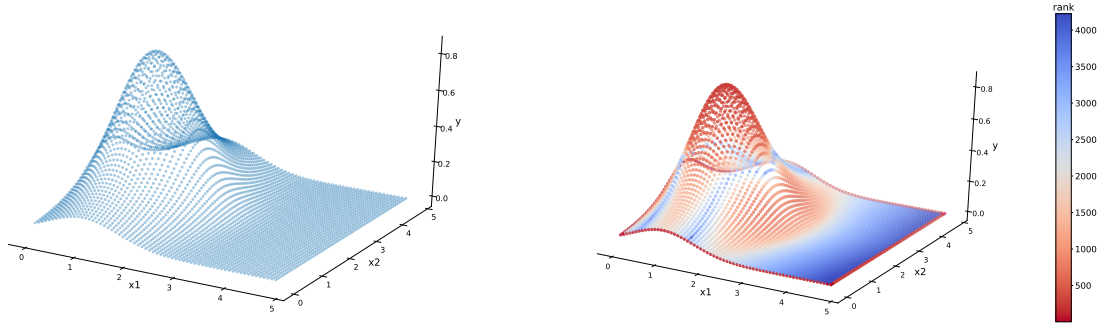
$$\nu(I_i, N_i, k) = \text{dist}(I_i, \Pi_i) \quad , \quad (5.6)$$

where Π_i is the hyperplane passing through the k neighbors of I_i in the input space. The number of neighbors must be equal or greater than the number of input attributes.

The weighting functions presented above allow us to rank the instances according to their inferred importance. Let w be one of these weighting functions. The vector $[w(I_1, N_1, k), w(I_2, N_2, k), \dots, w(I_n, N_n, k)]^T$ results from the application of w to each instance from T . By ordering the instances according to their weights we induce a ranking $R^{(w)}$, with $R_i^{(w)}$ as the index of the instance I_i in this ranking. To illustrate this idea, Figure 5.3 shows the function that represents the *kotanchek* dataset (a), followed by the results of applying the surrounding function (b)—in which instances in shades of blue can be considered easier to predict and therefore are candidates to be removed in the pruning process.

The remoteness function ρ —the fourth weighting function—defines the weight of an instance I_i as the rank of the average value of its proximity and the surrounding weights:

$$\rho(\gamma, \delta, I_i, N_i, k) = \frac{R_i^{(\gamma)} + R_i^{(\delta)}}{2}. \quad (5.7)$$



(a) *kotanche* dataset. The plot shows a set of 4225 instances, sampled from the function $kotanche(x_1, x_2) = e^{-(x_1-1)^2}/(1.2 + (x_2 - 2.5)^2)$, with the input space formed by a $2D$ uniform grid with intervals $[0, 4.8]^2$.

(b) Instances of the *kotanche* dataset colored according to the rank induced by weights produced using the surrounding function.

Figure 5.3: Illustration of the ranking process applied to the *kotanche* dataset.

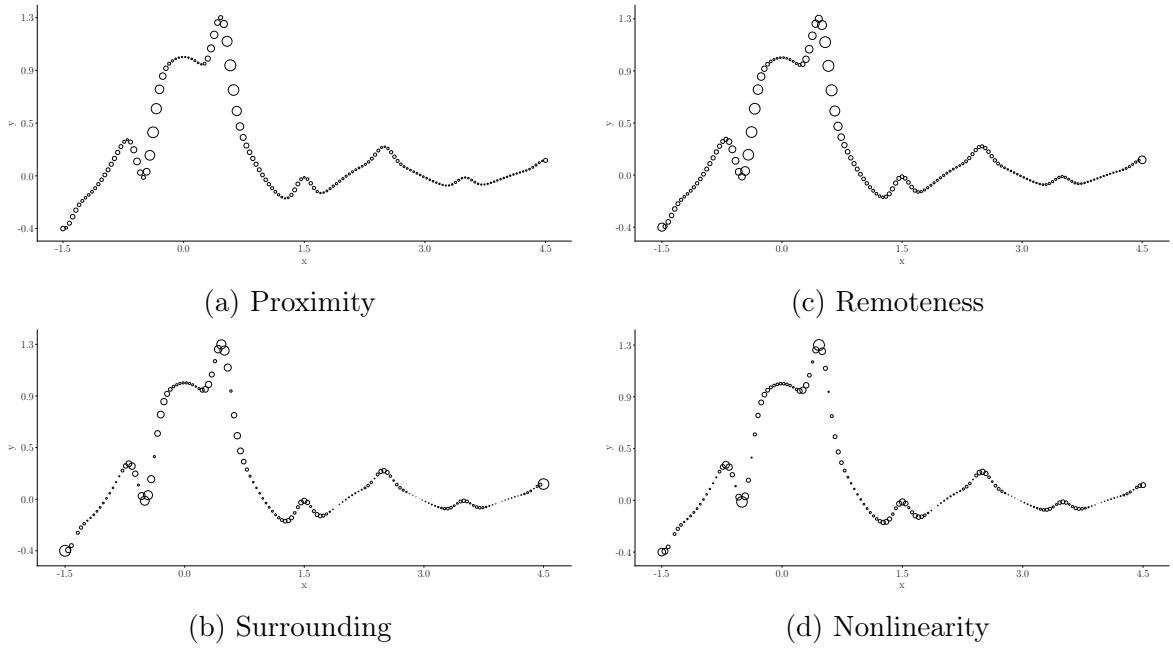


Figure 5.4: Relative weights obtained by applying the different metrics presented by Vladislavleva et al. [2010] to the training set from Figure 5.1.

To illustrate and compare the differences between the weighting functions, we apply them to the same set of instances presented in Figure 5.1. The results are shown in Figure 5.4.

5.1.2.2 Input Space Dimensionality Reduction

Most datasets used in real-world applications—and in our experiments—represent a complex high dimensional input space. The weighting functions presented in the last section rely heavily on the notion of closeness between each instance and their nearest neighbors, which can be deceiving in high dimensional spaces. As an attempt to mitigate this potential problem, as well as gaining some insight about the datasets, we applied dimensionality reduction techniques as one of the steps in our instance selection process.

Dimensionality reduction methods allow us to convert a high-dimensional dataset into a two or three-dimensional map, with the distances between instances in the low-dimensional representation reflecting, as much as possible, the similarities between instances in the high-dimensional dataset. Ideally, such methods would allow us to observe much of the underlying structure of the data and get an idea of how the instances are arranged in the original input space.

By incorporating dimensionality reduction methods in our instance selection process we expect to enhance the perception of adjacency between instances, eventually improving the significance of the values produced by the weighting functions.

We embedded the input attributes of all datasets containing four or more attributes into a two dimensional input space using the following dimensionality reduction techniques:

- Principal Component Analysis (PCA) [[Hotelling, 1933](#)]: uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.
- Isomap Mapping [[Tenenbaum et al., 2000](#)]: seeks a lower-dimensional embedding which maintains geodesic distances between all points.
- Multi-dimensional Scaling (MDS) [[Torgerson, 1952](#); [Borg and Groenen, 1997](#)]: seeks a low-dimensional representation of the data in which the distances respect well the distances in the original high-dimensional space.
- t-distributed Stochastic Neighbor Embedding (t-SNE) [[Maaten and Hinton, 2008](#)]: converts affinities of data points to probabilities. The affinities in the original space are represented by Gaussian joint probabilities and the affinities in the embedded space are represented by Student's t-distributions.

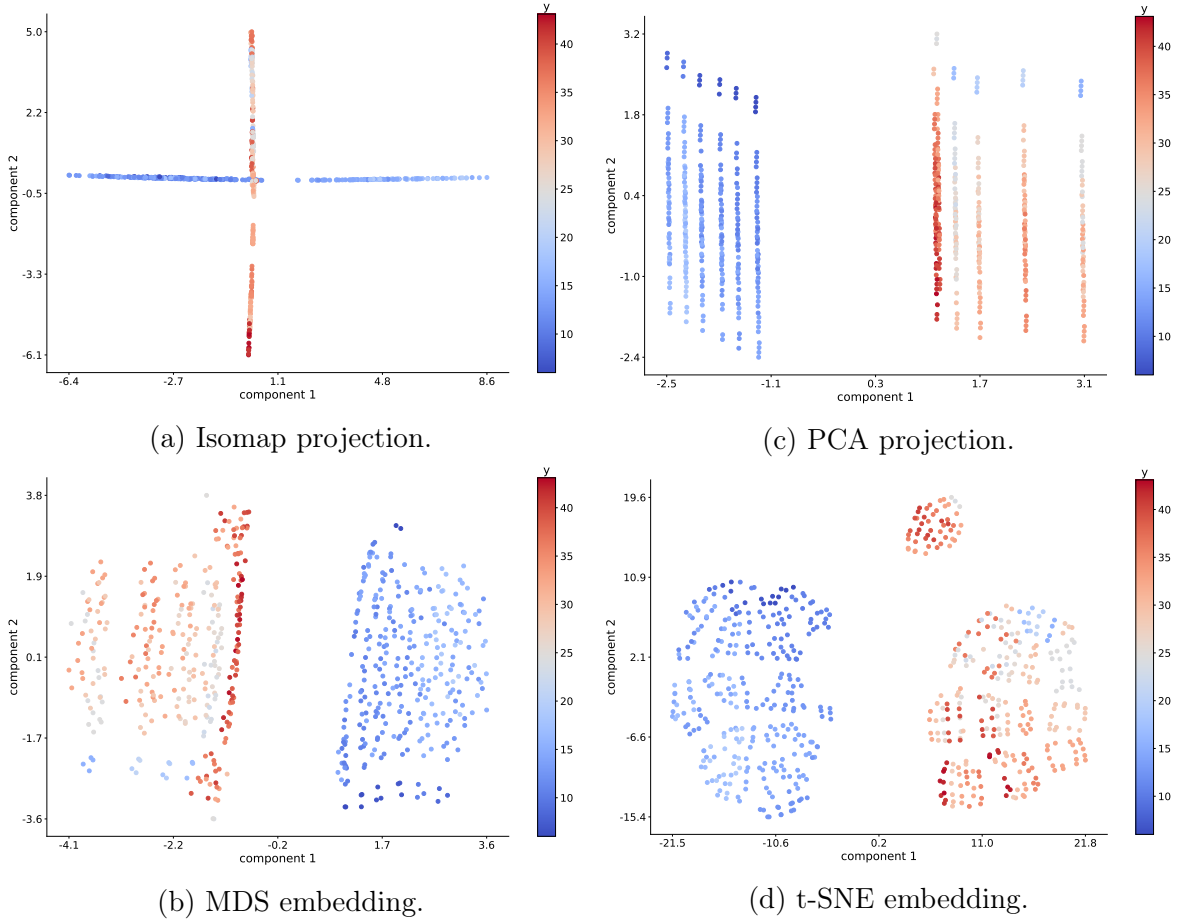


Figure 5.5: Projections and embeddings created by applying the dimensionality reduction methods on one of the training folds of the dataset *energy-heating*, with the number of components set to 2.

Figure 5.5 shows one of the ways the *energy-cooling* dataset—one of the datasets we used in our experiments—can be represented after the application of the four dimensionality reduction techniques.

5.1.2.3 Selection process

We adopted the instance selection method presented in Algorithm 3, which is based on the SMITS procedure [Vladislavleva et al., 2010]. It takes a training set T , a weighting function and some control variables as arguments and outputs a set of instances S , selected according to their information content. The algorithm starts by measuring, in line 2, the distance between each pair of instances in T . Note that the distances are measured considering only the input space, since the closeness between them can be misleading if we also include the output space. In lines 3-6, the algorithm builds, for each instance, their initial sets of neighbors and associates. Then, in lines 7-10,

Algorithm 3: Instance selection

Input: training set (T), number of neighbors (k), distance metric (L),
weighting function (w), merging strategy (m), selection factor (s)

Output: instance set $S \subseteq T$

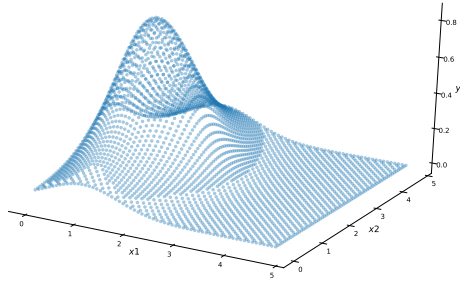
```

1  $X \leftarrow (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ , for  $\mathbf{x}_i \in \mathbb{R}^d$ ; // Inputs
2  $D \leftarrow \text{distBetweenInstances}(X, L)$ ;
3 foreach  $I_i = (\mathbf{x}_i, y_i) \in T$  do
4    $N_i \leftarrow k$  nearest neighbors of  $I_i$  (according to  $\mathbf{x}_i$ );
5   foreach  $n \in N_i$  do
6      $A_n \leftarrow A_n + I_i$ ;
7 if  $w = \rho$  then
8    $W \leftarrow \text{calculateWeights}(T, D, w, L, m)$ ; // Remoteness function
9 else
10   $W \leftarrow \text{calculateWeights}(T, D, w, L)$ ;
11 Let  $R[|T| - k]$  be a new array; // Ranking
12 for  $i \leftarrow 1$  to  $|T| - k$  do
13    $I_l \leftarrow$  instance with the lowest weight;
14    $R[i] \leftarrow I_l$ ;
15    $W_l \leftarrow \infty$ ; // Ignore  $I_l$  thereafter
16    $\text{updateAssociatesWeights}(D, W, A_l, l)$ ;
17 Randomly assign ranks from  $|T| - k + 1$  to  $|T|$  to the last  $k$  instances;
18  $S \leftarrow \text{selectInstances}(T, R, s)$ ;
19 return  $S$ ;

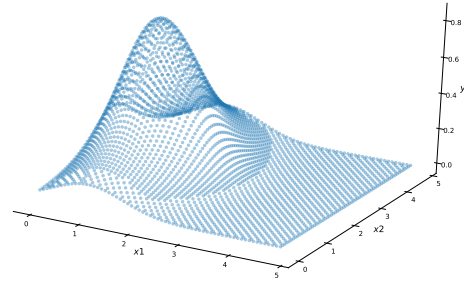
```

it weighs the entire dataset. This is a fairly straightforward process when we use the proximity, surrounding or the nonlinearity functions. For the remoteness function, however, it is necessary to perform a normalization step and include one additional parameter, m . This parameter indicates if the combination method should be ordinal, as defined in Equation 5.7, or if the combined weight should be computed by simply taking the average value between the proximity and surrounding weights. In lines 12-17, the algorithm iteratively ranks the entire dataset. It starts each iteration by finding the instance with the lowest weight and registering it in a ranking array. That instance has to be ignored thereafter, which is forced by setting its weight to ∞ . After that, the algorithm updates the weights of the instances that had I_l among its k nearest neighbors (line 16). The algorithm terminates after creating, in line 18, the set S with the selected instances, which is done by selecting a subset of instances from T based on the ranking created. The size of the subset is defined by the parameter s .

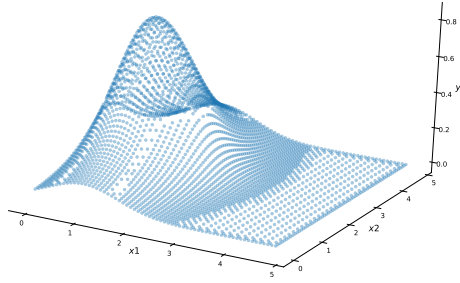
Figure 5.6 shows the result of the instance selection process applied to the *Kotanchek* dataset. The plots represent the same response surface seen in Figure



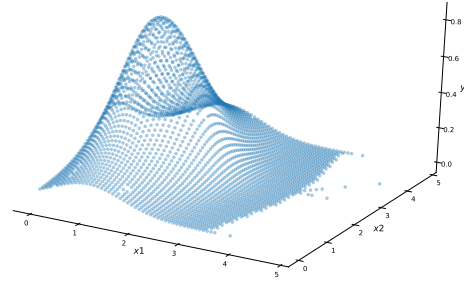
(a) Proximity



(c) Remoteness



(b) Surrounding



(d) Nonlinearity

Figure 5.6: Results of the instance selection process applied to the *kotanchek* dataset. Each plot represents the same response surface seen in Figure 5.3a, but with 25% of the instances removed using one of the four weighting function. The neighborhood size was set to four for all executions.

5.3a, but with 25% of the instances removed using the weighting functions presented in Section 5.1.2.1.

The time taken by our implementation of the SMITS procedure depends on the number of instances n , the number of dimensions d , and the number of neighbors k of each instance. In line 4 of Algorithm 3, we compute the distance matrix, which requires $O(n^2d)$ operations. The proximity and surrounding functions require all k neighbors, which can be found in $O(kn)$ (using k times selection in expected linear time) or $O(n \log n)$ if we sort the neighbors. In our implementation, we opted for the latter approach for considering that large values of k will often be used. Therefore, the complexity of the proximity and surrounding functions is $O(n^2d + n^2 \max(k, \log n))$.

For the nonlinearity function, the process of determining the plane approximating k neighbors requires solving a system of k linear equations, which makes the complexity of the nonlinearity function equals to $O(nk^3)$.

Algorithm 4: PSE method

Input: training set (T), population (pop), lower bound (λ)
Output: instance set $S \subset T$

```

1 foreach  $inst = (\mathbf{x}_i, y_i) \in T$  do    // Compute the median absolute error
2    $E \leftarrow [|p_1(\mathbf{x}_i) - y_i|, |p_2(\mathbf{x}_i) - y_i|, \dots, |p_m(\mathbf{x}_i) - y_i|]$ ;
3    $inst.med \leftarrow median(E)$ ;
4 Sort  $T$  by  $med$  value in descending order;
5  $S \leftarrow \{\}$ ;
6 for  $i \leftarrow 1$  to  $|T|$  do
7    $inst \leftarrow (\mathbf{x}_i, y_i) \in T$ ;
8    $\tilde{r} \leftarrow \frac{(i-1)}{|T|-1}$ ;                // Compute the normalized rank
9    $prob_{sel} \leftarrow 1 - (1 - \lambda) \cdot \tilde{r}^2$ ;    // Probability of selecting  $inst$ 
10  if  $prob_{sel} \geq rand()$  then // Add  $inst$  to  $S$  with probability  $prob_{sel}$ 
11     $S \leftarrow S \cup \{inst\}$ ;
12 return  $S$ ;
```

5.2 PSE

The methods presented so far disregard any information about the external regression algorithm, since they are used in a pre-processing phase. In order to overcome this limitation, we propose a method to select instances based on their median absolute error, considering the output of the programs in the current population. The method, called Probabilistic instance Selection based on the Error (PSE), probabilistically selects a subset from the original training set every ρ generations, as presented in Algorithm 4. The higher the median absolute error, the higher the probability of an instance being selected to compose the training subset used by GSGP. The rationale behind this approach is to give higher probability to instances which are, in theory, more difficult to be predicted by the current population evolved by GSGP.

Given a GSGP population $P = \{p_1, p_2, \dots, p_m\}$, the median absolute error of the i -th instance $(\mathbf{x}_i, y_i) \in T$ is given by the median value of the set $E = \{|p_1(\mathbf{x}_i) - y_i|, |p_2(\mathbf{x}_i) - y_i|, \dots, |p_m(\mathbf{x}_i) - y_i|\}$. These values are used to sort T in descending order, and the position of the instance in T is used to calculate its probability of being selected to be part of the training set.

In order to compute this probability, the method normalizes the rank of the instance in T to the range $[0, 1]$ by

$$\tilde{r} = \frac{i - 1}{|T| - 1}, \quad (5.8)$$

where i is the position of the instance in the ordered set T , $|\cdot|$ denotes the cardinality of the set and $\tilde{r} \in [0, 1]$ is the normalized rank. The value of $\tilde{r}$ is used to calculate the probability of selecting the instance, given by

$$prob_{sel} = 1 - (1 - \lambda) \cdot \tilde{r}^2, \quad (5.9)$$

where λ is a parameter that determines the lower bound of the probability function. The higher the value of λ , the more instances are selected. Figure 5.7 presents the value of $prob_{sel}$ according to $\tilde{r}$, using as example $\lambda = 0.3$. The resulting number of instances is proportional to the area under the curve (shaded area in the figure), equivalent to $\frac{2+\lambda}{3}$.

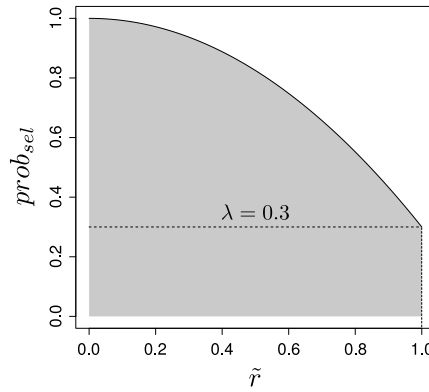


Figure 5.7: Probability of selecting an instance according to its normalized rank, for $\lambda = 0.3$.

5.3 Experimental Analysis

In this section, we present an experimental analysis of the instance selection methods applied to a diversified collection of real-world and synthetic datasets. In essence, our goal is to analyze how the predictive capability of GSGP is affected by the instance selection methods presented so far in this chapter. We start by presenting, in Section 5.3.1 the experimental design adopted in the experiments. In Section 5.3.2, we evaluate the results obtained by GSGP with instance selection performed by TCNN and TENN. In Section 5.3.3, we reason about the effects of the weighting functions and the input space dimensionality reduction in GP and GSGP, contrasting them in order to isolate the impact of the dimensionality reduction of the semantic space. Lastly, in Section 5.3.4, we wrap up our experimental analysis by debating the performance obtained by PSE.

Table 5.1: Datasets used in the experiments.

Dataset	# of attributes	# of instances	Nature	Sources	Exper. estategy
airfoil	6	1503	Real	1, 2	10×5 -CV
ccn	123	1994	Real	4	10×5 -CV
ccun	125	1994	Real	4	10×5 -CV
concrete	9	1030	Real	1, 2	10×5 -CV
energyCooling	9	768	Real	1, 2	10×5 -CV
energyHeating	9	768	Real	1, 2	10×5 -CV
keijzer-6	2	50	Synthetic	1, 3	$50 \times D$
keijzer-7	2	100	Synthetic	3	$50 \times D$
parkinsons	19	5875	Real	2	10×5 -CV
ppb	627	131	Real	1	10×5 -CV
towerData	26	4999	Real	1	10×5 -CV
vladislavleva-1	3	100	Synthetic	1, 3	10×5 -ND
wineRed	12	1599	Real	1, 2	10×5 -CV
wineWhite	12	4898	Real	1, 2	10×5 -CV
yacht	7	308	Real	1, 2	10×5 -CV

Sources: (1) [Albinati et al. \[2015\]](#), (2) [Lichman \[2015\]](#), (3) [McDermott et al. \[2012\]](#), (4) [Chen et al. \[2017\]](#)

5.3.1 Experimental Design

We carried out the experiments using a group of 15 datasets selected from the UCI machine learning repository [[Lichman, 2015](#)], GP benchmarks [[McDermott et al., 2012](#)], and GP studies from the literature [[Albinati et al., 2015](#); [Chen et al., 2017](#)], as presented in Table 5.1. We defined different strategies for the experiments according to the nature and source of the datasets, as detailed in the last column of Table 5.1. For real datasets, we randomly partitioned the data into 5 disjoint sets of the same size and executed the methods 10 times with a 5-fold cross-validations (10×5 -CV). For the synthetic ones, we used two different strategies according to the way the dataset was defined in its original work: datasets generated by non-deterministic sampling functions were resampled 5 times and the experiments were repeated ten times for each sampling (10×5 -ND); experiments with datasets deterministically sampled were repeated fifty times with the same data folds ($50 \times D$). Training and test sets were sampled with the same strategy. The only exception was the *Vladislavleva-1* dataset, in which the training set was sampled following the ten 5-ND strategy (10×5 -ND) and the test set was deterministically sampled one time only, following the original experiment [[McDermott et al., 2012](#)]. In the end, all methods were executed 50 times.

We adopted the paired Wilcoxon test to analyse the statistical differences regarding the results of the experiments performed with this test bed [[Demšar, 2006](#)]. We employed the Wilcoxon test provided by the *stats* package, from the R language, with continuity correction and exact p-value computation [[R Core Team, 2015](#)]. For all these tests we considered a confidence level of 95%.

The parameters configuration used in the experiments of this section is the same as the one presented in Section 4.1.3, except for the number of generations, which in this case was set to 250 for both GP and GSGP. We used the root mean squared error (RMSE) as fitness function.

5.3.2 TCNN and TENN

In this section we compare the results obtained by GSGP with and without the instance selection performed before the evolutionary stage (pre-processing), using the TCNN (GSGP-TCNN) and TENN (GSGP-TENN) methods, with 10 different values for α equally distributed in the intervals $[0.1, 1]$ and $[5.5, 10]$, respectively, and $k = 9$ —adopted after testing a range of values in preliminary experiments. Table 5.2 presents the median training and test RMSE’s and the data reduction obtained with α resulting in the largest data reduction by TCNN and TENN methods—1 and 5.5, respectively.

Table 5.2: Median training and test RMSE and reduction (% *red.*) achieved by the algorithms for each dataset. Values highlighted in bold correspond to test RMSE statically worse than the one obtained by GSGP, according to a Wilcoxon test with 95% confidence.

Dataset	GSGP		GSGP-TCNN			GSGP-TENN			GSGP-Rnd	
	tr	ts	tr	ts	% red.	tr	ts	% red.	tr	ts
airfoil	7.89	8.42	7.76	8.74	38.60	8.06	8.60	1.90	7.65	8.38
concrete	3.65	5.39	2.80	6.40	38.20	3.65	5.21	3.20	3.18	5.95
energyCooling	1.26	1.51	1.28	2.49	14.70	1.28	1.83	36.60	1.19	1.71
energyHeating	0.80	0.96	0.83	1.87	11.10	0.67	1.84	45.40	0.77	1.11
keijzer-6	0.01	0.40	0.01	0.36	10.60	0.00	1.25	53.00	0.01	0.32
keijzer-7	0.02	0.02	0.02	0.02	5.30	0.01	0.40	68.50	0.01	0.05
ppb	0.92	28.74	0.20	32.08	41.50	0.91	28.04	3.80	0.25	30.50
towerData	20.44	21.92	19.82	22.71	12.60	20.44	43.86	41.90	20.40	22.06
vladislavleva-1	0.01	0.04	0.01	0.07	20.90	0.01	0.07	43.40	0.01	0.06
wineRed	0.49	0.62	0.40	0.73	51.10	0.49	0.62	0.10	0.41	0.66
wineWhite	0.64	0.70	0.66	0.78	52.30	0.64	0.69	0.10	0.60	0.71
yacht	2.12	2.52	2.20	5.19	36.90	2.11	2.83	24.30	2.01	2.63

In order to investigate the significance of instance selection methods in GSGP, we compared it with a third strategy, where we randomly selected l instances from each dataset, with no replacement, to compose a new training set used as input by GSGP. The value of l is defined as the smallest size of the sets resulting from TENN and TCNN. Table 5.2 presents the median training and test RMSE of these experiments in the last two columns (denoted as ‘GSGP-Rnd’). The results obtained show that using TCNN and TENN do not make any systematic improvement on GSGP results. Moreover, the results obtained by them are no better than those generated by a random

selection scheme. Hence, the strategies used by these methods do not seem appropriate for the scenario we have.

5.3.3 Instance Weighting

In this section, we present an experimental analysis of the instance selection methods described in Section 5.1.2. Our main goal is to analyze how the predictive capabilities of GP and GSGP are affected by the instance selection performed as a preprocessing step when using the weighting functions and the embedding creation methods presented.

5.3.3.1 Parameter Tuning

In order to avoid that the difference between the range of attribute values causes the data weighting process to be biased towards high range attributes, we scaled the input and output values of all datasets in our test bed to the interval $[0, 1]$. Note, however, that although these scaled values were used by the instance selection method to decide which instances should be kept, the resulting subset is always formed by the original training instances.

We performed preliminary experiments to find out which neighborhood size (k) should be used by the weighting functions. In addition to constants, we also included values relative to the number of instances and attributes of each dataset. These experiments were executed using the same datasets listed in Table 5.1 and indicated that the value 5 provides the best results when considering all datasets and the proximity, surrounding and remoteness functions. The nonlinearity function, which needs to identify a unique hyperplane passing as close as possible to every instance set of neighbors, requires the neighborhood size to be at least as large as the number of attributes. Therefore, in the experiments presented in this section, we adopted the number of attributes as the neighborhood size when using the nonlinearity function and 5 when using any other function.

We also analyzed how the distance function used to find the nearest neighbors affects the instance selection process. We adopted four configurations for the parameter p —used in the equations 5.1 and 5.2: $p = 1$ and $p = 2$, corresponding to the Manhattan and Euclidean distances, respectively, and $p = 0.1$ and $p = 0.5$, generally called fractional distances (actually, when $p < 1$, we cannot consider it as a distance function, since it violates the triangle equality). All experiments were carried out with the four configurations. According to [Vladislavleva et al. \[2010\]](#), fractional distance metrics should be used to find the nearest neighbors when the number of dimensions in a dataset is large. Our experiments, however, did not confirm that hypothesis, even

Table 5.3: Test RMSE obtained by GP on a training set reduced using the proximity function. Values highlighted in bold correspond to test RMSE better than the one obtained by GP when fed with the complete set of instances, indicating results in which the instance selection had a beneficial impact on the search process.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	16.761	15.437	19.472	16.791	18.118	14.702	15.639
ccn	0.151	0.151	0.151	0.150	0.151	0.151	0.149
ccun	399.11	401.86	396.53	400.75	401.14	395.59	397.95
concrete	9.312	9.380	9.351	9.242	8.846	9.434	9.426
energyCooling	3.448	3.385	3.377	3.449	3.409	3.419	3.392
energyHeating	3.104	3.101	3.074	3.103	3.077	3.053	3.075
keijzer-6	0.353	0.386	0.355	0.371	0.382	0.391	0.394
keijzer-7	0.073	0.071	0.084	0.074	0.074	0.083	0.073
parkinsons	10.003	10.023	10.029	10.034	10.016	10.030	10.052
ppb	29.426	28.961	29.813	29.338	29.475	28.757	28.972
towerData	52.170	52.462	50.391	50.858	51.556	51.496	50.202
vladislavleva-1	0.088	0.088	0.087	0.091	0.086	0.083	0.085
wineRed	0.652	0.657	0.652	0.654	0.660	0.661	0.661
wineWhite	0.766	0.761	0.765	0.763	0.764	0.766	0.759
yacht	3.541	3.422	3.673	3.485	3.876	3.474	3.620

for datasets with more than 100 attributes. In order to avoid being unnecessarily extensive, we present in this section only the results regarding the Euclidean distance, which obtained the best results.

5.3.3.2 Experimental results - GP

With the neighborhood sizes and the distance metric defined, we focus on the impact the instance selection process has in the search performed by GP and GSGP. To quantify this impact, we employ two metrics: test RMSE—to measure the error associated with the regression models produced—and execution time, since the weighting process increases computational complexity.

We start by discussing the results obtained by GP. For each dataset, we fed the algorithm with the same set of parameters, with exception of the number of instances removed from the training set, which resulted in training sets of sizes ranging from 75% to 99% relative to their original sizes. For example, for the *keijzer-6* and *parkinsons* datasets—the smallest and the biggest datasets in our testbed—these selection factors correspond to a number of removed instances ranging from 1 to 13 and from 59 to 1469, respectively.

We carried out experiments using subsets built with the four weighting functions presented in the preceding section: proximity, surrounding, remoteness, and nonlin-

Table 5.4: Test RMSE obtained by GP on a training set reduced using the surrounding function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	16.761	18.138	17.562	18.077	16.143	17.226	18.174
ccn	0.151	0.149	0.151	0.152	0.151	0.151	0.153
ccun	399.11	401.18	395.22	401.36	401.22	401.37	399.22
concrete	9.312	9.133	9.287	9.567	9.255	9.172	9.237
energyCooling	3.448	3.416	3.365	3.367	3.408	3.445	3.349
energyHeating	3.104	3.046	3.056	3.024	3.211	3.067	3.011
keijzer-6	0.353	0.356	0.386	0.401	0.375	0.454	0.431
keijzer-7	0.073	0.079	0.080	0.079	0.083	0.085	0.088
parkinsons	10.003	10.029	10.011	10.006	10.031	10.028	10.028
ppb	29.426	28.736	29.360	29.384	28.208	29.296	29.458
towerData	52.170	51.003	51.245	51.020	51.799	51.226	51.241
vladislavleva-1	0.088	0.089	0.092	0.088	0.081	0.085	0.093
wineRed	0.652	0.650	0.651	0.663	0.654	0.663	0.652
wineWhite	0.766	0.764	0.762	0.759	0.762	0.762	0.767
yacht	3.541	3.228	3.445	3.895	3.626	3.756	3.640

Table 5.5: Test RMSE obtained by GP on a training set reduced using the remoteness function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	16.761	14.990	13.898	17.001	17.074	18.253	14.949
ccn	0.151	0.151	0.150	0.151	0.151	0.151	0.151
ccun	399.11	396.48	397.11	401.17	398.43	405.77	398.10
concrete	9.312	9.307	9.398	9.462	9.124	9.249	9.464
energyCooling	3.448	3.385	3.359	3.462	3.430	3.463	3.380
energyHeating	3.104	3.097	3.064	3.030	3.047	2.975	3.091
keijzer-6	0.353	0.386	0.355	0.372	0.403	0.351	0.389
keijzer-7	0.073	0.069	0.076	0.088	0.081	0.077	0.071
parkinsons	10.003	10.027	10.028	10.034	10.020	10.039	10.041
ppb	29.426	28.736	29.075	29.163	30.547	30.298	29.152
towerData	52.170	51.210	51.394	49.928	51.467	49.910	51.800
vladislavleva-1	0.088	0.088	0.090	0.084	0.085	0.091	0.084
wineRed	0.652	0.654	0.659	0.663	0.656	0.649	0.653
wineWhite	0.766	0.764	0.768	0.762	0.763	0.766	0.764
yacht	3.541	3.373	3.284	3.641	3.653	3.575	3.828

earity. The corresponding median test RMSE for these experiments are presented, respectively, in Tables 5.3, 5.4, 5.5, and 5.6.

Again in order to prevent being unnecessarily extensive, we present in this section only the results obtained for the test sets. All results for the training sets are shown in Appendix A. By comparing them to the test results, we observe that the instance selection process based on instance weighting did not lead to any case of overfitting (a conclusion in agreement with the one obtained when analyzing the other IS methods

Table 5.6: Test RMSE obtained by GP on a training set reduced using the nonlinearity function. The *ppb* dataset could not be used, since it has more attributes than instances.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	16.761	18.704	20.267	16.248	16.145	14.875	17.787
ccn	0.151	0.150	0.150	0.149	0.152	0.153	0.151
ccun	399.11	394.16	399.81	401.53	394.84	396.43	397.91
concrete	9.312	9.443	9.657	9.436	9.184	9.232	9.163
energyCooling	3.448	3.387	3.392	3.399	3.410	3.462	3.412
energyHeating	3.104	3.147	3.041	3.104	3.072	3.100	3.057
keijzer-6	0.353	0.370	0.360	0.366	0.380	0.382	0.386
keijzer-7	0.073	0.073	0.070	0.082	0.075	0.076	0.072
parkinsons	10.003	10.034	10.016	10.005	10.032	10.008	10.016
towerData	52.170	51.294	50.722	51.885	51.035	51.456	51.158
vladislavleva-1	0.088	0.089	0.090	0.089	0.085	0.086	0.085
wineRed	0.652	0.659	0.662	0.659	0.665	0.653	0.658
wineWhite	0.766	0.760	0.769	0.764	0.761	0.762	0.765
yacht	3.541	3.672	3.182	3.690	3.630	3.735	3.728

presented in this chapter).

In order to get a visual overview of the results, we analyzed the percentage variation in the test RMSE value as the number of removed instances increases. Due to the large number of overlapping lines, we concentrate our analysis on the general trend shown by the data, highlighting only particularly bad or inconclusive results. Figure 5.8 shows the results corresponding to the GP runs.

The results show that, for most datasets in our testbed, the selection process does not have a strong impact on the regression performed by GP, meaning that the compressed training sets successfully captures the information content of the data. More precisely, regardless of the weighting function used, for 10 out of 15 datasets (9 out of 14 for the nonlinearity function), the test RMSE values reveal no drastic quality changes when comparing the models built using the original and the compressed datasets, since, for any selection level, the corresponding RMSE variations are confined to the range -5% to 5%. In particular, for the proximity and remoteness functions, with 25% of the instances removed, the error value obtained decreases in 10 datasets. We observed poor or inclusive results—in which the error value seems to grow or shift arbitrarily as we increase the selection level—for the datasets *airfoil*, *keijzer-6*, *keijzer-7*, *vladislavleva-1*, and *yacht*.

For the *keijzer-6* and *keijzer-7* datasets, this behavior could be explained by the fact that we used an odd neighborhood size (5) in order to assign a weight value to instances with only one input attribute, with values equally distributed along a single

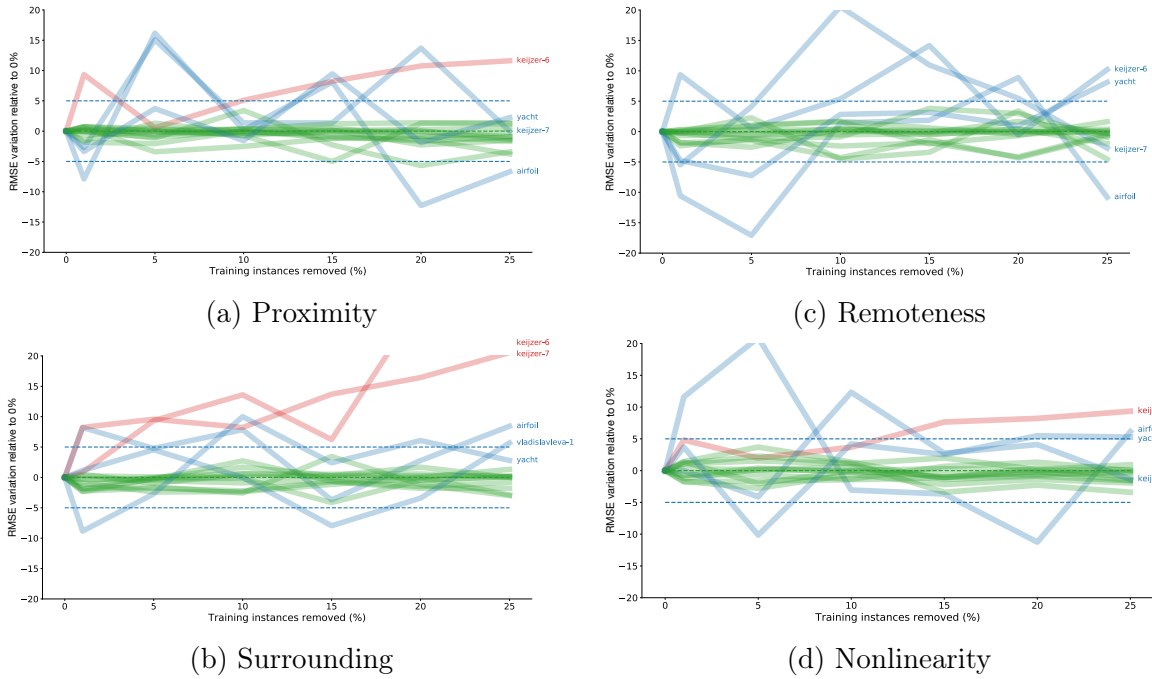


Figure 5.8: Evolution of the RMSE values obtained by GP according to the number of instances removed. The dashed blue lines represent RMSE variations corresponding to -5% and 5%, delimiting a range where the results can be seen as stable. Datasets whose corresponding lines are painted in blue do not stay within this range for most selection levels. Datasets whose corresponding lines are painted in red have a considerable loss of accuracy as the number of selected instances decreased.

dimension. In such cases, the initial weights assigned to instances that are not on the edge of the input space are certainly flawed. Consider, for example, the initial weight calculation for one of the instances of the *keijzer-6* dataset, shown in Figure 5.9. The four nearest neighbors can be easily determined, but the selection of the fifth neighbor requires an arbitrary decision between two instances equally close to the instance we want to weigh. As the selection progresses, this problem tends to be reduced, but not mitigated. However, if the neighborhood size was, in fact, the only reason behind these results, the behavior of these two datasets should change when we used the nonlinearity function, since for them we used $k = 2$. We see, however, a reduction in the level of randomness of the results, but with error values still indicating poor results when compared to those obtained by the other datasets.

It is also interesting to point out that the 5 datasets with poor results are also those with the lowest number of input attributes (ranging from 1 to 6). This may indicate that using a low neighborhood size—close to the number of input attributes—impairs the selection process. If this is true, such a behavior should be repeated with GSGP (since the failure would have occurred in an isolated process). However, instead

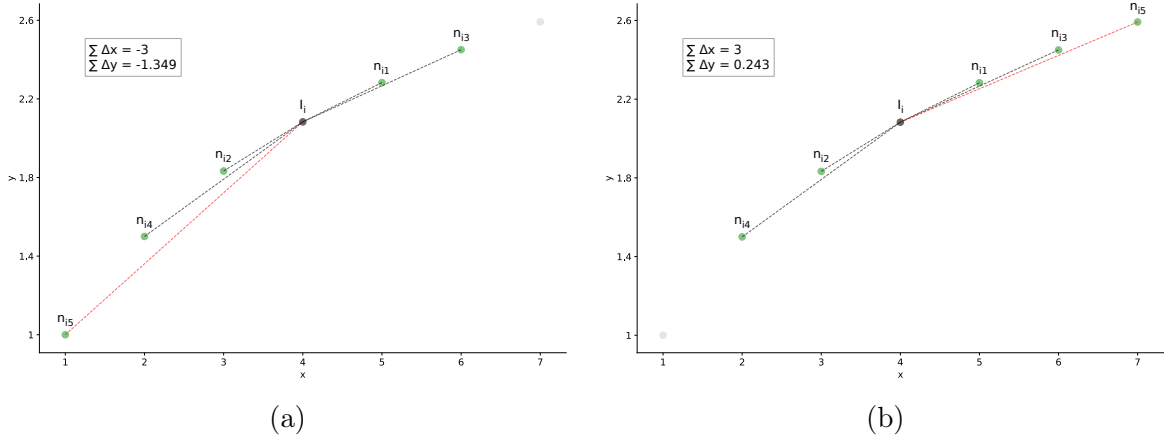


Figure 5.9: Initial weight calculation for one of the instances of the *keijzer-6* dataset. The selection of the fifth neighbor (n_{i5}) requires an arbitrary decision between two instances equally close to the instance for which we want to weigh (I_i), which tends to spoil its initial weight value.

of conducting new experiments, we will return to this question during the GSGP results analysis.

In order to verify how the time complexity of GP is affected by the selection process, we analyze the median execution time required by the method to create the regression models for each dataset. This analysis is shown in Figure 5.10. Despite a certain instability in the results, it is possible to see that the time complexity decreases as expected, falling linearly as we increase the number of instances removed. For the *keijzer-6* and *parkinsons* datasets, for example, the time spent for each execution decreases from 5 to 4 seconds and from 9.7 to 6.6 minutes, respectively. Of course, to fully analyze this aspect it would be necessary to take into account the time spent to perform the selection itself. We decided not to include these times in the analysis for two reasons: (i) during the experiments, it was possible to see that the time spent in this task is always small (less than two minutes, even for the largest datasets) when compared to the time spent by GP to induce its regression models and (ii) our implementation performs the complete instance ranking before creating the subsets (so that in a single execution it is possible to generate as many subsets as desired). Thus, the time used to make the selection does not depend on the number of instances removed. It would be possible to optimize the code in order to avoid unnecessary ranking operations, but that would not bring any concrete benefit.

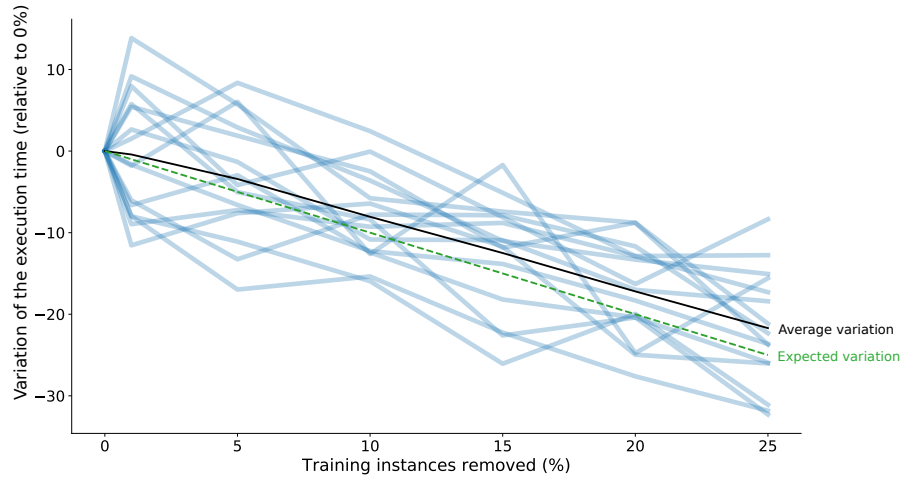


Figure 5.10: Variation of the median execution time for GP runs according to the number of instances removed. The expected variation corresponds to a linear decrease in the time complexity.

Table 5.7: Test RMSE obtained by GSGP on a training set reduced using the proximity function. Values highlighted in bold correspond to test RMSE better than the one obtained by GSGP when fed with the complete set of instances, indicating results in which the instance selection had a beneficial impact on the search process.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	27.083	27.184	27.152	27.113	26.946	26.937	26.804
ccn	0.139	0.138	0.139	0.139	0.139	0.139	0.140
ccun	382.00	380.11	384.65	384.17	378.74	384.22	383.40
concrete	6.871	6.849	6.849	6.881	6.805	6.865	6.901
energyCooling	2.422	2.376	2.377	2.391	2.374	2.375	2.316
energyHeating	1.913	1.935	1.926	1.914	1.845	1.817	1.872
keijzer-6	0.454	0.395	0.439	0.436	0.423	0.418	0.445
keijzer-7	0.025	0.024	0.023	0.025	0.027	0.027	0.026
parkinsons	9.805	9.801	9.819	9.821	9.828	9.848	9.825
ppb	29.697	29.341	29.038	29.449	28.758	29.653	29.351
towerData	33.799	33.870	33.778	33.793	33.438	33.681	33.645
vladislavleva-1	0.061	0.058	0.062	0.057	0.058	0.058	0.055
wineRed	0.629	0.629	0.630	0.630	0.631	0.630	0.630
wineWhite	0.719	0.720	0.720	0.723	0.723	0.722	0.723
yacht	6.251	6.203	6.160	6.195	6.107	6.076	5.990

5.3.3.3 Experimental results - GSGP

In this section, we analyze how GSGP results related to accuracy and running time are affected by the instance selection process, with focus on the differences between them and those obtained by GP. The main purpose of this comparison is to investigate if, and to which extent, the smaller number of instances and resulting dimensionality reduction of the semantic space are able to improve the search performed by GSGP. The

Table 5.8: Test RMSE obtained by GSGP on a training set reduced using the surrounding function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	27.083	27.213	27.073	27.142	27.013	26.993	26.969
ccn	0.139	0.138	0.139	0.139	0.139	0.139	0.140
ccun	382.00	385.84	380.98	382.71	383.36	381.17	382.00
concrete	6.871	6.806	6.795	6.828	6.862	6.854	6.851
energyCooling	2.422	2.414	2.389	2.368	2.330	2.337	2.315
energyHeating	1.913	1.843	1.878	1.847	1.845	1.844	1.872
keijzer-6	0.454	0.422	0.458	0.468	0.489	0.500	0.477
keijzer-7	0.025	0.027	0.025	0.024	0.026	0.027	0.027
parkinsons	9.805	9.804	9.816	9.823	9.839	9.840	9.836
ppb	29.697	28.805	28.367	28.615	29.549	29.057	28.738
towerData	33.799	33.797	33.579	33.675	33.573	33.779	33.790
vladislavleva-1	0.061	0.064	0.061	0.057	0.058	0.056	0.056
wineRed	0.629	0.631	0.630	0.628	0.630	0.632	0.633
wineWhite	0.719	0.719	0.720	0.720	0.721	0.721	0.722
yacht	6.251	6.213	6.135	6.070	6.177	6.085	5.956

experiments were carried out using the same strategy as the one adopted in the executions involving GP. The corresponding median test RMSE for GSGP runs trained with subsets built using the proximity, surrounding, remoteness, and nonlinearity functions are presented, respectively, in Tables 5.7, 5.8, 5.9, and 5.10.

Table 5.9: Test RMSE obtained by GSGP on a training set reduced using the remoteness function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	27.083	27.238	27.214	27.203	26.867	27.065	27.012
ccn	0.139	0.138	0.139	0.138	0.139	0.139	0.140
ccun	382.00	381.24	383.46	383.25	381.27	383.17	383.73
concrete	6.871	6.862	6.747	6.851	6.802	6.805	6.797
energyCooling	2.422	2.412	2.373	2.329	2.367	2.362	2.311
energyHeating	1.913	1.881	1.922	1.864	1.879	1.887	1.822
keijzer-6	0.454	0.395	0.489	0.436	0.434	0.474	0.458
keijzer-7	0.025	0.025	0.023	0.024	0.025	0.027	0.026
parkinsons	9.805	9.807	9.834	9.814	9.829	9.860	9.843
ppb	29.697	29.497	29.712	29.820	29.552	30.154	29.039
towerData	33.799	33.876	33.693	33.838	33.848	33.595	33.944
vladislavleva-1	0.061	0.059	0.056	0.056	0.059	0.055	0.056
wineRed	0.629	0.631	0.630	0.629	0.632	0.632	0.634
wineWhite	0.719	0.719	0.720	0.722	0.720	0.721	0.722
yacht	6.251	6.276	6.153	6.150	6.143	5.970	6.065

Figure 5.11 follows the same idea as Figure 5.8, giving an overview of the results and presenting the changes in the error as the number of instances removed increases.

Table 5.10: Test RMSE obtained by GSGP on a training set reduced using the non-linearity function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	27.083	27.156	27.067	27.106	27.163	27.080	27.084
ccn	0.139	0.139	0.138	0.138	0.139	0.140	0.139
ccun	382.00	379.66	382.57	382.87	383.38	384.47	383.28
concrete	6.871	6.878	6.722	6.927	6.969	6.960	7.121
energyCooling	2.422	2.394	2.373	2.363	2.383	2.392	2.437
energyHeating	1.913	1.983	1.945	1.955	1.951	1.940	1.950
keijzer-6	0.454	0.416	0.469	0.421	0.472	0.412	0.443
keijzer-7	0.025	0.026	0.025	0.025	0.026	0.027	0.030
parkinsons	9.805	9.794	9.791	9.790	9.806	9.798	9.796
towerData	33.799	33.792	34.029	33.662	33.722	33.810	34.337
vladislavleva-1	0.061	0.058	0.057	0.057	0.056	0.061	0.057
wineRed	0.629	0.629	0.630	0.630	0.631	0.635	0.634
wineWhite	0.719	0.719	0.719	0.722	0.722	0.724	0.728
yacht	6.251	6.254	6.210	6.134	6.288	6.253	6.269

The results presented in the previous section indicated that, using appropriate weighting functions, it is possible to compress training sets and keep a similar accuracy when using GP. Therefore, it seems reasonable to infer that results indicating substantial accuracy improvements in the regression models produced by GSGP could be associated with the smaller dimensionality of its semantic space. Nevertheless, the experiments showed that, despite the smaller size of the semantic space, GSGP also did not improve its predictive capabilities for most datasets. The surrounding function reached the best results, with negative RMSE variation, compared to the original dataset, in 9 out of 15 datasets after removing 25% of the training instances. Therefore, in the context of these experiments, there is no further evidence that the reduction in the dimensionality of the semantic space can help to improve GSGP results.

The datasets presenting inconclusive results were narrowed to the synthetic ones: *keijzer-6*, *keijzer-7*, and *vladislavleva-1*. Again, the smaller number of input attributes seems to be related to these results, but we were not able to confirm this claim.

The second point to be analyzed is the impact of the selection process on the execution time of GSGP. Figure 5.12 shows how time elapsed during the execution of GSGP—considering both training and test stages—varies as we increase the number of removed instances. Comparing the results with those obtained by GP, the fluctuations in the execution time are much less severe and, for most datasets, the results still agree with our expectations, presenting a linear decline, albeit in a less pronounced way. The synthetic bases once again exhibited a contradictory behavior, with execution times essentially constant regardless of the number of instances removed.

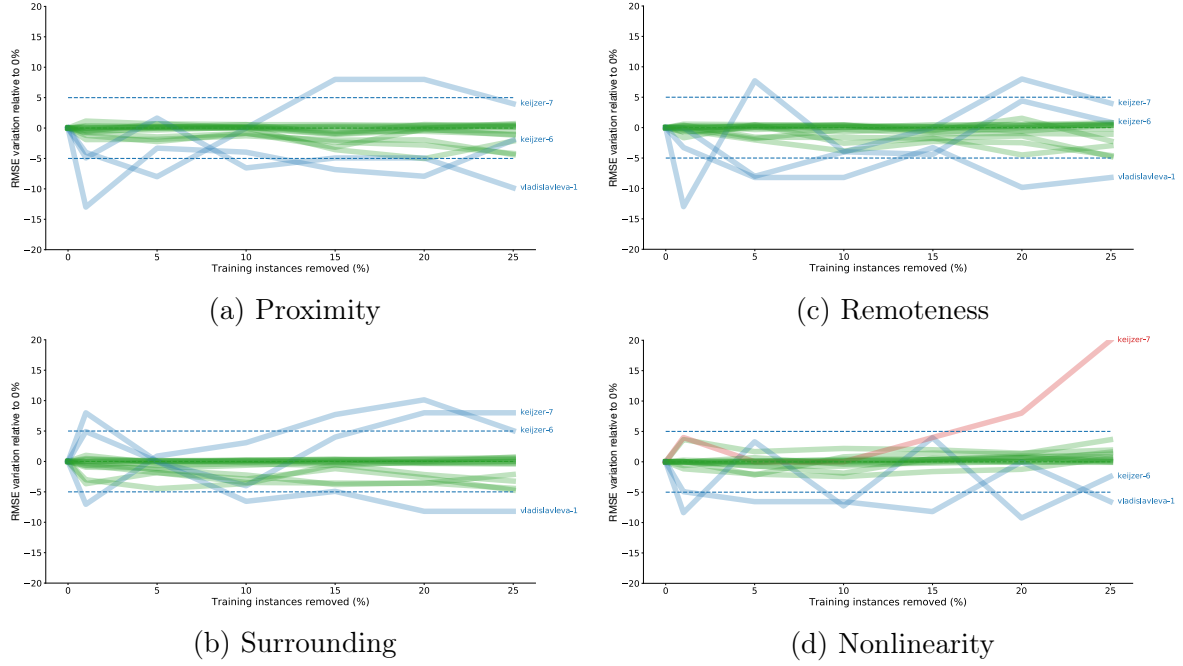


Figure 5.11: Evolution of the RMSE values obtained by GSGP according to the number of instances removed. The dashed blue lines represent RMSE variations corresponding -5% and 5%, delimiting a range where the results can be seen as stable. Datasets whose corresponding lines are painted in blue do not stay within this range for the most part of the selection levels. Datasets whose corresponding lines are painted in red had a considerable loss of accuracy as the selection level was increased.

5.3.3.4 Experimental results - Dimensionality reduction methods

In this section, we analyze if the application of input space dimensionality reduction methods as the first step in the selection process resulted in better error values.

We applied the four methods—Isomap, MDS, PCA, and t-SNE—to the training instances of all datasets with dimensionality ≥ 3 . The resulting embeddings were used to decide which instances to remove during the selection process. However, the selected instances have the original number of input attributes. All methods performed similarly and, because of space limitations, we restrict ourselves to the method with the best results (t-SNE) with GSGP.

Tables 5.11, 5.12, 5.13, and 5.14 presents the median RMSE in the test sets, according to 50 executions, and results can be better visualized in Figure 5.13. Overall, the results were similar to those obtained in the preceding section. Still, it seems that the use of input dimensionality reduction methods has a negative impact on the instance selection process. Using the nonlinearity function, for example, GSGP reached better solutions in only 2 datasets after removing 25% of the training instances.

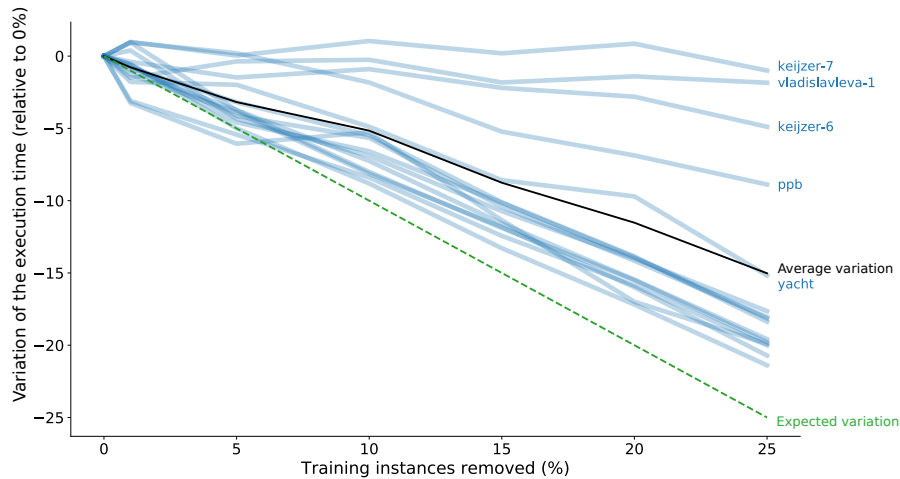


Figure 5.12: Variation of the median execution time for GSGP runs according to the number of instances removed. The expected variation corresponds to a linear decrease in the time complexity.

Table 5.11: Test RMSE obtained by GSGP on a training set embedded using the t -SNE method and reduced using the proximity function. Values highlighted in bold correspond to test RMSE better than the one obtained by GSGP when fed with the complete set of instances, indicating results in which the instance selection had a beneficial impact on the search process.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	27.083	27.139	27.066	27.086	27.140	27.296	27.202
ccn	0.139	0.138	0.138	0.138	0.138	0.138	0.137
ccun	382.00	380.17	380.51	381.74	386.63	383.92	386.10
concrete	6.871	6.783	6.944	6.800	6.803	6.815	7.031
energyCooling	2.422	2.375	2.365	2.354	2.323	2.293	2.329
energyHeating	1.913	1.938	1.902	1.853	1.869	1.846	1.806
parkinsons	9.805	9.805	9.835	9.837	9.837	9.851	9.864
ppb	29.222	28.848	29.157	27.498	29.148	29.449	29.618
towerData	33.799	34.066	34.211	34.070	33.691	34.006	34.278
wineRed	0.629	0.629	0.630	0.631	0.633	0.633	0.636
wineWhite	0.719	0.720	0.719	0.720	0.722	0.724	0.725
yacht	6.251	6.243	6.176	6.230	6.137	6.182	6.215

5.3.4 PSE

In this section, we first investigate the sensitivity of PSE parameters and then compare the performance of GSGP with and without the PSE method. PSE parameters ρ and λ have a direct impact on the number of instances selected and how they are selected. In order to analyze their impact on the search, we fixed the GSGP parameters and focused on looking at the results as we varied these parameters. The values of ρ were

Table 5.12: Test RMSE obtained by GSGP on a training set embedded using the t -SNE method and reduced using the surrounding function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	27.083	27.109	27.234	27.083	26.889	27.190	26.954
ccn	0.139	0.138	0.138	0.139	0.138	0.137	0.137
ccun	382.00	382.30	384.71	381.93	384.63	385.21	385.60
concrete	6.871	6.795	6.823	6.939	6.866	6.979	7.031
energyCooling	2.422	2.347	2.398	2.344	2.356	2.343	2.325
energyHeating	1.913	1.902	1.940	1.836	1.808	1.922	1.832
parkinsons	9.805	9.809	9.810	9.807	9.823	9.817	9.811
ppb	29.222	28.965	29.207	28.527	28.753	29.836	28.859
towerData	33.799	33.956	33.799	33.897	33.900	34.120	34.086
wineRed	0.629	0.630	0.633	0.632	0.632	0.634	0.637
wineWhite	0.719	0.720	0.719	0.722	0.724	0.724	0.722
yacht	6.251	6.264	6.177	6.184	6.269	6.170	6.203

Table 5.13: Test RMSE obtained by GSGP on a training set embedded using the t -SNE method and reduced using the remoteness function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	27.083	27.219	27.051	27.195	27.082	27.002	27.175
ccn	0.139	0.138	0.138	0.138	0.138	0.138	0.137
ccun	382.00	381.76	385.49	380.72	386.51	383.40	385.16
concrete	6.871	6.844	6.813	6.867	6.890	6.972	6.914
energyCooling	2.422	2.367	2.387	2.358	2.341	2.334	2.310
energyHeating	1.913	1.912	1.920	1.885	1.854	1.875	1.834
parkinsons	9.805	9.813	9.854	9.852	9.870	9.852	9.868
ppb	29.222	28.775	29.062	29.614	29.295	29.766	30.422
towerData	33.799	34.143	33.907	33.729	33.901	33.978	34.077
wineRed	0.629	0.630	0.631	0.635	0.634	0.639	0.641
wineWhite	0.719	0.722	0.724	0.723	0.728	0.728	0.727
yacht	6.251	6.250	6.198	6.203	6.053	6.187	6.349

set to 5, 10 and 15 while we varied the value of λ in 0.1, 0.4 and 0.7. Table 5.15 presents the median training RMSE obtained by GSGP with these PSE configurations.

The experiments with PSE adopt the values of ρ and λ resulting in the smallest median training RMSE, as presented in Table 5.15. Table 5.16 presents the median training and test RMSE obtained by GSGP and by GSGP with PSE (GSGP-PSE). In order to identify statistically significant differences, we performed Wilcoxon tests with 95% confidence level, regarding the test RMSE of both methods in 50 executions. The symbol $\blacktriangle(\blacktriangledown)$ in the last column indicates datasets where the GSGP-PSE performed better (worse) than the GSGP. Overall, GSGP with PSE performs better in terms of test RMSE than GSGP without PSE, being better in five datasets and worse in one.

Table 5.14: Test RMSE obtained by GSGP on a training set embedded using the *t-SNE* method and reduced using the nonlinearity function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	27.083	27.146	27.197	27.341	27.239	27.211	27.118
ccn	0.139	0.138	0.139	0.139	0.139	0.139	0.139
ccun	382.00	382.24	381.24	384.13	387.58	386.40	385.89
concrete	6.871	6.835	6.833	6.915	6.974	6.845	6.993
energyCooling	2.422	2.375	2.375	2.370	2.366	2.353	2.376
energyHeating	1.913	1.899	1.931	1.921	1.889	1.907	1.950
parkinsons	9.805	9.801	9.800	9.813	9.811	9.817	9.828
towerData	33.799	33.630	33.838	34.299	34.203	34.625	34.976
wineRed	0.629	0.630	0.629	0.634	0.634	0.636	0.636
wineWhite	0.719	0.719	0.720	0.721	0.724	0.725	0.725
yacht	6.251	6.314	6.303	6.234	6.215	6.172	6.094

Table 5.15: Median training RMSE of the PSE with different values of λ and ρ for the adopted test bed. The smallest RMSE for each dataset is presented in bold (ties were decided by checking differences in less significant digits, when they existed, or randomly, otherwise).

Dataset	$\rho = 5$			$\rho = 10$			$\rho = 15$		
	$\lambda = 0.1$	$\lambda = 0.4$	$\lambda = 0.7$	$\lambda = 0.1$	$\lambda = 0.4$	$\lambda = 0.7$	$\lambda = 0.1$	$\lambda = 0.4$	$\lambda = 0.7$
airfoil	8.03	8.15	8.11	7.97	8.05	8.16	8.12	8.05	8.11
concrete	3.35	3.49	3.56	3.35	3.45	3.58	3.34	3.45	3.56
energyCooling	1.13	1.19	1.23	1.12	1.18	1.22	1.12	1.17	1.23
energyHeating	0.66	0.72	0.77	0.67	0.72	0.76	0.67	0.71	0.76
keijzer-6	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
keijzer-7	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
ppb	0.50	0.65	0.81	0.53	0.65	0.80	0.52	0.63	0.76
towerData	19.22	19.74	19.98	19.22	19.61	20.09	19.18	19.61	19.92
vladislavleva-1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
wineRed	0.47	0.48	0.49	0.47	0.48	0.49	0.47	0.48	0.49
wineWhite	0.63	0.64	0.64	0.63	0.64	0.64	0.63	0.64	0.64
yacht	1.94	2.02	2.09	1.94	2.01	2.08	1.94	2.00	2.09

Figure 5.14 compares the evolution of the fitness of the best individual along the generations in the training and test sets for GSGP and GSGP-PSE, for two different datasets. Note that GSGP errors are overall higher than PSE. For instance, looking at the convergence of the dataset *towerData*, if we stop the evolution at generation 1,000, GSGP would have a test error of 25.02 and GSGP-PSE of 23.64. GSGP needs 293 more generations to reach that same error.

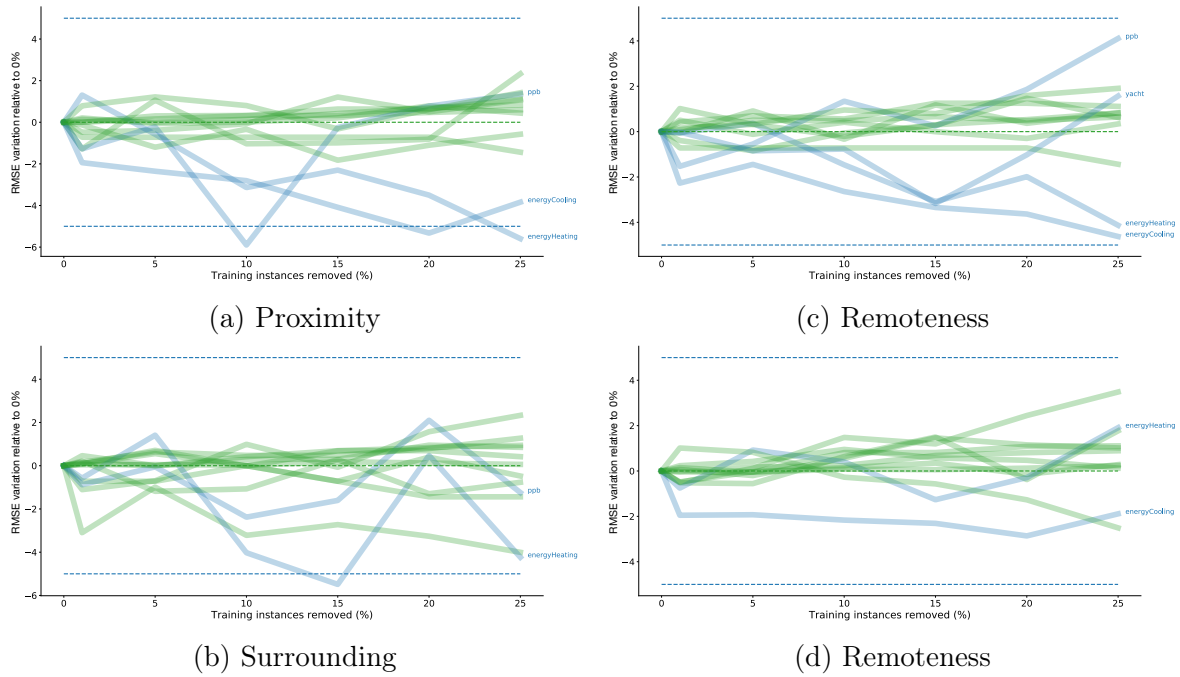


Figure 5.13: Evolution of the RMSE values obtained by GSGP combined with an embedding method, according to the number of instances removed. The dashed blue lines represent RMSE variations corresponding -5% and 5%, delimiting a range where the results can be seen as stable. Datasets whose corresponding lines are painted in blue do not stay within this range for the most part of the selection levels. Datasets whose corresponding lines are painted in red had a considerable loss of accuracy as the selection level was increased.

5.4 Noise Effects on PSE

As we mentioned, distinguishing outliers from instances that are likely to represent relevant information is not a trivial task, since the elements of both groups can be identified by the lower similarity to their neighbors. Therefore, it seems reasonable to think that regression models created with training sets reduced using instance selection methods may be less robust against noise when compared to models induced from the original training sets.

In this section, we investigate if the predictive capabilities of PSE—the instance selection method with the best results—are affected by noisy data in a more pronounced way when compared to GSGP (considering the version in which PSE was built upon).

5.4.1 Experimental Analysis

We carried out the experiments using a group of eight synthetic datasets—generated from the functions highlighted in bold in Table 4.1. The sampling strategies and

Table 5.16: Median training and test RMSE's obtained for each dataset. The symbol $\blacktriangle$ ($\blacktriangledown$) indicates GSGP-PSE is statistically better (worse) than GSGP in the test set according to a Wilcoxon test with 95% confidence.

Dataset	GSGP		PSE	
	tr	ts	tr	ts
airfoil	7.88	8.42	7.97	8.55
concrete	3.65	5.39	3.34	5.24 $\blacktriangle$
energyCooling	1.26	1.51	1.12	1.38 $\blacktriangle$
energyHeating	0.80	0.96	0.66	0.84 $\blacktriangle$
keijzer-6	0.01	0.40	0.01	0.32
keijzer-7	0.02	0.02	0.02	0.02
ppb	0.92	28.74	0.50	28.96 $\blacktriangledown$
towerData	20.44	21.92	19.18	20.95 $\blacktriangle$
vladislavleva-1	0.01	0.04	0.01	0.05
wineRed	0.49	0.62	0.47	0.62
wineWhite	0.64	0.70	0.63	0.69
yacht	2.12	2.52	1.94	2.47 $\blacktriangle$

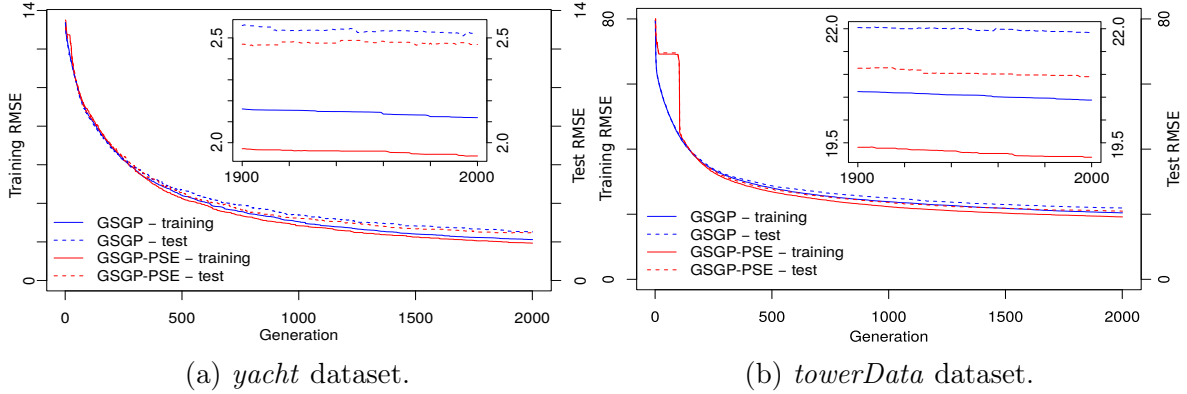


Figure 5.14: Median RMSE in the training and test sets over the generations for GSGP with and without PSE for the *yacht* and *towerData* datasets. The insets show an enlargement of the plots for the last 100 generations.

experiment settings are the same used in the experiments presented in section 4.1.

We compared GSGP and PSE based on the change in their test RMSE as the number of instances affected by noise increases. Table 5.17 presents the results corresponding to experiments in which Gaussian noise was added to 0% and 20% of the training instances. Figure 5.15, in turn, shows how the median training and test RMSE are affected when increasing the percentage of noisy instances.

Overall, the experiments reinforced the conclusion that, depending on the dataset, noisy data can have a great impact on the search process. However, they also indicate that both GSGP and PSE present a similar sensibility to noisy data. Based on the results, it seems reasonable to argue that PSE does not excessively emphasize instances on edges of the input space and, therefore, does not systematically decrease the robustness of GSGP.

Table 5.17: Median training and test RMSE's obtained by GSGP and PSE, for experiments in which Gaussian noise was added to 0% and 20% of the training instances. Values inside parentheses indicate the error variation comparing the latter configuration relative to the former.

Dataset	0%				20%			
	GSGP		PSE		GSGP		PSE	
	tr	ts	tr	ts	tr	ts	tr	ts
keijzer-1	0.028	0.028	0.058	0.056	0.106	0.062 (125%)	0.128	0.086 (54%)
keijzer-6	0.007	0.399	0.009	0.294	0.360	0.470 (18%)	0.459	0.401 (36%)
keijzer-7	0.017	0.018	0.017	0.017	0.346	0.100 (469%)	0.344	0.112 (539%)
keijzer-9	0.015	0.016	0.015	0.016	0.389	0.148 (846%)	0.381	0.169 (976%)
vladislavleva-1	0.011	0.040	0.009	0.045	0.256	0.389 (868%)	0.245	0.393 (776%)
vladislavleva-2	0.028	0.029	0.024	0.025	0.295	0.146 (402%)	0.327	0.145 (490%)
vladislavleva-7	0.733	1.264	0.679	1.259	0.815	1.334 (6%)	0.771	1.332 (6%)
vladislavleva-8	0.084	1.506	0.063	1.625	0.171	1.738 (15%)	0.138	1.744 (7%)

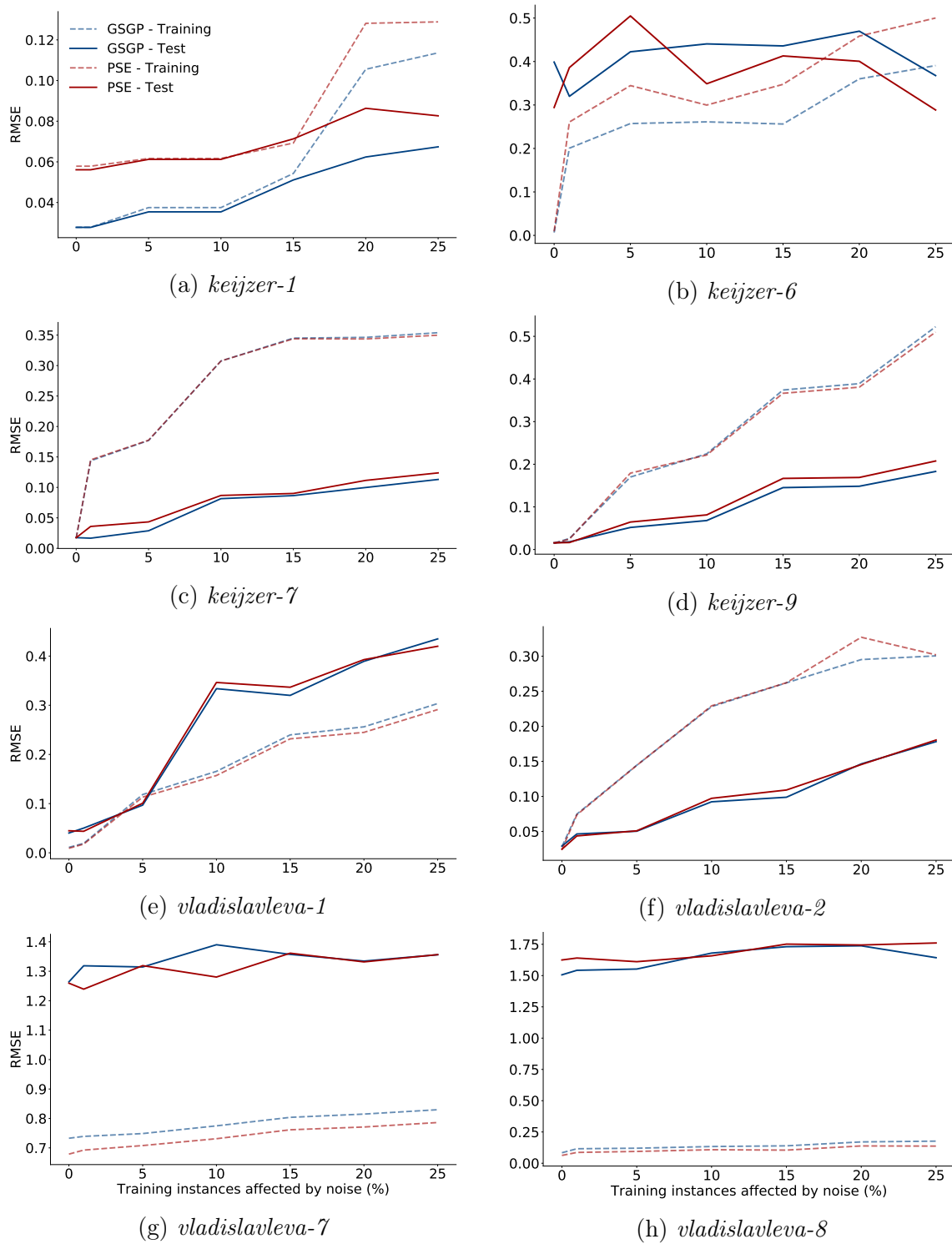


Figure 5.15: Median training and test RMSE obtained by GSGP and PSE for each dataset.

Chapter 6

Conclusions and Future Work

In this thesis, we presented an analysis of methods that allow reducing the size of the search space handled by GSGP, along with studies addressing noise impact on genetic programming and on the instance selection methods proposed.

We first presented an analytic study of the impact of noisy data on the performance of GSGP when compared to GP in symbolic regression problems. The performance of both methods was measured by the normalized RMSE and two robustness measures adapted from the classification literature to the regression domain, namely Relative Increase in Error (RIE) and Equalized Increase in Error (EIE), in a test bed composed of 15 synthetic datasets, each of them with 11 different levels of noise.

Results indicated that GP is more robust to all levels of noise than GSGP when the RIE measure is employed to analyze the outcomes. However, when the NRMSE or EIE values were analyzed, GSGP outperformed GP in terms of robustness to lower levels of noise and presented no significant differences regarding GP in higher levels of noise. Overall, these outcomes indicate that, although GSGP performs better than GP in low levels of noise, the methods tend to perform equivalently for larger levels of noise.

Regarding the strategies to reduce the dimensionality of the semantic space, our line of work was based on three main strategies, involving methods that select instances by (i) adapting existing methods from the classification domain, (ii) taking into account the relative importance of them, and (iii) integrating the selection to the evolutionary process.

The methods related to the first strategy—TCNN and TENN—consisted of modified versions of well-known instance selection methods used in classification contexts, so that they are able to handle regression tasks. The results showed that, in terms of test RMSE, GSGP performs better when it is fed with the whole dataset rather than

when it is fed with subsets built using TCNN or TENN.

The second strategy involved the application of four weighting functions in order to estimate the importance of each instance relative to its k nearest neighbors. We also applied four dimensionality reduction techniques in order to improve the notion of closeness between instances. Experiments were performed in a collection of 15 datasets and showed that the subsets built using the weighting functions were able to capture the underlying structure of the datasets, allowing GSGP to induce faster models with similar quality. On the other hand, the application of the functions were not able to confirm that a reduction in the size of the semantic space has beneficial impacts on the search performed by GSGP.

The last strategy introduced a new method, called PSE, that selects instances during the evolutionary process, taking into account their impact on the search. Experiments indicated that the integration between GSGP and PSE can improve the test RMSE in relation to GSGP alone and, therefore, that the reduction in the size of search space can bring beneficial impacts to the search performed by GSGP. We also analysed the impact of noisy data on the performance of PSE. The results showed that PSE can be as robust as GSGP and, therefore, can also be employed in noisy real-world scenarios.

Given these conclusions, potential future works include:

- Investigating techniques to identify the noisy instances in order to remove them or minimize their importance during the search;
- Analyzing the effect of fitness functions that weight semantic space dimensions;
- Studying approaches to insert information about noisy instances to the instance selection process;
- Analyzing the instance selection methods in other common regression techniques, such as polynomial regression.

Appendix A

Training Results for Instance Weighting

The tables in this appendix show the results obtained in the training data when instance selection considered different functions for instance weighting.

Table A.1: Training RMSE obtained by GP on a training set reduced using the proximity function. Values highlighted in bold correspond to training RMSE better than the one obtained by GP when fed with the complete set of instances, indicating results in which the instance selection had a beneficial impact on the search process.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	17.353	15.240	19.440	16.857	18.463	14.734	15.448
ccn	0.146	0.147	0.150	0.152	0.155	0.157	0.159
ccun	385.28	384.21	391.33	394.39	405.25	412.04	420.47
concrete	9.268	9.257	9.252	9.309	9.114	9.724	9.778
energyCooling	3.364	3.372	3.444	3.514	3.520	3.637	3.705
energyHeating	3.023	3.019	3.026	3.140	3.198	3.321	3.416
keijzer-6	0.054	0.059	0.058	0.062	0.061	0.062	0.061
keijzer-7	0.074	0.071	0.082	0.068	0.073	0.082	0.076
parkinsons	9.965	9.919	9.864	9.849	9.866	9.925	9.979
ppb	24.113	24.328	24.138	24.424	24.294	24.043	23.852
towerData	51.594	51.976	50.931	50.315	52.673	53.257	52.062
vladislavleva-1	0.045	0.048	0.048	0.047	0.047	0.051	0.052
wineRed	0.656	0.657	0.668	0.679	0.693	0.704	0.720
wineWhite	0.756	0.753	0.752	0.764	0.771	0.791	0.796
yacht	3.335	3.279	3.485	3.446	3.855	3.741	3.572

Table A.2: Training RMSE obtained by GP on a training set reduced using the surrounding function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	17.353	18.610	17.114	17.307	16.047	17.747	18.839
ccn	0.146	0.148	0.150	0.151	0.154	0.158	0.162
ccun	385.28	384.23	388.06	397.56	404.71	413.13	416.07
concrete	9.268	9.202	9.322	9.727	9.233	9.296	9.575
energyCooling	3.364	3.404	3.456	3.522	3.590	3.685	3.735
energyHeating	3.023	3.009	3.128	3.145	3.258	3.305	3.371
keijzer-6	0.054	0.056	0.063	0.060	0.054	0.057	0.058
keijzer-7	0.074	0.079	0.076	0.071	0.072	0.070	0.069
parkinsons	9.965	9.960	9.878	9.876	9.931	9.992	10.058
ppb	24.113	24.302	23.991	24.673	24.821	24.539	24.468
towerData	51.594	51.281	51.988	51.868	52.925	52.888	52.064
vladislavleva-1	0.045	0.048	0.050	0.049	0.046	0.052	0.054
wineRed	0.656	0.662	0.665	0.680	0.694	0.712	0.724
wineWhite	0.756	0.754	0.759	0.772	0.786	0.808	0.815
yacht	3.335	3.385	3.762	3.728	3.662	3.952	3.844

Table A.3: Training RMSE obtained by GP on a training set reduced using the remoteness function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	17.353	14.728	13.462	17.156	17.087	18.946	14.654
ccn	0.146	0.148	0.148	0.151	0.154	0.157	0.161
ccun	385.28	383.44	388.36	399.92	406.30	418.50	421.47
concrete	9.268	9.486	9.465	9.665	9.322	9.533	9.795
energyCooling	3.364	3.366	3.414	3.547	3.592	3.707	3.717
energyHeating	3.023	3.045	3.078	3.109	3.187	3.276	3.410
keijzer-6	0.054	0.059	0.058	0.056	0.055	0.055	0.058
keijzer-7	0.074	0.070	0.074	0.078	0.069	0.072	0.071
parkinsons	9.965	9.916	9.875	9.848	9.881	9.900	9.991
ppb	24.113	24.302	24.419	24.524	24.343	24.059	24.311
towerData	51.594	50.363	51.293	51.113	52.042	51.011	52.874
vladislavleva-1	0.045	0.047	0.050	0.047	0.047	0.057	0.050
wineRed	0.656	0.659	0.669	0.678	0.695	0.706	0.717
wineWhite	0.756	0.758	0.761	0.767	0.781	0.797	0.804
yacht	3.335	3.315	3.318	3.709	3.854	3.738	3.875

Table A.4: Training RMSE obtained by GP on a training set reduced using the nonlinearity function. The *ppb* dataset could not be used, since it has more attributes than instances.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	17.353	18.240	20.117	15.952	15.833	14.311	17.142
ccn	0.146	0.147	0.149	0.147	0.151	0.151	0.153
ccun	385.28	380.30	381.29	385.68	386.06	389.35	402.82
concrete	9.268	9.049	9.073	9.301	9.277	9.076	9.264
energyCooling	3.364	3.353	3.383	3.394	3.417	3.418	3.357
energyHeating	3.023	3.043	3.009	2.990	2.997	3.003	2.970
keijzer-6	0.054	0.053	0.053	0.053	0.056	0.053	0.056
keijzer-7	0.074	0.075	0.065	0.075	0.070	0.068	0.072
parkinsons	9.965	9.978	9.981	9.968	9.975	9.951	9.966
towerData	51.594	52.219	51.674	52.108	52.548	52.969	53.244
vladislavleva-1	0.045	0.053	0.049	0.049	0.050	0.050	0.050
wineRed	0.656	0.660	0.657	0.656	0.658	0.662	0.667
wineWhite	0.756	0.754	0.762	0.754	0.751	0.744	0.739
yacht	3.335	3.386	3.021	3.413	3.379	3.576	3.729

Table A.5: Training RMSE obtained by GSGP on a training set reduced using the proximity function. Values highlighted in bold correspond to test RMSE better than the one obtained by GSGP when fed with the complete set of instances, indicating results in which the instance selection had a beneficial impact on the search process.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	26.694	26.746	26.742	26.827	26.745	26.640	26.391
ccn	0.132	0.132	0.134	0.137	0.139	0.141	0.143
ccun	337.94	337.65	342.14	347.31	352.34	357.59	361.36
concrete	6.121	6.175	6.286	6.348	6.445	6.533	6.555
energyCooling	2.260	2.241	2.244	2.320	2.339	2.351	2.367
energyHeating	1.789	1.802	1.851	1.847	1.839	1.887	1.869
keijzer-6	0.011	0.011	0.012	0.012	0.012	0.012	0.013
keijzer-7	0.025	0.024	0.023	0.023	0.027	0.026	0.026
parkinsons	9.717	9.672	9.631	9.615	9.657	9.705	9.763
ppb	10.880	10.759	10.669	10.436	9.816	9.278	8.588
towerData	33.561	33.542	33.673	33.894	33.803	34.386	34.234
vladislavleva-1	0.034	0.034	0.034	0.034	0.034	0.033	0.033
wineRed	0.606	0.607	0.617	0.627	0.636	0.646	0.656
wineWhite	0.707	0.706	0.712	0.720	0.731	0.742	0.755
yacht	6.228	6.192	6.241	6.438	6.504	6.632	6.683

Table A.6: Training RMSE obtained by GSGP on a training set reduced using the surrounding function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	26.694	26.713	26.707	26.679	26.701	26.511	26.529
ccn	0.132	0.133	0.134	0.137	0.139	0.142	0.144
ccun	337.94	338.62	343.48	345.80	351.52	357.46	361.20
concrete	6.121	6.159	6.283	6.276	6.357	6.378	6.477
energyCooling	2.260	2.236	2.247	2.294	2.320	2.355	2.378
energyHeating	1.789	1.784	1.830	1.817	1.880	1.829	1.890
keijzer-6	0.011	0.011	0.012	0.012	0.011	0.010	0.012
keijzer-7	0.025	0.027	0.024	0.024	0.024	0.024	0.022
parkinsons	9.717	9.707	9.662	9.664	9.718	9.775	9.842
ppb	10.880	10.819	10.619	10.268	9.927	9.476	8.880
towerData	33.561	33.651	33.486	33.935	33.839	34.380	34.404
vladislavleva-1	0.034	0.034	0.033	0.034	0.033	0.033	0.032
wineRed	0.606	0.608	0.617	0.628	0.638	0.650	0.663
wineWhite	0.707	0.706	0.716	0.728	0.740	0.753	0.767
yacht	6.228	6.249	6.254	6.368	6.530	6.595	6.623

Table A.7: Training RMSE obtained by GSGP on a training set reduced using the remoteness function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	26.694	26.654	26.738	26.825	26.679	26.659	26.557
ccn	0.132	0.132	0.134	0.137	0.140	0.142	0.144
ccun	337.94	338.88	343.99	347.90	352.13	358.82	362.74
concrete	6.121	6.129	6.264	6.296	6.424	6.480	6.555
energyCooling	2.260	2.212	2.263	2.273	2.339	2.360	2.414
energyHeating	1.789	1.805	1.825	1.844	1.854	1.878	1.879
keijzer-6	0.011	0.011	0.012	0.012	0.012	0.014	0.014
keijzer-7	0.025	0.025	0.023	0.023	0.024	0.025	0.023
parkinsons	9.717	9.678	9.620	9.616	9.664	9.702	9.777
ppb	10.880	10.819	10.535	10.246	10.043	9.193	8.770
towerData	33.561	33.565	33.708	33.912	33.937	34.300	34.482
vladislavleva-1	0.034	0.034	0.033	0.033	0.033	0.033	0.033
wineRed	0.606	0.608	0.616	0.626	0.637	0.647	0.659
wineWhite	0.707	0.706	0.713	0.724	0.736	0.746	0.760
yacht	6.228	6.219	6.278	6.385	6.529	6.572	6.649

Table A.8: Training RMSE obtained by GSGP on a training set reduced using the nonlinearity function. The *ppb* dataset could not be used, since it has more attributes than instances.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	26.694	26.800	26.712	26.709	26.909	26.734	26.741
ccn	0.132	0.132	0.132	0.133	0.136	0.135	0.136
ccun	337.94	337.63	337.27	335.34	335.27	337.06	340.55
concrete	6.121	6.147	6.185	6.168	6.165	6.237	6.117
energyCooling	2.260	2.235	2.227	2.253	2.241	2.200	2.210
energyHeating	1.789	1.865	1.782	1.760	1.774	1.743	1.765
keijzer-6	0.011	0.011	0.013	0.012	0.012	0.012	0.012
keijzer-7	0.025	0.025	0.023	0.024	0.025	0.024	0.028
parkinsons	9.717	9.723	9.732	9.727	9.715	9.673	9.703
towerData	33.561	33.402	33.431	33.385	33.734	33.646	33.755
vladislavleva-1	0.034	0.034	0.033	0.033	0.034	0.034	0.034
wineRed	0.606	0.606	0.603	0.603	0.602	0.604	0.609
wineWhite	0.707	0.706	0.707	0.704	0.704	0.702	0.696
yacht	6.228	6.253	6.270	6.355	6.341	6.370	6.306

Table A.9: Training RMSE obtained by GSGP on a training set embedded using the *t-SNE* method and reduced using the proximity function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	26.694	26.733	26.696	26.719	26.664	26.796	26.678
ccn	0.132	0.133	0.134	0.136	0.138	0.140	0.142
ccun	337.94	338.07	344.41	353.32	359.60	367.76	374.94
concrete	6.121	6.171	6.252	6.308	6.302	6.326	6.395
energyCooling	2.260	2.244	2.249	2.304	2.326	2.309	2.343
energyHeating	1.789	1.803	1.793	1.845	1.843	1.849	1.890
parkinsons	9.717	9.641	9.586	9.677	9.785	9.890	10.000
ppb	11.402	11.406	11.235	10.805	10.687	10.157	9.723
towerData	33.561	33.601	33.853	33.946	33.985	34.444	34.464
wineRed	0.606	0.608	0.618	0.627	0.636	0.647	0.657
wineWhite	0.707	0.706	0.715	0.727	0.737	0.750	0.763
yacht	6.228	6.300	6.244	6.354	6.430	6.434	6.447

Table A.10: Training RMSE obtained by GSGP on a training set embedded using the t -SNE method and reduced using the surrounding function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	26.694	26.747	26.729	26.662	26.397	26.509	26.556
ccn	0.132	0.133	0.135	0.137	0.140	0.142	0.145
ccun	337.94	337.89	343.27	351.06	354.49	364.12	371.03
concrete	6.121	6.135	6.120	6.234	6.316	6.287	6.404
energyCooling	2.260	2.241	2.256	2.270	2.321	2.334	2.351
energyHeating	1.789	1.787	1.822	1.866	1.829	1.880	1.912
parkinsons	9.717	9.717	9.730	9.756	9.832	9.956	10.064
ppb	11.402	11.250	10.983	10.758	10.485	10.080	9.555
towerData	33.561	33.465	33.540	33.993	34.114	34.141	34.274
wineRed	0.606	0.608	0.618	0.629	0.640	0.653	0.665
wineWhite	0.707	0.706	0.716	0.728	0.742	0.758	0.773
yacht	6.228	6.179	6.264	6.326	6.353	6.300	6.338

Table A.11: Training RMSE obtained by GSGP on a training set embedded using the t -SNE method and reduced using the remoteness function.

Dataset	Training instances removed (%)						
	0	1	5	10	15	20	25
airfoil	26.694	26.676	26.688	26.771	26.787	26.580	26.490
ccn	0.132	0.133	0.135	0.137	0.139	0.141	0.143
ccun	337.94	338.18	343.70	352.63	359.80	367.87	375.09
concrete	6.121	6.126	6.204	6.182	6.289	6.371	6.460
energyCooling	2.260	2.223	2.278	2.299	2.314	2.340	2.369
energyHeating	1.789	1.819	1.820	1.835	1.851	1.894	1.912
parkinsons	9.717	9.676	9.561	9.600	9.686	9.785	9.905
ppb	11.402	11.380	11.107	10.854	10.552	10.037	9.581
towerData	33.561	33.708	33.722	33.782	33.937	33.887	34.508
wineRed	0.606	0.609	0.616	0.629	0.640	0.651	0.663
wineWhite	0.707	0.701	0.710	0.719	0.731	0.746	0.762
yacht	6.228	6.217	6.259	6.355	6.369	6.480	6.536

Bibliography

- Albinati, J., Pappa, G. L., Otero, F. E. B., and Oliveira, L. O. V. B. (2015). The effect of distinct geometric semantic crossover operators in regression problems. In *Proc. of EuroGP*, pages 3–15.
- Alsing, R. (2008). Genetic programming: Evolution of mona lisa.
- Arnaiz-González, Á., Blachnik, M., Kordos, M., and García-Osorio, C. (2016). Fusion of instance selection methods in regression tasks. *Information Fusion*, 30:69–79.
- Arnaiz-González, A., Díez-Pastor, J. F., Rodríguez, J. J., and García-Osori, C. (2016). Instance selection for regression: Adapting drop. *Neurocomputing*, 201(Supplement C):66 – 81. ISSN 0925-2312.
- Back, T. and Schwefel, H. P. (1996). Evolutionary computation: an overview. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 20–29. ISSN .
- Banzhaf, W., Francone, F. D., Keller, R. E., and Nordin, P. (1998). *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-510-X.
- Beadle, L. and Johnson, C. G. (2008). Semantically driven crossover in genetic programming. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 111–116. ISSN 1089-778X.
- Borg, I. and Groenen, P. (1997). *Modern Multidimensional Scaling: Theory and Applications*. Springer Series in Statistics. Springer. ISBN 9780387948454.
- Borrelli, A., De Falco, I., Della Cioppa, A., Nicodemi, M., and Trautteur, G. (2006). Performance of genetic programming to extract the trend in noisy data series. *Physica A: Statistical Mechanics and its Applications*, 370(1):104–108.

- Busch, J., Ziegler, J., Aue, C., Ross, A., Sawitzki, D., and Banzhaf, W. (2002). Automatic generation of control programs for walking robots using genetic programming. In *European Conference on Genetic Programming*, pages 258–267. Springer.
- Calvo-Zaragoza, J., Valero-Mas, J. J., and Rico-Juan, J. R. (2015). Improving knn multi-label classification in prototype selection scenarios using class proposals. *Pattern Recognition*, 48(5):1608 – 1622. ISSN 0031-3203.
- Cano, J. R., Herrera, F., and Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study. *IEEE Transactions on Evolutionary Computation*, 7(6):561–575.
- Cardie, C. and Howe, N. (1997). Improving minority class prediction using case-specific feature weights. In *ICML*, pages 57–65.
- Castelli, M., Castaldi, D., Giordani, I., Silva, S., Vanneschi, L., Archetti, F., and Maccagnola, D. (2013a). An efficient implementation of geometric semantic genetic programming for anticoagulation level prediction in pharmacogenetics. In Correia, L., Reis, L. P., and Cascalho, J., editors, *16th Portuguese Conference on Artificial Intelligence, EPIA 2013*, volume 8154 of *LNCS*, pages 78–89. Springer Berlin Heidelberg.
- Castelli, M., Manzoni, L., and Vanneschi, L. (2012). An efficient genetic programming system with geometric semantic operators and its application to human oral bioavailability prediction. *arXiv preprint arXiv:1208.2437*.
- Castelli, M., Silva, S., and Vanneschi, L. (2015a). A C++ framework for geometric semantic genetic programming. *Genetic Prog. and Evolvable Machines*, 16(1):73–81. ISSN 1389-2576.
- Castelli, M., Silva, S., Vanneschi, L., Cabral, A., Vasconcelos, M. J., Catarino, L., and Carreiras, J. M. (2013b). Land cover/land use multiclass classification using GP with geometric semantic operators. In Esparcia-Alcázar, A. I., editor, *16th European Conference, EvoApplications 2013*, volume 7835 of *LNCS*. Springer Berlin Heidelberg.
- Castelli, M., Trujillo, L., and Vanneschi, L. (2015b). Energy consumption forecasting using semantic-based genetic programming with local search optimizer. *Computational Intelligence and Neuroscience*, 2015:8.

- Castelli, M., Trujillo, L., Vanneschi, L., and Popovič, A. (2015c). Prediction of energy performance of residential buildings: A genetic programming approach. *Energy and Buildings*, 102:67 – 74.
- Castelli, M., Vanneschi, L., , and Popovič, A. (2015d). Predicting burned areas of forest fires: An artificial intelligence approach. *Fire Ecology*, 11(1):106–118.
- Castelli, M., Vanneschi, L., and Felice, M. D. (2015e). Forecasting short-term electricity consumption using a semantics-based genetic programming framework: The south italy case. *Energy Economics*, 47:37 – 41.
- Castelli, M., Vanneschi, L., and Silva, S. (2013c). Prediction of high performance concrete strength using genetic programming with geometric semantic genetic operators. *Expert Systems with Applications*, 40(17):6856--6862.
- Chen, Q., Zhang, M., and Xue, B. (2017). Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. *IEEE Transactions on Evolutionary Computation*, 21(5):792–806. ISSN 1089-778X.
- Cover, T. M. and Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21--27.
- De Falco, I., Della Cioppa, A., Maisto, D., Scafuri, U., and Tarantino, E. (2007). Parsimony doesn't mean simplicity: Genetic programming for inductive inference on noisy data. In Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., and Esparcia-Alcázar, A., editors, *Proceedings of the 10th European Conference, EuroGP'07*, volume 4445 of *LNCS*, pages 351--360. Springer.
- Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1--30.
- Domingos, P. (2012). A few useful things to know about machine learning. *Commun. ACM*, 55(10):78--87. ISSN 0001-0782.
- Fitzgerald, J. and Ryan, C. (2014). On size, complexity and generalisation error in gp. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14*, pages 903--910, New York, NY, USA. ACM.
- Fogel, D. B. (2006). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, 3rd Edition*. Wiley. ISBN 978-0-471-66951-7.

- Garcia, S., Derrac, J., Cano, J. R., and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–435.
- Grochowski, M. and Jankowski, N. (2004). Comparison of instance selection algorithms ii. results and comments. In Rutkowski, L., Siekmann, J. H., Tadeusiewicz, R., and Zadeh, L. A., editors, *Proc. of the 7th International Conference on Artificial Intelligence and Soft Computing - ICAISC 2004*, pages 580–585, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Guillen, A., Herrera, L. J., Rubio, G., Pomares, H., Lendasse, A., and Rojas, I. (2010). New method for instance or prototype selection using mutual information in time series prediction. *Neurocomputing*, 73(10):2030–2038.
- Harmeling, S., Dornhege, G., Tax, D., Meinecke, F., and Müller, K.-R. (2006). From outliers to prototypes: Ordering data. *Neurocomput.*, 69(13-15):1608–1618. ISSN 0925-2312.
- Hart, P. (1968). The condensed nearest neighbor rule. *Information Theory, IEEE Transactions on*, 14(3):515–516.
- Hickey, R. J. (1996). Noise modelling and evaluating learning from examples. *Artificial Intelligence*, 82(1):157–179.
- Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *J. Educ. Psych.*, 24.
- Imada, J. H. and Ross, B. J. (2008). Using feature-based fitness evaluation in symbolic regression with added noise. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*, pages 2153–2158. ACM.
- Johnson, C. G. (2007). *Genetic Programming with Fitness Based on Model Checking*, pages 114–124. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. In Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., and Costa, E., editors, *6th European Conference, EuroGP 2003*, volume 2610, pages 70–82. Springer Berlin Heidelberg.
- Kharin, Y. and Zhuk, E. (1994). Robustness in statistical pattern recognition under “contaminations” of training samples. In *Pattern Recognition, 1994. Vol. 2-*

- Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, volume 2, pages 504–506. IEEE.
- Kinnear, K. E. (1994). Fitness landscapes and difficulty in genetic programming. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 142–147 vol.1. ISSN .
- Kordos, M. and Blachnik, M. (2012). Instance selection with neural networks for regression problems. In *Artificial Neural Networks and Machine Learning–ICANN 2012*, pages 263–270. Springer.
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA. ISBN 0-262-11170-5.
- Krawiec, K. (2016). *Semantic Genetic Programming*, pages 55–66. Springer International Publishing, Cham.
- Krawiec, K. and Pawlak, T. (2012). *Quantitative Analysis of Locally Geometric Semantic Crossover*, pages 397–406. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Krawiec, K. and Pawlak, T. (2013). Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. *Genetic Programming and Evolvable Machines*, 14(1):31–63. ISSN 1573-7632.
- Kunimatsu, K., Ishikawa, Y., Takata, M., and Joe, K. (2015). A music composition model with genetic programming-a case study of chord progression and bassline. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, page 256. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).
- Langdon, W. B. (1996). *Genetic programming and data structures*. University College London. Department of Computer Science.
- Langdon, W. B. (2015). Performance of genetic programming optimised bowtie2 on genome comparison and analytic testing (gcat) benchmarks. *BioData mining*, 8(1):1.
- Leyva, E., González, A., and Perez, R. (2015a). A set of complexity measures designed for applying meta-learning to instance selection. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):354–367.

- Leyva, E., González, A., and Pérez, R. (2015b). Three new instance selection methods based on local sets: A comparative study with several approaches from a bi-objective perspective. *Pattern Recognition*, 48(4):1523 – 1537. ISSN 0031-3203.
- Lichman, M. (2015). UCI mach. learning repository.
- Liu, L., Shao, L., Li, X., and Lu, K. (2016). Learning spatio-temporal representations for action recognition: A genetic programming approach. *IEEE transactions on cybernetics*, 46(1):158--170.
- Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579--2605.
- McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaskowski, W., Krawiec, K., Harper, R., De Jong, K., and O'Reilly, U.-M. (2012). Genetic programming needs better benchmarks. In *Proc. of GECCO*, pages 791--798.
- Miranda, L. F., Oliveira, L. O. V. B., Martins, J. F. B. S., and Pappa, G. L. (2017). How noisy data affects geometric semantic genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17*, pages 985--992, New York, NY, USA. ACM.
- Moraglio, A. (2011). Abstract convex evolutionary search. In *Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms, FOGA '11*, pages 151--162, New York, NY, USA. ACM.
- Moraglio, A., Krawiec, K., and Johnson, C. G. (2012). *Geometric Semantic Genetic Programming*, pages 21--31. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Nettleton, D. F., Orriols-Puig, A., and Fornells, A. (2010). A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306. ISSN 0269-2821.
- Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2014). Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation*, 18(2):193--208.
- Ni, J., Driberg, R. H., and Rockett, P. I. (2013). The use of an analytic quotient operator in genetic programming. *Evolut. Computation, IEEE Trans. on*, 17(1):146--152.

- Oliveira, L. O. V. B. (2016). *Improving Search in Geometric Semantic Genetic Programming*. PhD thesis, Universidade Federal de Minas Gerais.
- Oliveira, L. O. V. B., Miranda, L. F., Pappa, G. L., Otero, F. E. B., and Takahashi, R. H. C. (2016). Reducing dimensionality to improve search in semantic genetic programming. In Handl, J., Hart, E., Lewis, P. R., López-Ibáñez, M., Ochoa, G., and Paechter, B., editors, *Proc. of the 14th International Conference on Parallel Problem Solving from Nature (PPSN XIV)*, volume 9921 of *LNCS*, pages 375–385. Springer International Publishing.
- Olvera-López, J. A., Carrasco-Ochoa, J. A., Martínez-Trinidad, J. F., and Kittler, J. (2010). A review of instance selection methods. *Artificial Intelligence Review*, 34(2):133–143. ISSN 1573-7462.
- Pawlak, T. P. (2015). Competent algorithms for geometric semantic genetic programming. *review. PhD thesis. Poznań, Poland: Poznan University of Technology*.
- Peteiro-Barral, D. and Guijarro-Berdiñas, B. (2013). A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11. ISSN 2192-6360.
- Poli, R. (2005). Tournament selection, iterated coupon-collection problem, and backward-chaining evolutionary algorithms. *Foundations of Genetic Algorithms*, pages 382–387.
- Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd.
- R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
- Raidl, G. (2005). Evolutionary computation: An overview and recent trends. In *OGAI Journal (Oesterreichische Gesellschaft fuer Artificial Intelligence)*. Citeseer.
- Ritchie, M. D., White, B. C., Parker, J. S., Hahn, L. W., and Moore, J. H. (2003). Optimization of neural network architecture using genetic programming improves detection and modeling of gene-gene interactions in studies of human diseases. *BMC bioinformatics*, 4(1):28.
- Rodriguez-Fdez, I., Mucientes, M., and Bugarin, A. (2013). An instance selection algorithm for regression and its application in variance reduction. In *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on*, pages 1–8. IEEE.

- Rousseeuw, P. J. and Leroy, A. M. (2005). *Robust regression and outlier detection*, volume 589. John Wiley & sons.
- Sáez, J. A., Luengo, J., and Herrera, F. (2011). Fuzzy rule based classification systems versus crisp robust learners trained in presence of class noise’s effects: a case of study. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 1229–1234. IEEE.
- Sáez, J. A., Luengo, J., and Herrera, F. (2016). Evaluating the classifier behavior with noisy data considering performance and robustness: the equalized loss of accuracy measure. *Neurocomputing*, 176:26–35. ISSN 0925-2312.
- Sivapragasam, C., Vincent, P., and Vasudevan, G. (2007). Genetic programming model for forecast of short and noisy data. *Hydrological processes*, 21(2):266–272.
- Spears, W. M., Jong, K. A. D., Bäck, T., Fogel, D. B., and Garis, H. d. (1993). An overview of evolutionary computation. In *Proceedings of the European Conference on Machine Learning, ECML ’93*, pages 442–459, London, UK, UK. Springer-Verlag.
- Tenenbaum, J. B., Silva, V. d., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323. ISSN 0036-8075.
- Torgerson, W. S. (1952). Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419. ISSN 1860-0980.
- Vanneschi, L. (2014). Improving genetic programming for the prediction of pharmacokinetic parameters. *Memetic Computing*, 6(4):255–262.
- Vanneschi, L., Castelli, M., Costa, E., Re, A., Vaz, H., Lobo, V., and Urbano, P. (2015). Improving maritime awareness with semantic genetic programming and linear scaling: Prediction of vessels position based on ais data. In Mora, A. M. and Squillero, G., editors, *18th European Conference, EvoApplications 2015.*, volume 9028 of *LNCS*, pages 732–744. Springer International Publishing.
- Vanneschi, L., Castelli, M., Manzoni, L., and Silva, S. (2013). A new implementation of geometric semantic gp and its application to problems in pharmacokinetics. In Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A. c., and Hu, B., editors, *16th European Conference, EuroGP 2013*, volume 7831 of *LNCS*, pages 205–216. Springer Berlin Heidelberg.

- Vanneschi, L., Castelli, M., and Silva, S. (2010). Measuring bloat, overfitting and functional complexity in genetic programming. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 877–884, New York, NY, USA. ACM.
- Vanneschi, L., Castelli, M., and Silva, S. (2014a). A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, 15(2):195–214. ISSN 1573-7632.
- Vanneschi, L., Silva, S., Castelli, M., and Manzoni, L. (2014b). Geometric semantic genetic programming for real life applications. In Riolo, R., Moore, J. H., and Kotanchek, M., editors, *Genetic Programming Theory and Practice XI*, Genetic and Evolutionary Computation, pages 191–209. Springer New York.
- Vladislavleva, E., Smits, G., and den Hertog, D. (2010). On the importance of data balancing for symbolic regression. *IEEE Trans. Evolutionary Computation*, 14(2):252–277.
- Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, pages 408–421.
- Wilson, D. R. and Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286.