NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Ensemble Learning with GSGP

Olivier GAU

Dissertation presented as partial requirement for obtaining the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

ENSEMBLE LEARNING WITH GSGP

por

Olivier GAU

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Métodos Analíticos Avançados

Orientador: Leonardo Vanneschi

Fevereiro 2020

Ensemble Learning with GSGP

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2]. [1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

I would like to thank Professor Leonardo Vanneschi, Professor Mauro Castelli and Assistant Professor Illya Bakurov for their guidance and help during this project.

Abstract

The purpose of this thesis is to conduct comparative research between Genetic Programming (GP) and Geometric Semantic Genetic Programming (GSGP), with different initialization (RHH and EDDA) and selection (Tournament and Epsilon-Lexicase) strategies, in the context of a model-ensemble in order to solve regression optimization problems.

A model-ensemble is a combination of base learners used in different ways to solve a problem. The most common ensemble is the mean, where the base learners are combined in a linear fashion, all having the same weights. However, more sophisticated ensembles can be inferred, providing higher generalization ability.

GSGP is a variant of GP using different genetic operators. No previous research has been conducted to see if GSGP can perform better than GP in model-ensemble learning. The evolutionary process of GP and GSGP should allow us to learn about the strength of each of those base models to provide a more accurate and robust solution. The base-models used for this analysis were Linear Regression, Random Forest, Support Vector Machine and Multi-Layer Perceptron. This analysis has been conducted using 7 different optimization problems and 4 real-world datasets. The results obtained with GSGP are statistically significantly better than GP for most cases.

Keywords: GP, GSGP, Model-ensemble, Linear Regression, Random Forest, Support Vector Machine, Multi-Layer Perceptron

Resumo

O objetivo desta tese é realizar pesquisas comparativas entre Programação Genética (GP) e Programação Genética Semântica Geométrica (GSGP), com diferentes estratégias de inicialização (RHH e EDDA) e seleção (Tournament e Epsilon-Lexicase), no contexto de um conjunto de modelos, a fim de resolver problemas de otimização de regressão.

Um conjunto de modelos é uma combinação de alunos de base usados de diferentes maneiras para resolver um problema. O conjunto mais comum é a média, na qual os alunos da base são combinados de maneira linear, todos com os mesmos pesos. No entanto, conjuntos mais sofisticados podem ser inferidos, proporcionando maior capacidade de generalização.

O GSGP é uma variante do GP usando diferentes operadores genéticos. Nenhuma pesquisa anterior foi realizada para verificar se o GSGP pode ter um desempenho melhor que o GP no aprendizado de modelos. O processo evolutivo do GP e GSGP deve permitir-nos aprender sobre a força de cada um desses modelos de base para fornecer uma solução mais precisa e robusta. Os modelos de base utilizados para esta análise foram: Regressão Linear, Floresta Aleatória, Máquina de Vetor de Suporte e Perceptron de Camadas Múltiplas. Essa análise foi realizada usando 7 problemas de otimização diferentes e 4 conjuntos de dados do mundo real. Os resultados obtidos com o GSGP são estatisticamente significativamente melhores que o GP na maioria dos casos.

Palavras-chave: GP, GSGP, Model-ensemble, Linear Regression, Random Forest, Support Vector Machine, Multi-Layer Perceptron

Contents

List of Figures

List of Tables

Listings

Acronyms

ATA	Automatic Threshold Adaptation.
BL	Base Learner.
EDV	Error Deviation Variation.
EDV-SCC	Error Deviation Variation Semantic Stopping.
GP	Standard Genetic Programming.
GSC	Geometric Semantic Crossover.
GSGP	Geometric Semantic Genetic Programming.
GSM	Geometric Semantic Mutation.
GSO	Geometric Semantic Operators.
LS	Lexicase Selection.
not tuned BLs	Base Learners whose hyper-parameters were not estimated, instead kept default.
SC	Stopping Criteria.
S-GSGP	Stacking with GSGP.
S-SGP	Stacking with standard GP.

S-GP	Stacking with GP-like techniques (any).
TIE	Training Improvement Effectiveness.
TIE-SCC	Training Improvement Effectiveness Semantic Stopping.
TS	Tournament Selection.
tuned BLs	Base Learners whose hyper-parameters were estimated from Grid-Search Cross Validation.

Chapter 1

Introduction

Ensemble Learning (EL) is a sub-field in Machine Learning which is inspired on humans' natural tendency to seek and weight the opinion of others' before making any important decision. Under this perspective, EL consists of combining several individual models, the base-learners, in a way to produce a model (the ensemble), which is expected to solve a given task better than any of the base-learners [43]. Stacked Generalization, or simply stacking, consists of training an ensemble from the combined predictions of several other, ideally heterogeneous, base-learners. More specifically, it consists of training the base-learners to solve the underlying task, then it trains a meta-learner from their predictions [55].

In this paper, we study Genetic Programming (GP) as the meta-learner which combines four heterogeneous base-learners. More technically, we allow GP to automatically evolve computer programs having as a terminal set the combined predictions of four heterogeneous base-learners. The objective which drives such an evolutionary process is ensemble's generalization ability on a given Supervised Machine Learning (SML) task. For the sake of simplicity, we will refer to this kind of approaches as Stacking-GP (S-GP). Our motivation relies upon intrinsic properties of GP. We expect that, with *properly* chosen operators and hyper-parameters, GP is capable of combining base-learners in a highly non-linear fashion which could better exploit their outputs and achieve superior generalization ability.

In fact, the usage of S-GP is not new in the research-field. To our knowledge, the first work dates to 2006 [26] where GP was used to combine ten Artificial Neural Networks into an ensemble. Since then, several other important contributions were proposed [1, 12, 20, 29]. Nevertheless, we consider that research in S-GP is still much in demand and we have identified two major reasons for that. First, we have found that the majority of contributions in S-GP are assessed on classification problems, whereas

none of the previous works provides a concise benchmark over regression problems. The second reason has to do with the recent methodological achievements of the in the research field: Geometric-Semantic Operators (GSO), Semantic Stopping Criterion (SSC), ϵ-Lexicase Selection (ϵ-LS) and Evolutionary Demes Despeciation Algorithm (EDDA) are among numerous recent methods which were not broadly studied in the context of S-GP. In the objective of this paper consists of covering aforementioned gaps by providing a concise overview of S-GP over 7 synthetic and 4 real-world regression problems while using state-of-the art achievements in the field of GP.

This thesis is organized as follows: in Chapter 2 we introduce the necessary theoretical background. Chapter 3 describes the research hypothesis and the proposed approach. Chapter 4 presents our experimental framework, results' exhibition, their detailed analysis. A critical discussion is made in Chapter 5. Finally, Chapter 6 concludes the work and proposes ideas for future research.

Chapter 2

Theory

2.1 Machine Learning

Machine Learning (ML) is a field of artificial intelligence using algorithms and computation power to learn patterns from a given sample of observations called a dataset, without human interaction. An algorithm is a function composed of sets of rules in charge of the learning process using a dataset as input data.

The data can be provided in a tabular form composed of instances and features, corresponding respectively to the rows and columns in a worksheet. This structure can contain an expected value (target) for each set instance (sample) of the dataset. To transform data (sample and target) into knowledge, recognizable elements have to be detected in order to give a prediction for an unseen instance.

If there is no target value the analysis of the data will be unsupervised (e.g. clustering, dimension reduction, and association).

If there is a target the algorithm can adjust its set of rules to by learning from the comparison of the result to the target then we talking about supervised learning. When the target value is a category or numerical, the algorithm's task will be a classification for the former or regression for the latter. In this thesis, we will focus on regression.

2.2 Ensemble Learning

In the field of Machine Learning, one can find numerous methodologies which mimic biological and social processes of the real world. The Artificial Neural Network (ANN), for example, is a biologically-inspired computer system which simulates the Biological Neural Network (BNN) and its bio-chemical processes [25]. As a result, an ANN consists of a set of interconnected layers of neurons (the basic processing units) which, all together, form a powerful and versatile computer system able to solve numerous complex optimization problems such as automatic speech and image recognition, natural language processing, bio-informatics, fraud-detection, etc [46]. The Particle Swarm Optimization (PSO) [19] is another type of biological system - a social system - whose computational metaphor is the collective behavior of simple individuals interacting with their environment and each other, like a flock of birds or schools of fishes. The individual in this algorithm is a particle, which is described by its position and velocity in the search space. The direction in which each particle moves is influenced by the behaviour of the other particles in the swarm - the social factor - and individual memory - the cognitive factor. Such computer system is frequently used in function optimization, ANNs training, fuzzy system control and other areas.

Ensemble Learning (EL), in this context, is not an exception as it reflects the natural predisposition of human beings to seek for other opinions before making any important decision. We tend to weigh several individual opinions, and combine them through some thought process in order to reach a final decision, which is expected to be the most appropriate [43]. In this sense, an ensemble combines several individual models, the base-learners (BLs), in a way which yields a model whose overall generalization ability is expected to be *better* when compared to any of its BLs. In fact, there is a considerable number of empirical studies showing that in both classification and regression problems the ensembles are significantly *superior* than the BLs which compose them [7, 10, 11, 18, 21, 55], and several theoretical explanations have been proposed to justify their superiority [55], majority of them based on the so called *bias-variance decomposition of the error*. Apart from that, when provided a spectrum of SML techniques, it is true that the most appropriate is frequently chosen based on some global approximation measure such as the Root Mean Squared Error (RMSE). Considering a SML problem and two candidate modelling techniques, a and b, it might happen that a, from the perspective of some global performance measure, provides *better* global approximation on a given problem than b, however b might exhibit *better* local approximation than a, i.e., might approximate better than a in a specific region of the search space. In this context, by wisely using both a and b in an ensemble, one can combine the most well-approximated regions of the search space allowing for levels of global approximation which are *superior* to those provided by a and b when taken singularly. Another relevant motivation for using ensembles is the difficulty the one

faces when choosing the most appropriate modeling technique when solving a given problem. In fact, the abundance of conceptually distinct SML techniques, along with a varied number of adjustable parameters they bring, makes the SML taks *laborious* and *time-consuming*. Under this perspective, using an ensemble can substantially reduce this *complexity* to a limited set of models and/or parameters.

2.2.1 Stacked Generalization

The field of EL, although relatively recent, has been extensively researched and several conceptually different approaches, along with different ways to categorize them, were proposed so far. In general terms, Ensemble Methods (EMs) differ in the way input data is represented and manipulated within the system, the procedure to make the final prediction, whether ensemble's BLs can be trained independently from each other, etc. [43, 45, 56].

In this sub-sub-section we will provide an overview of stacked generalization, proposed by David H. Wolpert [55] in 1992, because the approach we propose in this paper strongly relates to this specific ensemble method. Stacked generalization, or simply *stacking*, consists of training a meta-learner from the combined predictions of two (or more) BLs. In other words, the predictions obtained from the BLs, which are trained independently from each other, are used as inputs of a meta-learner which can be any known ML technique. In practice, the system can have several sequential layers of BLs before using one final meta-learner. In figure 2.1 you can find a visual representation of stacking consisting of one layer of five BLs.

The approach we present in this paper has a simple although rational motivation - in the context of stacking, to use GP as the meta-learner from combined predictions of several conceptually different BLs.

2.3 Regression Estimators

In statistics, one or many independent variables also known as features are used to describe one dependent variable also known as a target. Regression estimators are statistical methods (algorithms) which allow us to model either a linear or a non linear relationship between independent and dependent variables.

In a linear relationship, the proportion between the dependent and independent variables will remain the same, the plot resulting in a flat line. In a nonlinear relationship, the logic between the dependent and independent variable is unstable, the plot resulting in a curve.

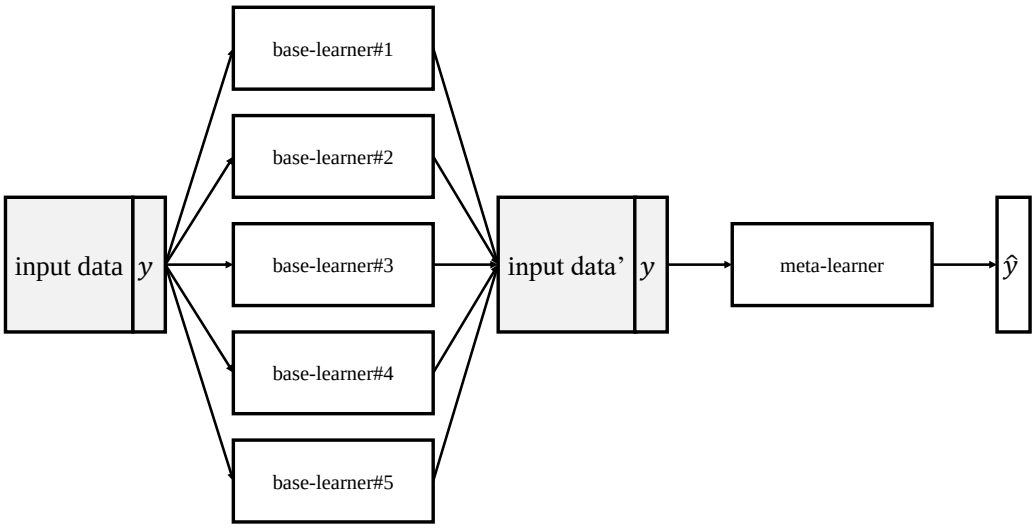

Figure 2.1: Example of a Stacked Generalization consisting of 1 layer of five base-learners.

2.3.1 Multiple Linear Regression

In supervised machine learning, singular linear regression is a popular method to find the relationship between one independent variable (the feature) and one dependent variable (the target). This relationship between variables can be represented by the equation Y = a + b X with Y the dependent variable and X the independent variable. The result of this analysis is a predictive model fitting the observed features and corresponding target.

Multiple linear regression works similarly but is using several independent variables, attributing a slope coefficient to each of them, determining the effect each of them will have on the dependent variable.

2.3.2 Random Forest Regression

Decision Tree is a predictive model that can be use for classification and regression. The model uses a tree-like structure composed of decision nodes, branches and leaf nodes. The decision nodes represent tests with different possibilities represented by branches. A branch can be followed by a decision node or a leaf node. The leaf node corresponds to the final outcome. The solution allows to understand visually the different predictions of the model following a clear logic (tests -> possibilities -> outcomes) based on the given dataset.

The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base learners. More formally we can write this class of models as: g(x)=f0(x)+f1(x)+f2(x)+... where the final model g is the sum of simple base learners. Here, each base learner is a simple decision tree. This broad technique of using multiple models to obtain better predictive performance is called model ensembling. In random forests, all the base learners are constructed independently using a different sub-sample of data.

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. In the Random Forest method, Bagging involves training each decision tree on different data samples (sampling with replacement).

2.3.3 Support Vector Machine Regression

Support Vector Machine (SVM) [9] is an algorithm mainly used for classification purposes where provides good accuracy with less computational power. The goal of SVM for classification is to find the hyperplane in an n-dimensional space able to separate classes with the widest margin possible. The support vectors are data points located at the border of the margin.

SVM can also be used for regression problems, but in this case, it will produce hyperplanes in n-dimensional space whereas many data points can fit within the margin of tolerance 'epsilon'. Only the data points outside the margin 'epsilon' will be used to calculate the error (distance from the border of the margin 'epsilon'). The hyperplane with the lowest error will be selected.

2.3.4 Multilayer Perceptron Regression

An Artificial Neural Network (ANN) is an algorithm mimicking the structure of the human brain. It is composed of connected artificial neurons, also known as nodes, which are mathematical functions similar to biological neurons in their process. A node can have multiple inputs and outputs as a result of its operation.

A Multi-Layer Perceptron (MLP) is the first and most simple of feedforward type of ANN, its structure consisting of at least three layers of nodes.

An input layer is in charge of carrying the original data to the network. One or more hidden layers placed between the input and output layer are computing the weights attributed to each node in the network. Finally, the output layer, which is the final layer of the network is producing the output result. Except for the input nodes,

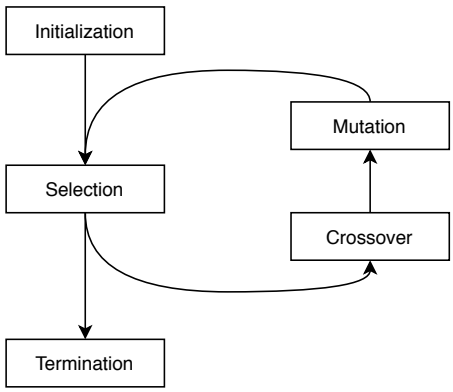

Figure 2.2: Evolutionary Algorithm

each node uses a nonlinear activation function that is able to solve complex problems, taking a node's output as input and outputting its interpretation.

2.4 Evolutionary Algorithm

This particular branch of data science relies heavily on the biological principles governing the natural world. In a nutshell, deoxyribonucleic acid or DNA for short is a molecule containing the genetic code of all organisms on Earth. Each cell in each organism contains these bespoke instructions for which proteins should be made by which cell. It's the reason why parents and children share certain physical traits. The passing of the genes is called heredity and a gene is its basic unit. Humans are a unique blend of their parent's genetic material which is joined through the process of recombination after a reproductive event. A mutation is what we call a change occurring in our DNA sequence due to internal (e.g. DNA copying fault) or external (e.g. UV light, cigarettes) factors without reproduction. In biology, natural selection means that certain human traits/genes are preferable to others and evolution is more likely to preserve them in our genetic material over generations to ensure survival by picking the parents with the most beneficial features for reproduction.

In data science however, Evolutionary Algorithm (Figures 2.2 and 2.3) is a subset of evolutionary computation, based on population and optimization, inspired by biology or more specifically it's sub-field of genetics. By mimicking processes such as reproduction, mutation, recombination, and selection it is able to find a solution within given limitations. The process follows this pseudo code, the steps of which we will discuss later on.

2.4.1 Genetic Programming

In the field of Evolutionary Algorithms (EAs), Genetic Programming (GP) is among the most recent and dynamically growing sub-fields. Introduced and popularized by

1. Create Initial population;
2. Calculate Fitness of all individuals;
3. While Termination condition not met:
    a) Select fitter individuals for reproduction;
    b) Recombine between individuals;
    c) Mutate individuals;
    d) Evaluate fitness of all individuals;
    e) Generate a new population;
4. Return Best individual;

Figure 2.3: Pseudo-code for a simple version of Evolutionary algorithm.

John Koza [30–34], GP is, in fact, an adaption of Genetic Algorithm (GA) for evolution of computer programs. In simple terms, GP is a population-based algorithm which follows principles of *Darwin's Theory of Evolution* [17] to evolve computer programs, among the space of all possible computer programs, which can solve a given optimization problem. The figure 2.4 contains the simple version of the pseudo-code of GP:

1. generate an initial population P of N individuals by means of an initialization method;
2. **repeat** until satisfying some termination condition:
    a) evaluate the fitness $\forall\ i \in P$;
    b) create empty population P';
    c) **repeat** until P contains n individuals:
        i. choose a genetic operator: crossover or mutation with probability p_c or $1 - p_m$, respectively;
        ii. by means of a selection method, select two individuals from P if crossover was chosen, otherwise select one;
        iii. apply the variation operator chosen in point 2.3.1 to the individual(s) selected in point 2.3.2;
        iv. insert the individual(s) obtained in point 2.3.3 into P';
    d) replace P with P';
3. return the best individual

Figure 2.4: Pseudo-code for a simple version of Genetic Programming algorithm.

As we can see from 2.4, evolution of the population starts with individuals' initialization. Then, by applying selection mechanism and variation operators to the selected *parents*, *offspring* are created and transited to the next generation. This process iterates until reaching certain stopping criteria (like the maximum number of generations).

In GP, individuals are computer programs composed by specific elements of a

given programming language arranged in a particular way. Commonly, individuals are represented in a tree-based structure. For the sake of example, consider the following two sets of program elements, necessary components of a computer program: $terminals = \{x1,\ x2,\ x3\}$ and $functions = \{+,\ -,\ *,\ /\}$. A possible individual resulting from composition of such elements is represented in figure 2.5.

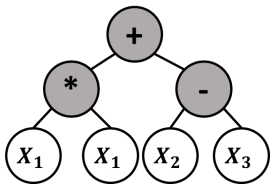

Figure 2.5: Example of a tree-based representation of a GP individual

In other words, an individual evolved by means of GP can be a mathematical function of the form $f(x1, x2, x3) = X_1 * X_1 + X_2 - X_3$.

2.4.2 Geometric Semantic Genetic Programming

In the terminology adopted by a considerable part of Genetic Programming (GP) research community [8, 27, 28, 35, 39], the term *semantics* defines the vector of output values of a candidate solution, calculated on the training observations. Following this definition, a candidate solution obtained by means of GP can be seen as a point in multidimensional space of dimensionality *number of observations in the training set*. Let's call it *semantic space*.

Geometric Semantic Genetic Programming (GSGP) is a recently introduced variant of GP where standard crossover and mutation operators are replaced by the so-called *Geometric Semantic Operators* (GSOs) [38]. GSOs, gained popularity in the GP community [13, 15, 16, 50–53] because of their geometric property of inducing a unimodal error surface (characterized by the absence of locally optimal solutions) for any Supervised Machine Learning (SML) problem. The proof of this property can be found in [38, 50]. In this document, we report the definition of the GSOs, as given by Moraglio et al. for real functions domains, since these are the operators that we have used in our experiments. For applications that consider other types of data, the reader is referred to [38] and [3].

Geometric Semantic Crossover (GSC) generates, as the unique offspring of parents $T_1, T_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, the expression: $T_{XO} = (T_1 \cdot T_R) + ((1 - T_R) \cdot T_2)$, where T_R is a random real function whose output values range in the interval $[0, 1]$. Moraglio and co-authors show that GSC corresponds to geometric crossover in the semantic space, i.e., the point representing the offspring stands on the segment joining the points representing the

parents. Consequently, the GSC inherits the key property of geometric crossover: the offspring is never worse than the worst of the parents.

Geometric Semantic Mutation (GSM) returns, as the result of the mutation of an individual $T : \mathbb{R}^n \to \mathbb{R}$, the expression: $T_M = T + ms \cdot (T_{R1} - T_{R2})$, where T_{R1} and T_{R2} are random real functions with codomain in $[0, 1]$ and ms is a parameter called the mutation step. Similarly to GSC, Moraglio and co-authors show that GSM corresponds to the box mutation on semantic space. Consequently, the operator induces a unimodal error surface on any SML problem.

As Moraglio and co-authors point out, GSOs create an offspring which is substantially larger than their parents, and the fast growth of the individuals' size rapidly makes fitness evaluation very slow, making the system unusable. As a solution to this problem, Castelli et al. [13] proposed a computationally efficient implementation of Moraglio's operators making them usable in practice.

Given fact GSC generates an offspring whose semantics stands on the segment joining the semantics of the two parents, it can only achieve the global optimum if the semantics of the individuals in the population "surround" the semantics of the global optimum. Using the terminology of [14, 41], GSC only has the possibility of generating a globally optimal solution if it lays within the semantic *convex hull* identified by the population. The need for overcoming this drawback has led to several methods to properly initialize a population of GSGP, like for instance the ones presented in [2, 40, 54].

2.4.3 Initialization

In any Evolutionary Algorithm (EA), population initialization is the very first step in the evolutionary process [17]. Assuming a tree-based representation, the initialization of individuals in GP consists of creating almost random trees, such that program elements, starting from the root node of the tree, are combined one after another in a specific manner, until reaching a pre-defined tree depth (d). In his work, John Koza described three initialization methods: Grow, Full and Ramped Half-and-Half (RHH) [34]. In this experimentation, RHH or EDDA are used.

Following the example related to figure 2.5, let's consider a tree-based representation of individuals and a program set divided in two semantically distinct classes: terminals (T) and functions (F).

Grow Initialization The procedure starts with random selection of a node from F as the root node of the tree, in order to avoid trees composed by one single terminal.

Then nodes are selected with uniform probability regardless the set they belong to, until reaching maximum depth d. Once a given branch contains a terminal node, it is ended even if d has not been reached. Finally, in order to trim the tree at d, the nodes of remaining branches are chosen at random exclusively from the set T. By allowing selection of nodes regardless the set, trees are likely to have irregular shape, i.e. to contain branches of different lengths.

Full Initialization Unlike Grow, the Full method chooses nodes only from F until the tree achieves maximum depth d. After reaching d, it chooses nodes at random only from the set T. The result is that every branch of the tree goes to the full maximum depth, which results in *bushy* trees of regular shape.

2.4.3.1 Ramped Half-and-Half

John Koza pointed that population initialized with Grow or Full methods produces too similar trees, which floors the diversity in GP populations [34]. Correspondingly, authors in [42] highlight methods' sensibility towards sizes of the function and terminal sets; as they exemplify, if, the set of program elements has significantly more terminals than functions, grow method will almost always generate very short trees regardless of the depth limit.

In order to overcome the drawbacks of previously introduced initialization methods, John Koza proposed a combination of both called Ramped Half-and-Half (RHH). The RHH method is summarized my means of pseudo-code presented in figure 2.6.

Let d be the maximum depth parameter and P the population size:

1. divide P in d groups;
2. in each group (g_i), set distinct maximum depth equal to 1, 2, (...), $d-1$, d;
3. $for(i = 1; c <= n; c++)$:
   a) initialize one half of group g_i with Full method;
   b) initialize one half of group g_i with Grow method;

Figure 2.6: Pseudo-code for Ramped Half-and-Half initialization method.

From figure 2.7, the one can visually perceive how RHH works for $d = 3$ and $P = 6$. In the figure, the individuals represented in red were initialized by means of Full method, whereas those in blue by Grow method.

2.4.3.2 Evolutionary Demes Despeciation Algorithm

Initialization is known to play an important role in any population-based algorithm. In GP, this aspect plays particular importance since a wide variety of programs of various sizes and shapes is desirable [34, 42]. With the introduction of GSOs, new

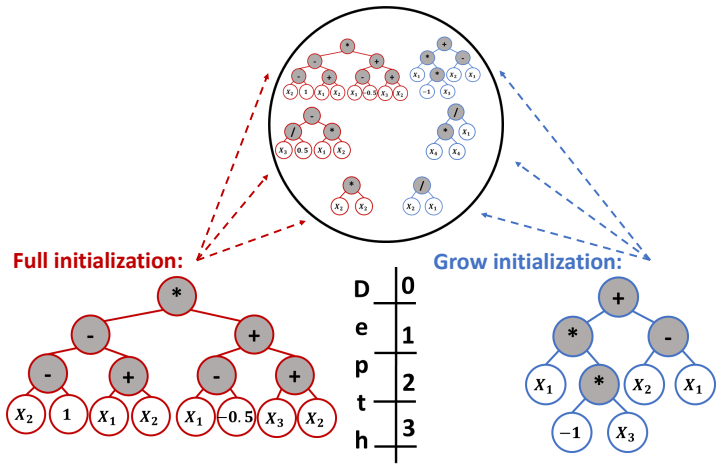

Figure 2.7: Example of functioning of RHH for a population of size 6 and maximum depth of 3.

techniques which take in consideration their particularities, have been developed [2, 40, 54]. In this subsection we will focus on one of these contributions, the Evolutionary Demes Despeciation Algorithm (EDDA) [54], since it is the initialization technique used in our experiments.

In Biology, demes are independent populations, or sub-populations, of individuals that actively interbreed and mature. The term despeciation indicates the combination of demes of previously distinct species into a new population, where distinct biological lineage is blended. Although in Nature the despeciation phenomenon is rare to happen, when it does, it is known to *reinforce* the population making it more competitive.

In EDDA, the initial population of GSGP is generated using the best individuals obtained from a set of independent sub-populations (demes), left to evolve for *few* generations and under different evolutionary conditions [54]. For example, some demes use standard GP operators, while the remaining use GSOs. Besides that, each deme is being evolved under distinct search parameters such as the mutation and crossover probability, the mutation step (in the case of GSM), etc.

Although EDDA was introduced in the GP community recently, it was successfully applied when solving several fundamentally distinct problems [4–6]. GSGP using EDDA demonstrated its superiority over GP initialized with traditional Ramped Half-and-Half (RHH) [34] method over six complex symbolic regression applications [54]. More specifically, on all problems, EDDA allowed for generation of solutions with better or comparable generalization ability and of significantly smaller size than using RHH. In [5, 6] EDDA demonstrated its utility when evolving PSO-based search

rules in unknown vector field whereas in [4] it was used to support medical decisions in the field of rare diseases.

The performance of EDDA depends on two main parameters: the proportion of GSGP demes in the system (n) and the number of generations to evolve each deme (m). Using algorithm-specific notation, given two natural numbers n and m, where $n \in [0,100]$, $EDDA_m - n\%$ represents a system where demes are left to evolve for m generations such that $n\%$ of the population was initialized using individuals from GP demes, while the remaining $(100-n)\%$ was initialized using standard GP demes. The pseudo-code in Figure 2.8 explains the process.

$EDDA_m - n\%$:

1. Create an empty population P of size N;
2. Repeat $N * (n/100)$ times:
   a) Create an empty deme;
   b) Randomly initialize this deme using a classical initialization algorithm (RHH used here);
   c) Evolve individuals from 2.b) for m generations using GSGP;
   d) After finishing 2.c), select the best individual from the deme and store it in P;
3. Repeat $N * (1 - n/100)$ times:
   a) Create an empty deme;
   b) Randomly initialize this deme using a classical initialization algorithm (RHH used here);
   c) Evolve individuals from 3.b) for m generations using standard GP;
   d) After finishing 3.c), select the best individual from the deme and store it in P;
4. Retrieve P and use it as the initial population of GP

Figure 2.8: Pseudo-code of $EDDA_m - n\%$ system, in which demes are left to evolve for m generations.

In the pseudo-code of Figure 2.8, points 2.b), 2.c), 3.b) and 3.c) implement the phase of *demes evolution*, such that different demes are left to evolve in an independent manner. Points 2.d) and 3.d) implement the phase of *despeciation* where individuals, coming from different demes and thus from different evolutionary *journeys* and histories, are blended into a new population (P in the pseudo-code). To evolve P, GSGP is preferred over standard GP as the later is known to outperform the former in several application domains [50, 53]. In order to confirm this evidence, in this study, after *despeciation* phase, we have compared the performance of S-GP and S-GSGP systems to conduct the main evolutionary process (MEP).

The rationale behind EDDA system is that it should generate an initial population composed of diverse and, at the same time, good quality genetic material. In fact, each

individual in the initial GSGP population comes from a different evolution history, performed in an independent deme and evolved under different search parameters. Given that each individual in the initial GSGP population was the best individual in its deme, good quality is expected.

2.4.4 Parent Selection

The parent selection is the mechanism allowing only the individuals with the best features for a given problem to become parents and to produce offspring with certain inherited traits. In this experimentation, the Tournament and ϵ-Lexicase methods were used.

2.4.4.1 Tournament

In Genetic Programming, Tournament is the most popular selection method. First, a defined number (Tournament size) of individuals in the population is randomly selected. Then, only the individuals with the best fitness in this intermediary group are picked to become parents. The genetic operator crossover for instance, uses 2 parents, so 2 tournaments need to be done. The selection pressure is the ratio between the number of individuals in the population and the number of individuals randomly selected by the tournament method. It measures the chance of any individual to participate in the tournament.

2.4.4.2 ϵ-Lexicase Selection for Regression

ϵ-Lexicase Selection (ϵ-LS) [36] is a recently introduced improvement upon already existing Lexicase Selection (LS) of parents in GP [47]. The latter was proposed by Lee Spector in 2012 to provide a simple, problem and representation-independent way to solve multimodal problems without interfering with other components of a GP system. In simple terms, LS is an iterative procedure which consist of the following steps:

Let d_i be the i-th data instance (a.k.a. fitness case) taken in random order from training dataset D_{train} and S_i the selection pool at iteration i, initially composed by the whole population:

1. $for\ d_i\ in\ D_{train}$:
   a) evaluate all candidate solutions in S_i on d_i;
   b) remove those candidate solutions from S_i whose fitness is worse than of the best-found solution;
2. if S contains more than one candidate solution, return one at random.

Figure 2.9: Pseudo-code for Lexicase Selection (LS) technique.

The underlying assumption embedded in LS is that problem's multi-modality is, at least partially, a factor of its fitness cases, each of which represents a circumstance with which a *correct* solution must deal. As such, different subsets of fitness cases may

call for different *modes of response*, i.e., may require the system to respond in a different manner. Under the light of this assumption, extension of lexicographic ordering to the fitness cases in randomized order ensures that a *good* fitness, calculated on a given case d_i, will be rewarded independently on solution's performance on other cases, while, at the same time, still rewarding the progress on a larger set of fitness cases (up to the size of training dataset).

By looking at the pseudo-code in 2.9, one can identify technique's vulnerability when dealing with regression problems: in regression, exact solutions to fitness cases can mostly be expected for *toy* problems, whereas real-world problems are often subject to noise and measurement error. As such, when applied on real-world regression problems, the standard LS typically uses only one fitness case for each parent selection, resulting in poor performance [36]. To deal with this limitation, authors in [36] proposed to *relax* the passing criteria (from S_i to S_{i+1}) by introducing a parameter ϵ, such that only individuals inside a predefined ϵ are selected for S_{i+1}. In their work, authors proposed four different definitions of ϵ and assessed their performance, along with four state-of-the-art techniques, on 3 synthetic and 3 real-world regression problems. The experimental results allowed to identify the most performing definitions of ϵ and proved the effectiveness of proposed technique when compared to state-of-the-art.

Given the results presented in [36], we decided to choose ϵ-LS with Automatic Threshold Adaptation (ATA) defined as semi-dynamic. Although authors did not find a statistically significant difference with another version of ATA, defined as dynamic, we decided to opt the former as semi-dynamic, as it has been defined as default solution by the author.

ϵ semi-dynamic formula: ϵ = median(abs(error_pop - median(error_pop)))

2.4.5 Fitness Evaluation

The Fitness function is used to evaluate how close a prediction is to the actual value. The main goal is to drive the algorithm to the optimal solution. In GP and GSGP, when used for regression, the prediction is a vector of continuous values. The fitness function compares the individuals of the population in order to find the elite (individual with the best fitness in the population). During the evolution process, it helps to design the solution of the algorithm. Since it's used for each individual and at every step of the evolutionary process, it has to be fast to save some computation time.

2.4.6 Elitism

The strategy of elitism is to maintain the best element observed so far through all generations. After the parent selection and the genetic operator variations, a new

population of offspring is created. The fitness evaluation of each offspring allows for the best individual of the new population to be determined. Best individuals from the current and previous generation are compared with their fitness for the training data. The elite of the previous generation will replace randomly an individual of the new population only if its fitness is better than the best individual of this population. This way the best solution observed still has a chance to produce offspring in the next generation.

2.4.7 Semantic Stopping Criterion

If the model's behavior on unseen data *highly* differs from the one on training data, one can say it *lacks generalisation*. This can be caused by many factors, among them, in the context of EAs, an inappropriate number of iterations. In fact, after a given point, further *machine learning* on available data can potentially make the model *overfit*, i.e. memorize the training data instead of generalizing from it.

Semantic Stopping Criterion (SSC) is a recently proposed stopping criterion [23] which operates in the context of GSOs, described in 2.4.2 and further extended in the context of neuroevolution [24]. More specifically, to decide when to stop, SSC uses information gathered from the *semantic neighborhood*, a set of *semantic neighbours* of the current-best solution in terms of training data, which are obtained by means of GSM. Authors propose two types of SSC: Error Deviation Variation (EDV) and Training Improvement Effectiveness (TIE). The first measures the percentage of those neighbors that, besides being *fitter* than the current-best, have a lower sample standard deviation of the absolute errors. The second measures the percentage of times the underlying semantic variation operator, in our case GSM, is able to produce a neighbor that is superior to the current-best. In both versions, only training data is considered for fitness calculation. The experimental results proved that the proposed stopping criteria are able to achieve a *competitive* generalization on the set of problems considered in their experiments, however no clear conclusions were provided regarding which of the two criteria is preferred. For this reason, and because computational effort for computing EDV nearly implies computation of TIE, we decided to assess both variants.

Figure 2.10 illustrates functioning of both SSC's variants - EDV in blue and TIE in red. Consider current-best solution at iteration i represented as a green point in 2D semantic space. Assuming a neighbourhood size of 10, the points generated within the gray box represent its semantic neighbors. Those represented in red, regard to semantic neighbors which are better than the current-best; since they are 5 out of 10, TIE is 50%. Those represented in red with a blue contour, regard to those neighbors that, besides being better than the current-best, have a lower sample standard deviation of the absolute errors; since they are 2 out of 5, EDV is 40%. Assuming the latter was

chosen as the stopping criterion with a threshold value of 50%, the search would stop at iteration i.

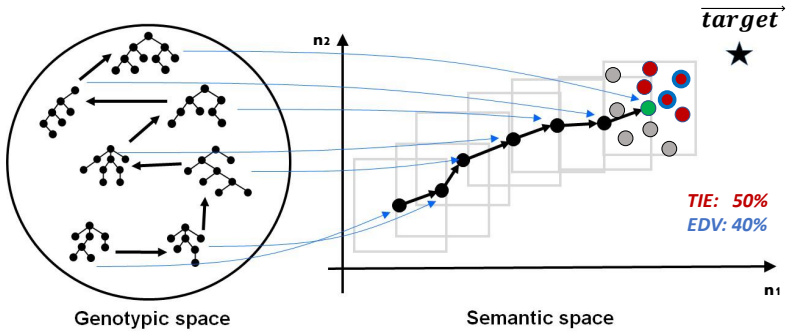

Figure 2.10: Illustration of SSC's functioning. The *star* in the figure represents the target vector on training data.

2.4.8 Genetic Programming as a Meta-Learning Technique

It is known that maximal generalization ability of an ensemble can be achieved through *prudent* combination of *diverse* BLs. Many ensemble methods, however, require their manual selection, parametrization and combination. This means that full potential of this combination of *search-paradigms* in generating *synergistic* effects might be under-explored. In this light, GP can be seen as an *effective* assembling method due to its *flexible* representation, *high* interpretability of evolved solutions and *powerful* inductive capabilities. To explore GP's role as a meta-learning technique, one has fulfill a fundamental requisite: to build the set of terminals from the combined predictions of several distinct base-learners.

The use of GP as an automatic EL technique is not new in the research field. To our knowledge, the first evidence comes from 2006 [26], when GP was used to combine 10 ANNs into an ensemble. Through experiments' analysis, a *notorious* superiority of the proposed approach was verified, when evaluated on 22 publicly available real-world SML problems. In [12], authors compared an equivalent approach against 3 ensemble approaches based on GAs. The experimental results involving 4 synthetic and 1 real-world symbolic regression tasks confirmed the preeminence of GP-based ensemble not only against the best BL, which was an ANN, but also the three different types of GA-based ensembles.

In [20], authors extended the usage of GP to learn ensemble policies by using up to 6 heterogeneous BLs, some of which ensembles themselves (which was the case of

Random Forest), and assessed system's performance on 10 synthetic symbolic regression problems. Additionally, in attempt to increase the synergistic effect of combining different BLs into an ensemble policy, authors proposed a novel operator entitled as *decision expression*, which splits the input space into sub-spaces that can then be handled by different sub-policies. Despite of their expectations, their approach performed significantly better than the best BL only at one problem. While on the remaining problems the performance of their system was comparable or worse. In their work, authors pointed several limitations, namely the overfitting - perceived by the fact GP simply selects from the BLs instead of *learning* a *complex* and *meaningful* policy - and the absence of real-world benchmark problems. The latter translates in the absence of potentially *challenging* fitness landscapes for the BLs, a scenario which seems suitable for the method they have proposed. Despite of carrying, in our opinion, an important contribution, for our surprise, this work was not published neither in a conference or a journal...

In [44] authors exploited the fact that semantics, defined in 2.4.2 as the vector of output values of a candidate solution calculated on the training observations, are independent from the underlying model and proposed an extension to GSGP, called Universal-GP (U-GP) in which some initial individuals are created by means of external programs, i.e., other ML techniques. More specifically, authors proved that, by producing semantics using fundamentally distinct ML models, in this context seen as BLs, and including them in the initial population of GSGP's evolutionary process along with random initial programs, it is possible to obtain a significant improvement over standard RHH initialization. It is also important to point that, after identifying system's sensibility towards overfitting of BLs, authors have introduced the *pre-evolutionary selection procedure*, referred as PESP, which attempts to exclude from the evolution those BLs which are unable to individually achieve *good* generalization performance after training and after a short run of GSGP. This approach, which essentially introduces an additional level of cross-validation, allowed the system to achieve superior performance when compared to standard S-GP on 4 real-world regression problems.

All aforementioned contributions, except [44], can be categorized as stacking, where standard GP assumes the role of a meta-learner from the combined predictions of several BLs. For this reason and from this moment on, we will entitle these kind of approaches as Stacking-SGP (S-SGP). Taking in consideration the information presented in the next section and to facilitate the reading of this document, we will use the nomenclature Stacking-GSGP (S-GSGP) for those cases when GSGP assumes the role of a meta-learner and Stacking-GP (S-GP) for any GP-based approach to combine the predictions of several BLs.

Chapter 3

Methodology

3.1 Proposed approach

As it was mentioned in 2.2.1, in this paper we explore S-GP; nevertheless, this paper presents several fundamental differences regarding the previous work. Since the first usage of S-GP [26], the research field in GP has dynamically evolved and several new approaches have emerged. Geometric-Semantic Operators (GSO), Semantic Stopping Criterion (SSC), ϵ-Lexicase Selection (ϵ-LS) and Evolutionary Demes Despeciation Algorithm (EDDA), the methods appear enumerated in ascending order of their recency, are among numerous examples of recent achievements of the scientific community in the research field. The first fundamental difference, hence a scientific contribution, consists of applying and comparing these novel achievements, assessing their effectiveness on the underlying tasks, as such updating the state-of-the-art in the research field. The second if related to the fact that none of the previous work provides a concise benchmark over regression problems when using S-GP: in [1, 29] authors study the effectiveness of their approach under the light of classification problems, whereas in [20] authors only cover synthetic regression problems. Apart from updating field's state-of-the-art, we also provide a concise overview of 7 synthetic and 4 real-world regression problems, which complements the scientific panorama in the research field.

3.2 Objectives

The experimental environment is built upon 7 synthetic and 4 real-world regression problems and the experiments were conducted to accomplish the following six objectives:

1. Identify and characterize hyper-parameter sets of S-GSGP which exhibit the highest performance on all the problems simultaneously, and on real-world and synthetic problems separately;

2. Compare the performance between different GP-based techniques in the context of stacking, namely: RHH initialization, Tournament selection (TS) and traditional Stopping Criteria with recently introduced EDDA, ϵ-LS and SSC, respectively;

3. Assess the validity of GSOs, in the context of stacking by comparing them with standard GP operators;

4. Compare system's performance against BLs, some of which ensembles themselves (which is the case of Random Forest);

It is important to highlight that, in order to assess systems' generalization ability, the experiments were conducted involving both training and unseen data instances.

3.3 Ensemble hyper-parameters

Table 3.1 provides an exhaustive enumeration of hyper-parameters that were used in our experiment for all the problems (both synthetically-generated and real-world). It is important to notice that during experimental phase we have performed an exhaustive search over the following parameter values: *Meta-Learner*, *BLs pre-training*, *Initialization*, *Selection*, $P(Crossover)$ and *Stopping criteria*, while maintaining all other parameters fixed. This means that a single execution of the benchmark environment, i.e., one run, implies 128 experiments for each of 11 considered problems (which yields a total of 1408 experiments per run). Throughout our experiments we guaranteed an equal number of fitness evaluations for all the types of S-GP systems studied - 70000 generations. For a S-GP system which uses RHH initialization technique this computational resource results in 500 generations for a population size of 140 (i.e., 500x140 = 70000). Similarly, for a S-GP system where initialization is performed by means of EDDA technique with maturation of 5 generations and percentage of GSGP demes equal to 50% ($EDDA_5 - 50\%$), this computational resource results in 200 generations for a population size of 100 (i.e., 100x5x100 + 200x100). The stopping criteria was compared after executing the experiments for the number of generations specified in the table (see *Nºgenerations*).

Having in mind the stochastic nature of employed algorithms and results' volatility upon data partition, i.e., to provide a robust and statistically-consistent analysis of experimental results, we repeated the experiments 60 times (runs), each with a different

pseudo-random number generator (a.k.a. *seed*), used for partitioning the data, and algorithms' initialization and execution.

Parameters	**Values**
Nºruns	60
Meta-Learner	S-SGP, S-GSGP
Nºgenerations	500_{RHH}, $200_{EDDA_5-50\%}$
Population size	140_{RHH}, $100_{EDDA_5-50\%}$
Function set	add, sub, mul, avg, min, decision
BLs set	LR, SVM, RF, MLP
Tuned BLs	True, False
Initialization	RHH, $EDDA_5 - 50\%$
Selection	TS, ϵ-LS
Crossover	Swap, GSC
Mutation	Subtree, GSM
$P(C)$	0, 0.2, 0.8, 1.0
$P(M)$	$1 - P(C)$
Stopping criteria	TIE, EDV, validation fitness, Nºgenerations

Table 3.1: Enumeration of hyper-parameters. It is worth to notice that *decision* stands for the decision expression operator proposed in [20], *BLs pre-training* indicates if the BLs were tuned or not, *ϵ-LS* represents the selection algorithm proposed in [36], $P(C)$ and $P(M)$ indicate the crossover and mutation probabilities, and *validation fitness* stands for the stopping criteria which operates upon the fitness calculated from validation partition by estimating the point from where further induction allows further generalization.

3.4 Experimental Problems

In this section, the reader can find a detailed characterization of the experimental problems we used in our experiments. Table 3.2 contains the mathematical formulation for 7 synthetic regression problems used in this study, whereas tables 3.3 and 3.1 complement the latter by specifying their bounds (the domain) and providing a graphical visualisation of the fitness landscapes. It is worth to highlight that these functions were studied in two dimensional input space. For each of these problems, we have generated 200 two-dimensional data points under Continuous Uniform Distribution where parameters for each dimension were chosen from table 3.3. Then, these points were used as the input for the functions defined in table 3.2 to generate the respective target vectors. As a result, each synthetic regression problem is defined by 200 data observations characterized in two-dimensional input feature-space and one dimensional output.

Table 3.4 summarizes the 4 real-world regression problems. The *Boston* problem [48] is from the field of real estate analysis and it consists of predicting the value of owner-occupied homes in suburbs of Boston, a city in United States of America, as

Name	$f(x_1, x_2)$
Branin	$a\left(x_2 - bx_1^2 + cx_1 - r\right)^2 + s(1-t)cos(x_1) + s$
Discus	$x_1^2 10^6 + x_2^2$
Griewank	$1 + \frac{1}{4000}\, x_1^2 + \frac{1}{4000}\, x_2^2 - \cos(x_1\ )\ \cos(\frac{1}{2}\, x_2\, \sqrt{2})$
Kotanchek	$\frac{e^{-(x_1-1)^2}}{3.2+(x_2-2.5)^2}$
Mexican Hat	$\frac{1}{\pi\sigma^2} 1 - \frac{1}{2} \frac{x_1^2+x_2^2}{\sigma^2} e^{-\frac{x_1^2+x_2^2}{2\sigma^2}}$
Rastrigin	$10d + \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi\ x_i\ )\ ]$
Weierstrass	$\sum_{i=1}^{d} \sum_{k=1}^{kmax} \frac{1}{2}^k \cos\left(2\pi 3^k (x_i + \frac{1}{2})\right) - d \sum_{k=1}^{kmax} \frac{1}{2}^k \cos\left(2\pi 3^k \frac{1}{2}\right)$

Table 3.2: Mathematical formulation of synthetic regression problems used in this study. The problems are listed in alphabetical order.

Bound:	Lower		Upper	
Problem	x_1	x_2	x_1	x_2
Branin	-5	0	10	15
Discus	-32.786	-32.786	32.786	32.786
Griewangk	-600	-600	600	600
Kotanchek	-2	-1	7	3
Mexican Hat	-5	-5	5	5
Rastrigin	-5	-5	5	5
Weierstrass	-0.5	-0.5	0.5	0.5

Table 3.3: Enumeration of search space's boundaries (the domain) for each function. It is worth to notice that columns *Lower* and *Upper* stand for lower and upper bounds of two dimensional search space.

a function of its geographic, socioeconomic, environmental and housing characteristics. The *Diabetes* problem [37] contains blood pressure and demographic data of 442 persons who happen to have diabetes and the target value is a quantitative measure of disease progression one year after the baseline measurement. The *PPB* problem [22] comes from the field of pharmacokinetics and consists of predicting the percentage of the initial drug dose which binds plasma proteins as a function of its 626 molecular descriptors. Finally, the *Parkinson* problem [49] is mostly composed of biomedical voice measurements from 42 people who, at the time of data-collection, happened to have Parkinson's disease at its early stage. They were recruited to a six-month trial of a tele-monitoring biomedical speech recording device for remote symptom progression monitoring. The objective is to predict Parkinson's symptom progression measured through total Unified Parkinson's Disease Rating Scale (UPDRS).

We considered to include these problems in our study as all of them have been target of research using several different ML techniques, they are significantly diverse and, in the case of real-world problems, are considered of *high* importance in their respective industries.

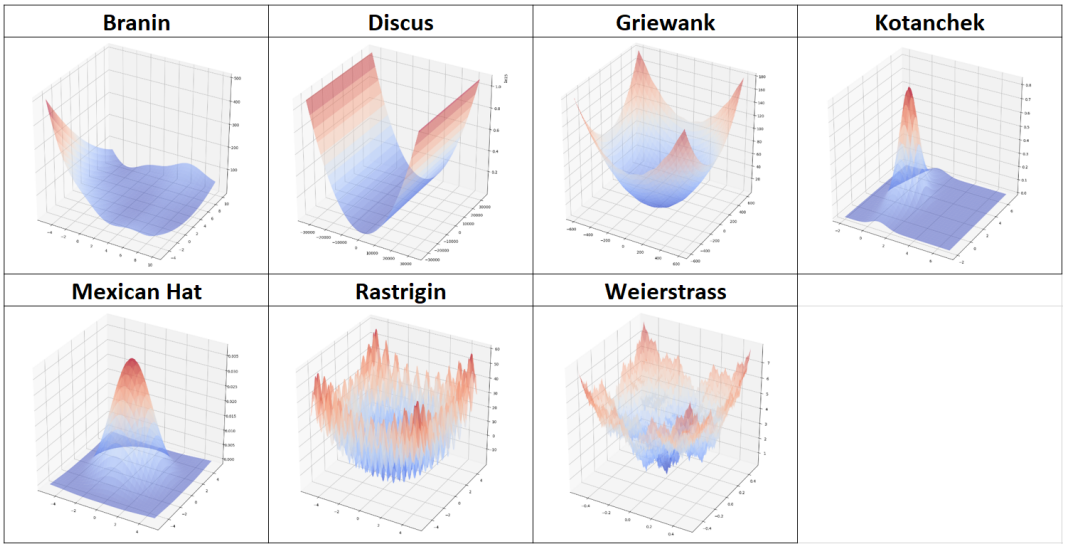

Figure 3.1: Optimization Problems

Dataset (ID)	#Features	#Instances	Field
Boston	13	506	Real Estate
Diabetes	10	442	Medicine
PPB	626	131	Pharmacokinetics
Parkinson	20	5876	Medicine

Table 3.4: Description of real-world regression problems used in the experiments. For each problem, the name (ID), number of input features (#Features), data instances (#Instances) and field of application (Field) is presented.

3.5 Problem dataset - Train and Test split

In the workflow of the experiment (figure A.1), we can see that each dataset has to be split into two parts 3.2, the training set used to train the base learners during the learning process, and the testing set which remains during the training but used for the final prediction.

Listing 3.1: Python code for Train and Test split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test
    = train_test_split(X, y, test_size=0.3, random_state=run_number)
```

The function 'train_test_split' from Scitkit-Learn shuffle randomly and split the given arrays 'X' and 'y' using the common rule of thumb 70/30. Meaning that 70% of data will be used for the training set, output arrays 'X_train' and 'y_train', and 30% the testing set, output arrays 'X_test' and 'y_test'. The random state ensures that at each run the random distribution will be different.

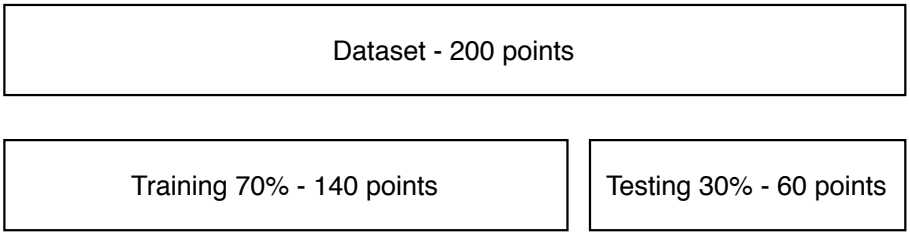

Figure 3.2: Dataset split representation

This way the dataset of 200 points is split into two parts, the training set of 140 points and the testing set of 60 points.

3.6 Base Learners hyper-parameters tuning

Hyper-parameter tuning is the fact of finding the right settings of a model for a specific dataset.

In a separated process from the workflow, only the problem training dataset is used.

GridSearchCV a module from Scikit-Learn (same library as the base learners) allows trying different parameters for a given model and dataset. A cross-validated score is given to all combinations of parameters, confirming that the best parameters set will perform well on different samples of the dataset. For each problem, the best parameter set is saved and used if the variable Tuned is set at True.

When not Tuned the base learners use their default hyper-parameters values which are not optimized for the given dataset, resulting in potentially bad performance.

3.7 Base Learners dataset - K-Fold data generation

As shown in the workflow of the experiment (figure A.1), after the problem dataset split, the base learners are trained with only a part of the training dataset because we want to output predictions based on unseen data also coming from the same training set. If the data to be predicted has already been seen during the training, the outputted predictions will be unrealistic and the ensemble will overfit for unseen data.

Using a K-fold technique, the training set is divided into K parts. For the base learners, K minus 1 folds are used as inputs during the training, and 1 fold is used as input for the prediction. We chose to use 10 folds, so the base learners are trained 10 times, to predict the target of 10 different folds.

The combination of predictions of those 10 base learners is the same number of instances (rows) as the training set. By using 90% of the training set has been used at each repetition, we ensure that the training is close to the complete training set and by using unseen data (10% of the training set at each repetition) the prediction is realistic.

Listing 3.2: Base Learners Train and Predict pseudo code

```
# function f(x, y, k=10) perform k-fold data generation
# X: independent variables of the training set
# y: dependent variable of the training set
# K: number of folds, default is 10
# base_learners: list containing the base learners used to train and predict
# fold_labels: vector with the same length as X, containing the folds labels.
# Equal number of instances by folds. Used to filter X and y.

1: for i = 1 to k:
2:  for each base_learners:
3:    Train using X and y, where fold_labels are different from i
4:    Predict using X and y, where the fold_labels equal i
5: concatenate all predictions in one array
```

The base learners dataset (Figure 3.3) is the concatenation of the prediction of each base learner for a given dataset. The array 'y' will remain the target data, but the array 'X' will no longer be used in the ensemble training workflow. Instead, the base learners predictions will be used as independent variables.

3.8 Base Learners dataset - Train and Validation split

A validation set is used to observe the behavior of the ensemble on unseen data (Validation stopping criteria 3.12).

The implementation of the split for the ensemble is quite simple because we are using the fold labels vector. By randomly selecting a fold (in our case a value between 1 and 10) the training set will be the instances with a different label than the one selected, corresponding to 90%. The rest of the instances will be used as a validation set.

This base learners prediction split (Figure 3.4) is different from the first split performed on the optimization problems data.

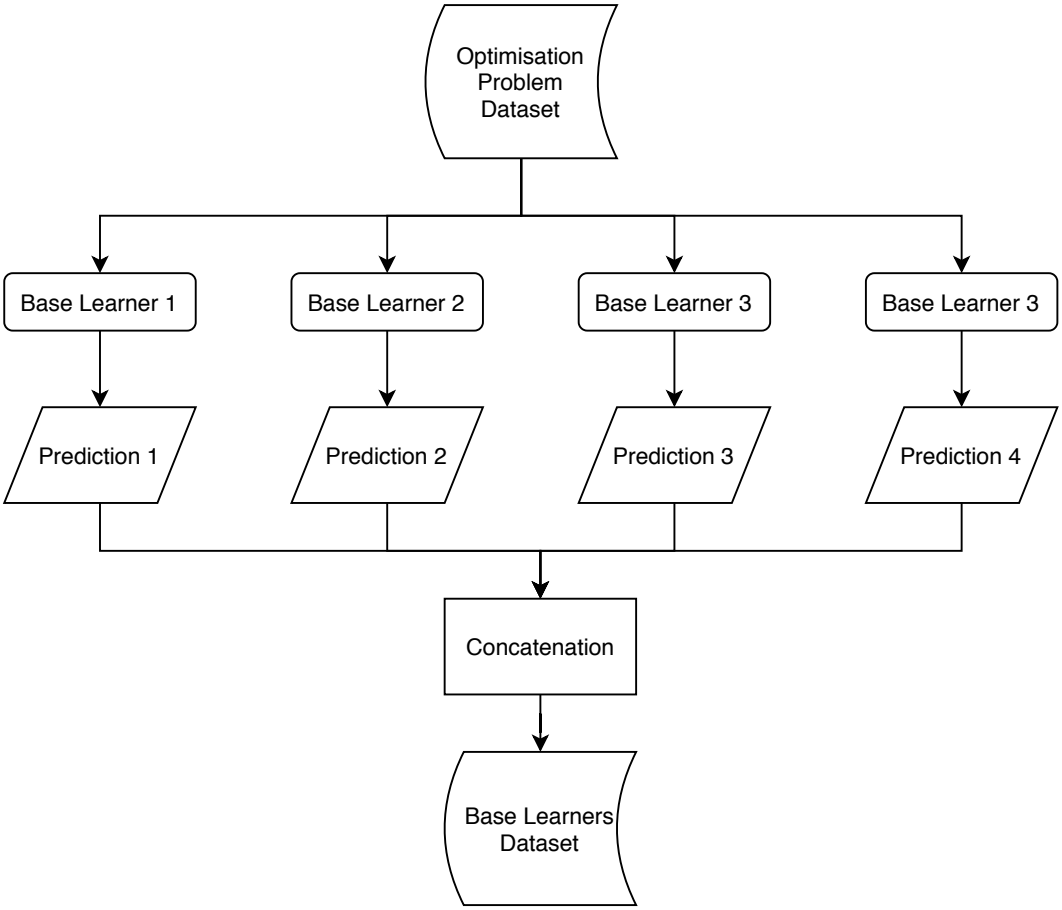

Figure 3.3: Base Learners dataset generation

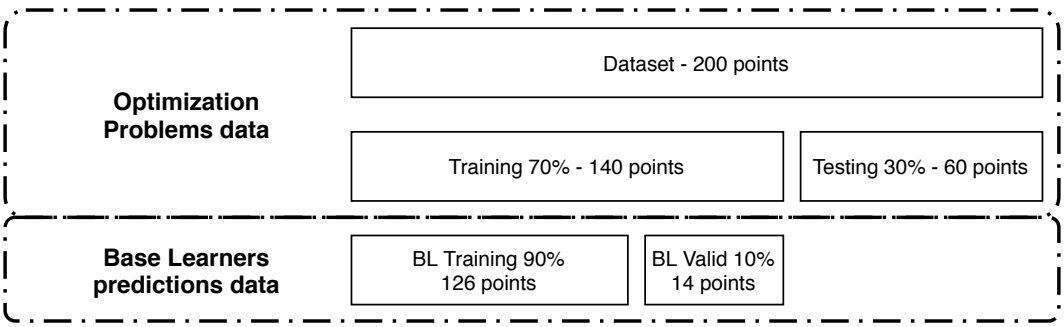

Figure 3.4: Base Learners predictions data split

3.9 Function set and Terminal set

Function Set

Defined as a parameter of the algorithm, the function set is a toolbox of arithmetic functions used in the tree structure of an individual. A function is selected randomly during the population's initialization. Terminals are used as inputs (arity 1 or 2) and the output will be the mathematical association represented by the function. A big enough function set also helps to maintain diversity in the individual composition.

Listing 3.3: Function set variable

```
primitive_function_set=('add', 'sub', 'mul', 'avg', 'min', 'deci')
# Add: Addition
# Sub: Subtraction
# Mul: Multiplication
# Avg: Average
# Min: Minimum
# Deci: Decision
```

Terminal Set

A feature from the base learners dataset 3.3 can be selected randomly and used as terminal. A terminal value can also be picked randomly within a defined range of constant. In our case, the constant range is between -1 and 1.

3.9.1 Decision

The function set is mostly composed of basic arithmetic functions. But it is possible to add more complex functions like the Decision function.

Each application of decision function splits the search space in two sub-spaces by randomly selecting a threshold in the range -1 and 1 at a randomly chosen input feature. Then, each sub-set of the space will be approximated by another sub-policies. This way, the decision operator looks like the Decision Trees learning but with random feature and threshold selection.

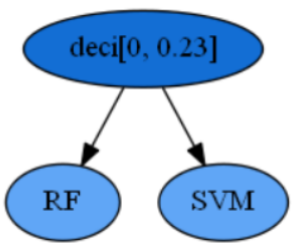

Figure 3.5: Decision

In the figure 3.5, the first feature (position 0) of a base learner dataset 3.3 is randomly selected. The randomly selected threshold 0.23 split the feature vector in two. All the corresponding values between -1 and 0.23 will use the predictions of the base learner RF, and all the corresponding values between 0.23 and 1 will use the predictions of the base learner SVM.

3.10 Fitness Evaluation

The fitness evaluation of an individual happens in 2 steps:

At the execution of an individual, a vector is returned representing the semantic of the individual structure for a given dataset (training, validation or test data).

The fitness function 'Root Mean Square Error' returns the error comparing the predictions (from the execution) to the corresponding target values.

3.10.1 Memoization

Memoization is an optimization technique used to speed up functions by storing values in the cache. A function using memoization stores the result of an execution the first time an input parameter occurs. The next time the same input parameter is used, the result is retrieved from the cache, saving some computational time.

Since the structure of the individuals of the same population shares the same parentage, memoization is particularly efficient with GSGP. The genotype of each parent is calculated only once, then called in from the cache for any offspring.

Listing 3.4: Memoization Pseudo-code

```
# Memoization pseudo-code:
# Avg: mean of the dataset, use as id of the dataset
# Parent_id: ID of the individual
memoize(function):
  Avg = mean(dataset)
  If tuple (avg, parent_id) in cache:
    Return result, execution of the individidual for dataset
  Else
    Add in cache result, execution of the individual for a dataset
```

Listing 3.5: Memoization Python code

```
# Memoization function:
def memoize(f):
    # Add a cache memory to the input function.
    def decorated_function(*args):
        avg = np.mean(args[0])
        if (avg, args[1]) in variables.cache:
            return variables.cache[(avg, args[1])]
        else:
            variables.cache[(avg, args[1])] = f(*args)
            return variables.cache[(avg, args[1])]
    return decorated_function

# Usage:
# t: ID of a parent
parent_result = lambda data, t: self.find_parent(t).execute(data)
parent_result = memoize(parent_result)
```

3.11 Parent Selection

In the code, the parent selection is used during the initialization (EDDA or RHH) and at the start of each generation to create the new population. This means that the parent selection is repeated for the number of individuals necessary in the new population (2 parents for a crossover operator, 1 parent for a mutation operator).

3.11.1 Tournament

The Tournament selection probability is fixed at 20%, meaning that at each Tournament, for a population of 100 individuals, 20 will be selected randomly. Then from this intermediary group, the individual with the best fitness is selected as a parent.

3.11.2 Epsilon Lexicase

In this implementation, Epsilon Lexicase treats each instance of the dataset as one case. So the number of cases is equal to the length of the dataset (number of rows).

3.12 Stopping Criteria

GSGP is prone to learn the examples of a given dataset instead of learning from the patterns. Using semantic neighborhood it is possible to stop the iterations using TIE or EDV with a given threshold to limit the overfitting effect of GSGP.

TIE and EDV

- Sample size: 100 - Number of semantic neighbors produced before analysis
- Threshold: 25% - For 100 semantic neighbors, if less than 25 of them respect the criteria of EDV or TIE (success rate < threshold), the evolution process is stopped.
- In our experimentation, the evolution process is not stopped, but we keep the corresponding elite for future analysis (edv_elite and tie_elite).

Minimum Validation and Nºgenerations

- We also store the elite for the minimum fitness observed using the validation set and the elite of the last generation using the training set for future analysis (Validation and Nºgenerations).

Chapter 4

Results

In this chapter we will present the experimental results obtained on 7 synthetic and 4 real-world regression problems reported in 3.4. The results are exhibited and analyzed according to objectives exposed in 3.2.

4.1 Performance Analysis

In order to identify the best parametrization for S-SGP and S-GSGP system, we used the average of the performance ranks, because of the different scale of each problem. More specifically, first we ranked all the hyper-parameters sets by their average performance (RMSE) on each problem. Then, we computed the average of their ranks on all the problems simultaneously, and on real-world and synthetic problems separately. Finally, we have sorted the average ranks in ascending order and analyzed the top 5 hyper-parameter sets. These results can be found in tables [4.1, 4.2]. Notice that, in this context, the smaller the average rank, the higher the average performance across a given problem set. There is in total 256 different combinations of hyper-parameters, called systems in the following sections (2 algorithms x 2 initializations x 2 selections x 2 BLs tuning x 4 stopping criteria x 4 P(C) = 256 systems). The rank is established by problem, so from 1 to 256.

From the analysis of the table one can clearly see that, problem-wise, the S-GSGP achieves the highest performance when the initialization is EDDA, selection is Tournament, the base-learners are tuned and the mutation is not used at all. The latter observation appears to be a confirmation of our previous speculations about Convex-Hull and the fact BLs might surround the global optima, so that by means of Geometric

Type	Algo	Initialization	Selection	Tuned BLs	P(C)	Stopping Criteria	AVG Rank	Index
All	S-GSGP	EDDA	TS	TRUE	100%	TIE	20.55	1
	S-GSGP	EDDA	TS	TRUE	100%	Validation	25.00	2
	S-GSGP	EDDA	TS	TRUE	100%	EDV	27.27	3
	S-GSGP	EDDA	TS	TRUE	100%	Nºgenerations	28.09	4
	S-GSGP	RHH	TS	TRUE	0%	TIE	53.36	5
Synthetic	S-GSGP	EDDA	TS	TRUE	100%	TIE	24.00	1
	S-GSGP	EDDA	TS	TRUE	100%	Validation	31.43	2
	S-GSGP	EDDA	TS	TRUE	100%	EDV	34.86	3
	S-GSGP	EDDA	TS	TRUE	100%	Nºgenerations	35.71	4
	S-GSGP	EDDA	TS	FALSE	80%	TIE	36.00	5
Real-world	S-GSGP	EDDA	TS	TRUE	100%	Validation	13.75	1
	S-GSGP	EDDA	TS	TRUE	100%	EDV	14.00	2
	S-GSGP	EDDA	TS	TRUE	100%	TIE	14.50	3
	S-GSGP	EDDA	TS	TRUE	100%	Nºgenerations	14.75	4
	S-GSGP	EDDA	TS	TRUE	80%	TIE	24.25	5

Table 4.1: Average performance rank - Top 5 by problem type

Type	Tuned BLs	Algorithm	Initialization	Selection	P(C)	Stopping Criteria	AVG Rank
All	FALSE	S-GSGP	EDDA	TS	20%	EDV	63.55
		S-SGP	EDDA	TS	100%	EDV	111.45
	TRUE	S-GSGP	EDDA	TS	100%	TIE	20.55
		S-SGP	EDDA	ϵ-LS	100%	EDV	94.82
Synthetic	FALSE	S-GSGP	EDDA	TS	80%	TIE	36.00
		S-SGP	EDDA	TS	100%	EDV	116.00
	TRUE	S-GSGP	EDDA	TS	100%	TIE	24.00
		S-SGP	EDDA	ϵ-LS	20%	Validation	111.14
Real-world	FALSE	S-GSGP	EDDA	ϵ-LS	0%	TIE	86.75
		S-SGP	EDDA	ϵ-LS	100%	EDV	85.00
	TRUE	S-GSGP	EDDA	TS	100%	Validation	13.75
		S-SGP	EDDA	ϵ-LS	100%	TIE	56.25

Table 4.2: Average performance rank - Best system by problem type, base learners hyper-parameters and algorithm

Semantic Operators, namely the Geometric Semantic Crossover, one can effectively approximate their semantics towards the target. Moreover, one can see that the hyper-parameter set with the smallest average rank uses a SCC, namely TIE. There is one parameter-set which significantly deviates from the trend in the top 5: S-GSGP initialized by means of RHH with no crossover at all; nevertheless, its average performance rank is significantly higher from the first 4 sets, which follow the trend.

When looking at the results obtained only on synthetic problems, one can confirm the trend verified previously. There is one parameter-set which slightly deviates from the trend in the top 5: S-GSGP which BLs are not tuned, the $P(C) = 80\%$ and TIE-SSC. In this case, one can speculate that, with some mutation, the system is still able to achieve almost as good results as with no tuned BLs, no mutation and TIE-SSC; this probably happens because the BLs, although not tuned, provide a fair approximation on synthetic problems.

When looking at the results obtained only on real-world problems, one can confirm the trend verified in the previous two paragraphs. The S-GSGP achieves the highest performance when the Initialization is EDDA, Selection is Tournament, the base-learners are tuned and the mutation is not used at all. Regarding the best performing hyper-parameter set, the only difference with the previous two types of problems consists in the stopping criterion 'Validation', which is not a SCC.

4.2 Statistical Assessment

In this section we provide a statistically-sustained comparison of experimental hyper-parameters of all systems, namely: S-SGP with S-GSGP algorithms, EDDA with RHH initialization techniques, Tournament with ϵ-Lexicase selection procedures, different stopping criteria and BLs' pre-training. The statistical assessment was performed by means of Wilcoxon rank-sum test for pairwise data comparison of the average RMSE under the alternative hypothesis that the samples do not have equal medians. Tables [4.3, 4.4, 4.5, 4.6, 4.7] report the test statistic and the respective p-value for the two samples being compared (in the table, *Sample A* and *Sample B*). Moreover, we also report the average rank of each sample.

4.2.1 Comparison of system's hyper-parameters - Global

Hyper-Parameter	Sample A	Sample B	AVG Rank A	AVG Rank B	Test Statistic	P-Value
P(C)	0%	20%	129.20	131.52	123993	9.87E-01
	0%	80%	129.20	132.35	118485	3.00E-01
	0%	100%	129.20	120.93	112162	2.73E-02
	20%	80%	131.52	132.35	120118	4.63E-01
	20%	100%	131.52	120.93	113771	5.62E-02
	80%	100%	132.35	120.93	107040	1.60E-03
Stopping Criteria	EDV	Nºgenerations	122.80	150.41	68068	2.28E-21
	EDV	TIE	122.80	119.22	111149	3.03E-01
	EDV	Validation	122.80	121.56	115329	1.34E-01
	Nºgenerations	TIE	150.41	119.22	53839	1.35E-26
	Nºgenerations	Validation	150.41	121.56	47112	7.94E-45
	TIE	Validation	119.22	121.56	120517	7.35E-01
Tuned BLs	FALSE	TRUE	140.00	117.00	408558	1.02E-08
Selection	ϵ-LS	TS	135.66	121.34	422113	1.30E-06
Initialization	EDDA	RHH	107.70	149.30	262142	5.36E-53
Algorithm	S-SGP	S-GSGP	150.90	106.10	303616	1.98E-36

Table 4.3: Statistical assessment - Global

In table 4.3, we are comparing globally the hyper-parameters, without any focus on a particular hyper-parameter, like it is the case in the following sections. For the hyper-parameters algorithms, initializations, selections and tuned BLs, each sample contains 1408 systems (256 systems x 11 problems / 2). For the hyper-parameters P(C) and stopping criteria, each sample contains 704 systems (256 systems x 11 problems / 4).

The crossover probability hyper-parameter P(C) 100% (avg rank 120.93) outperforms P(C) 80% (avg rank 132.35) and 0% (avg rank 129.20) in a statistically significant way. It also outperforms P(C) 20% (avg rank 131.52) but not in a statistically significantly way. The stopping criteria hyper-parameter, Nºgenerations (avg rank 150.41) is significantly worse than any other stopping criteria. TIE (avg rank 119.22) outperforms EDV (avg rank 122.80) and Validation (avg rank 121.56) but not in a statistically significant way. Regarding the tuning of the Base Learners hyper-parameters, set as True (avg rank 117) outperforms set as False (avg rank 140) in a statistically significant way. The selection TS (avg rank 121.34) outperforms ϵ-LS (avg rank 135.66) in a statistically significant way. The initialization EDDA (avg rank 107.70) outperforms RHH (avg rank 149.30) in a statistically significant way. The algorithm S-GSGP (avg rank 106.10) outperforms S-SGP (avg rank 150.90) in a statistically significant way.

Based on those results, it seems more beneficial to use: S-GSGP as algorithm, base learners with tuned hyper-parameters, EDDA as initialization, Tournament as selection, TIE as stopping criteria and P(C) 100%. This is coherent with global performance table 4.2 where the best performing system on all problems is the same.

4.2.2 Comparison of system's hyper-parameters - By algorithm and BLs tuning

In tables 4.4, 4.5, 4.6 and 4.7, for the hyper-parameters initializations and selections, each sample contains 704 systems (256 systems x 11 problems / (2 initializations x 2 BLs tuning)). For the hyper-parameters P(C) and stopping criteria, each sample contains 176 systems (256 systems x 11 problems / (2 initializations x 2 BLs tuning x 4)).

4.2.2.1 For all S-GSGP systems without tuned BLs

Hyper-Parameter	Sample A	Sample B	AVG Rank A	AVG Rank B	Test Statistic	P-Value
P(C)	0%	20%	30.26	33.18	6101	1.27E-02
	0%	80%	30.26	35.14	5090	6.72E-05
	0%	100%	30.26	31.43	6296	2.75E-02
	20%	80%	33.18	35.14	6674	9.98E-02
	20%	100%	33.18	31.43	6903	1.91E-01
	80%	100%	35.14	31.43	7472	6.41E-01
Stopping Criteria	EDV	Nºgenerations	30.68	40.97	4095	1.17E-05
	EDV	TIE	30.68	26.83	5357	8.10E-03
	EDV	Validation	30.68	31.52	6572	7.24E-02
	Nºgenerations	TIE	40.97	26.83	1391	3.01E-15
	Nºgenerations	Validation	40.97	31.52	3268	6.63E-11
	TIE	Validation	26.83	31.52	7462	6.30E-01
Selection	ϵ-LS	TS	38.38	26.63	16873	1.10E-13
Initialization	EDDA	RHH	27.68	37.32	15946	2.51E-15

Table 4.4: Statistical assessment - Without tuned BLs, for all S-GSGP, by hyper-parameter

In table 4.4, only the S-GSGP systems not using tuned BLs are compared. The crossover probability P(C) 0% (avg rank 30.26) outperforms P(C) 80% (avg rank 35.14), 20% (avg rank 33.18) and P(C) 100% (avg rank 31.14) in a statistically significant way. The stopping criteria hyper-parameter, TIE (avg rank 26.83) outperforms EDV (avg rank 30.68) and Nºgenerations (avg rank 40.97) in a statistically significant way. TIE outperforms Validation (avg rank 31.52) but not in a statistically significant way. The selection Tournament (avg rank 26.33) outperforms ϵ-LS (avg rank 27.68) in a statistically significant way. EDDA (avg rank 27.68) outperforms RHH (avg rank 37.32) in a statistically significant way.

Based on those results, it seems more beneficial to use S-GSGP without tuned base learners: EDDA as initialization, Tournament as selection, TIE as stopping criteria and P(C) 0%. This combination differs slightly from the corresponding combination observed in 4.2 where EDV is used as stopping criteria and 20% for P(C).

4.2.2.2 For all S-SGP systems without tuned BLs

Hyper-Parameter	Sample A	Sample B	AVG Rank A	AVG Rank B	Test Statistic	P-Value
P(C)	0%	20%	35.36	31.98	6471	5.17E-02
	0%	80%	35.36	34.56	7449	6.17E-01
	0%	100%	35.36	28.10	5524	8.24E-04
	20%	80%	31.98	34.56	7420	5.87E-01
	20%	100%	31.98	28.10	6442	4.68E-02
	80%	100%	34.56	28.10	5455	5.68E-04
Stopping Criteria	EDV	Nºgenerations	30.90	38.03	4851	3.33E-05
	EDV	TIE	30.90	33.39	6381	6.42E-02
	EDV	Validation	30.90	27.68	6684	1.63E-01
	Nºgenerations	TIE	38.03	33.39	5385	1.18E-03
	Nºgenerations	Validation	38.03	27.68	2536	3.94E-14
	TIE	Validation	33.39	27.68	5802	6.51E-03
Selection	ϵ-LS	TS	31.79	33.21	28938	2.66E-01
Initialization	EDDA	RHH	23.34	41.66	14867	2.29E-17

Table 4.5: Statistical assessment - Without tuned BLs, for all S-SGP, by hyper-parameter

In table 4.5, only the S-SGP systems not using tuned BLs are compared. The crossover probability P(C) 100% (avg rank 28.10) outperforms P(C) 0% (avg rank 35.36), P(C) 20% (avg rank 31.98) and P(C) 80% (avg rank 34.56) in a statistically significant way. The stopping criteria Validation (avg rank 27.68) outperforms TIE (avg rank 33.39) and Nºgenerations (avg rank 38.03) in a statistically significant way. Validation also outperforms EDV (avg rank 30.90) but not in a statistically significant way. The selection ϵ-LS (avg rank 31.79) outperforms Tournament (avg rank 33.21) but not in a statistically significant way. The initialization EDDA (avg rank 23.34) outperforms RHH (avg rank 41.66) in a statistically significant way.

Based on those results, it seems more beneficial to use S-SGP without tuned Base Learners: EDDA as initialization, ϵ-LS as selection, Validation as stopping criteria and

100% as crossover probability. This system differs slightly from the one observed in 4.2 where Tournament is used as selection instead of ϵ-LS, and EDV as stopping criteria instead of validation.

4.2.2.3 For all S-GSGP systems with tuned BLs

Hyper-Parameter	Sample A	Sample B	AVG Rank A	AVG Rank B	Test Statistic	P-Value
P(C)	0%	20%	30.90	35.14	6906	1.93E-01
	0%	80%	30.90	36.63	6085	1.19E-02
	0%	100%	30.90	27.34	7557	7.33E-01
	20%	80%	35.14	36.63	6855	1.68E-01
	20%	100%	35.14	27.34	7073	2.91E-01
	80%	100%	36.63	27.34	5259	1.87E-04
Stopping Criteria	EDV	Nºgenerations	30.61	40.82	3937	2.19E-07
	EDV	TIE	30.61	26.61	5358	1.12E-02
	EDV	Validation	30.61	31.97	7081	2.96E-01
	Nºgenerations	TIE	40.82	26.61	1286	3.59E-16
	Nºgenerations	Validation	40.82	31.97	3579	8.24E-10
	TIE	Validation	26.61	31.97	6518	1.27E-01
Selection	ϵ-LS	TS	37.63	27.37	16540	2.91E-14
Initialization	EDDA	RHH	27.24	37.76	14557	5.61E-18

Table 4.6: Statistical assessment - With tuned BLs, for all S-GSGP, by hyper-parameter

In table 4.6 only the S-GSGP systems using tuned BLs are compared. The crossover probability P(C) 100% and P(C) 0% are statistically significantly better than P(C) 80% and obtain better results than P(C) 20% without statistical evidence. P(C) 100% offers better average rank than P(C) 0%. TIE is statistically significantly better than EDV and Nºgenerations, it also offers better average rank than Validation. Tournament is statistically significantly better than ϵ-LS. EDDA is statistically significantly better than RHH.

Based on those results, it seems more beneficial to use S-GSGP with tuned base learners: EDDA as initialization, Tournament as selection, TIE as stopping criteria and P(C) 100%. This is the same combination observed in the performance table 4.2 for S-GSGP with tuned base learners.

4.2.2.4 For all S-SGP systems with tuned BLs

In table 4.7 only the S-SGP systems using tuned BLs are compared. The crossover probability P(C) 80% (avg rank 30.01) outperforms P(C) 0% (avg rank 35.96) in a statistically significant way, it also outperforms P(C) 100% (avg rank 31.68) and P(C) 20% (avg rank 32.35) but not in statistically significant way. The stopping criteria EDV (avg rank 29.27) outperforms Nºgenerations (avg rank 39.75) in a statistically significant way, it also outperforms TIE (avg rank 31.54) and Validation (avg rank 29.44) but not in a statistically significant way. The selection ϵ-LS (avg rank 30.26) outperforms Tournament (avg rank 37.74) in a statistically significant way. The initialization EDDA (avg rank 25.15) outperforms RHH (avg rank 39.88) in a statistically significant way.

Hyper-Parameter	Sample A	Sample B	AVG Rank A	AVG Rank B	Test Statistic	P-Value
P(C)	0%	20%	35.96	32.35	6800	1.44E-01
	0%	80%	35.96	30.01	5530	8.51E-04
	0%	100%	35.96	31.68	5647	1.56E-03
	20%	80%	32.35	30.01	6506	5.82E-02
	20%	100%	32.35	31.68	6535	6.42E-02
	80%	100%	30.01	31.68	7492	6.62E-01
Stopping Criteria	EDV	Nºgenerations	29.27	39.75	4230	1.47E-07
	EDV	TIE	29.27	31.54	6995	2.94E-01
	EDV	Validation	29.27	29.44	7009	2.50E-01
	Nºgenerations	TIE	39.75	31.54	4974	3.22E-05
	Nºgenerations	Validation	39.75	29.44	2629	2.51E-14
	TIE	Validation	31.54	29.44	7675	8.67E-01
Selection	ϵ-LS	TS	30.26	34.74	23529	8.01E-05
Initialization	EDDA	RHH	25.15	39.88	20592	4.22E-08

Table 4.7: Statistical assessment - With tuned BLs, for all S-SGP, by hyper-parameter

Based on those results, it seems more beneficial to use S-SGP with tuned Base Learners: EDDA as initialization, ϵ-LS as selection, Validation as stopping criteria and 80% as crossover probability. This combination differs slightly from the corresponding one observed in the performance table 4.2 where the crossover probability is 100% instead of 80%.

4.2.3 Best S-GSGP vs Best S-SGP - By problem type and base learners hyper-parameters

Type	Tuned BLs	A - S-GSGP, EDDA	B - S-SGP, EDDA	AVG Rank A	AVG Rank B	Test statistic	P-Value
All	FALSE	[TS, 20%, EDV]	[TS, 100%, EDV]	63.55	111.45	99153	4.31E-02
	TRUE	[TS, 100%, TIE]	[ϵ-LS, 100%, EDV]	20.55	94.82	46326	1.57E-37
Synthetic	FALSE	[TS, 80%, TIE]	[TS, 100%, EDV]	36.00	116.00	42703	5.46E-01
	TRUE	[TS, 100%, TIE]	[ϵ-LS, 20%, Val]	24.00	111.14	11871	1.40E-38
Real-world	FALSE	[ϵ-LS, 0%, TIE]	[ϵ-LS, 100%, EDV]	86.75	85.00	13444	3.45E-01
	TRUE	[TS, 100%, Val]	[ϵ-LS, 100%, TIE]	13.75	56.25	10521	2.54E-04

Table 4.8: Statistical assessment - Comparison of best S-GSGP vs best S-SGP - By problem type and base learners hyper-parameters

In table 4.8, Sample A corresponds to the best system using S-GSGP as algorithm and EDDA as initialization, Sample B corresponds to the best performing system using S-SGP as alogrithm and EDDA as initialization. Each sample contains 660 runs for all problems, 420 runs for the synthetic problems and 240 runs (60 runs x number of problems). Each run report the RMSE of the system for a specific problem.

For all problem considered, and without tuned base learners the best S-GSGP system (avg rank 63.55), which uses Tournament as selection, P(C) 20% and EDV as stopping criteria, outperforms in a statistically significant way the best S-SGP system (avg rank 111.45), which uses Tournament as selection, P(C) 100% and EDV as stopping criteria. Still for all problem considered, and with tuned base learners the best S-GSGP system (avg rank 20.55), which uses Tournament as selection, P(C) 100% and TIE as stopping criteria, strongly outperforms in a statistically significant way the best

S-SGP system (avg rank 94.82), which uses ϵ-LS as selection, P(C) 100% and EDV as stopping criteria.

Regarding the synthetic problems, when the base learners are not tuned, the best S-GSGP system (avg rank 36.00), which uses Tournament as selection, P(C) 80% and TIE as stopping criteria, outperforms not in a statistically significant way the best S-SGP system (avg rank 116.00), which uses Tournament as selection, P(C) 100% and EDV as stopping criteria. Still with the synthetic problems, when the base learners are tuned, the best S-GSGP system (avg rank 24.00), which uses Tournament as selection, P(C) 100% and TIE as stopping criteria, strongly outperforms in a statistically significant way the best S-SGP system (avg rank 111.14), which uses ϵ-LS as selection, P(C) 20% and Validation as stopping criteria.

Regarding the Real-world problems, when the base learners are not tuned, the best S-SGP system (avg rank 85.00), which uses ϵ-LS as selection, P(C) 100% and EDV as stopping criteria, outperforms not in a statistically significant way the best S-GSGP system (avg rank 86.75), which uses ϵ-LS as selection, P(C) 0% and TIE as stopping criteria. Still with the Real-world problems, when the base learners are tuned, the best S-GSGP system (avg rank 13.75), which uses Tournament as selection, P(C) 100% and Validation as stopping criteria, outperforms in a statistically significant way the best S-SGP system (avg rank 56.25), which uses ϵ-LS as selection, P(C) 1000% and TIE as stopping criteria.

4.2.4 Comparison of the best performing system vs BLs

As shown in table 4.1 and 4.2 our best performing ensemble system for all types of problems is obtained using tuned base learners, EDDA as initialisation, Tournament as selection. Its 4 stopping criteria are always in the top 4 for each type of problem.

It has also been statistically assessed in tables 4.3 and 4.6, that the best hyper-parameters are: S-GSGP as algorithm, tuned base learners, EDDA as initialization, Tournament as selection. No stopping criterion has been shown as statistically significantly better than all the others.

It is important to compare the stopping criteria of our best performing ensemble, between themselves but also against the base learners feeding the system.

To create the ranking used in table 4.9, the average RMSE (B.3) of each problems for 8 base learners systems (RF Tuned, LR Tuned, SVM Tuned, MLP Tuned, RF Not Tuned, LR Not Pr-etrained, SVM Not Tuned, MLP Not Tuned) were added to the previous global rank used for table 4.1 and the rank has been updated. This way the base

Sample A	Sample B	AVG Rank A	AVG Rank B	Test Statistic	P-Value
TIE	Validation	21.27	25.91	74938	1.74E-01
	EDV	21.27	28.09	53572	6.57E-01
	Nºgenerations	21.27	28.82	32034	1.73E-01
	RF	21.27	123.45	48656	6.42E-35
	SVM	21.27	174.45	28122	2.72E-61
	MLP	21.27	219.18	7293	8.30E-96
	LR	21.27	255.73	15652	5.15E-80
Validation	EDV	25.91	28.09	85125	4.82E-01
	Nºgenerations	25.91	28.82	97353	4.35E-01
	RF	25.91	123.45	47945	1.05E-35
	SVM	25.91	174.45	25571	4.24E-65
	MLP	25.91	219.18	5424	2.75E-99
	LR	25.91	255.73	16794	2.77E-78
EDV	Nºgenerations	28.09	28.82	25237	4.07E-01
	RF	28.09	123.45	50340	4.31E-33
	SVM	28.09	174.45	29214	1.07E-59
	MLP	28.09	219.18	6848	1.25E-96
	LR	28.09	255.73	15068	3.27E-81
Nºgenerations	RF	28.82	123.45	48234	2.20E-35
	SVM	28.82	174.45	28032	2.00E-61
	MLP	28.82	219.18	7074	3.27E-96
	LR	28.82	255.73	16723	2.10E-78
RF	SVM	123.45	174.45	39506	9.84E-46
	MLP	123.45	219.18	15054	4.95E-82
	LR	123.45	255.73	86610	4.60E-06
SVM	MLP	174.45	219.18	31458	1.72E-56
	LR	174.45	255.73	87202	8.13E-06
MLP	LR	219.18	255.73	77932	2.11E-10

Table 4.9: Statistical assessment - Best performing ensemble system for all problems and BLs: [Tuned BLs, S-GSGP, EDDA, TS, P(C) 100%] - (Average RMSE by problem in table B.3)

learners can be fairly compared to all analyzed ensemble systems.

For table 4.9, since our best performing ensemble systems are using only tuned base learners, only those ones will be used in that comparison. As in the previous tables the statistical assessment was performed by means of Wilcoxon rank-sum test using the RMSE of each run. Each sample contains 660 runs (60 runs x 11 problems).

For the best performing ensemble system, which uses tuned base learners, S-GSGP as algorithm, EDDA as initialization, Tournament as selection and P(C) 100% as crossover probability, the stopping criteria TIE (avg rank 21.27) outperforms Validation (avg rank 25.91), EDV (avg rank 28.09) and Nºgenerations (avg rank 28.82) not in a significant way. This same system with any stoppy criteria strongly outperforms in a significant way its own base learners, which are tuned RF (avg rank 123.45), tuned

SVM (avg rank 174.45), tuned MLP (avg rank 219.18) and tuned LR (avg rank 255.73).

Chapter 5

Discussion

5.1 Summary

As seen in table 4.1, the best performing ensemble system for all types of problems (Synthetic and Real-world) is using: Tuned base learners, S-GSGP as algorithm, EDDA as initialization, Tournament as selection, 100% as crossover probability and TIE as stopping criteria which perform better considering all problems. For Real-world problems, the only difference is Validation used as stopping criteria. The top 5 systems in the 3 types of problems are all using S-GSGP as algorithm and Tournament as selection. Differences can be noticed in the fifth position of each type of problem on the Initialization, pre-training of the base learners and the crossover probability.

In table 4.1, we can also appreciate the performance of the S-SGP systems when the base learners are tuned or not. S-GSGP systems always perform better than S-SGP, except for Real-world problems when the base learners are not tuned. We can also notice that for all type of problems and base learners not tuned, S-GSGP never uses 100% as crossover probability (mutation is always used), which is always the case when using tuned base learners.

The comparison of the system hyper-parameters table 4.3 statistically confirms the composition of our best performing ensemble.

When using not tuned base learners and S-GSGP, the most individually beneficial hyper-parameters are: EDDA as initialization Tournament as selection, TIE as stopping criteria and 0% as crossover probability (Table 4.4). When using not tuned base

learners and S-SGP, the most individually beneficial hyper-parameters are: EDDA as initialization ϵ-LS as selection, Validation as stopping criteria and 100% as crossover probability (Table 4.5). When using tuned base learners and S-GSGP, the most individually beneficial hyper-parameters are: EDDA as initialization, Tournament as selection, TIE as stopping criteria and 100% as crossover probability (Table 4.6). When using tuned base learners and S-SGP, the most individually beneficial hyper-parameters are: EDDA as initialization, ϵ-LS as selection, Validation as stopping criteria and 100% as crossover probability (Table 4.7).

In table 4.8, we can observe that the best S-GSGP is always statistically significantly better than the best S-SGP with tuned base learners. With synthetic problems and not tuned base learners, the best S-GSGP offers a better average rank without statistical evidence. With real-world problems, the best S-SGP offers a slightly better average rank without statistical evidence. For all problems and not tuned base learners, the best S-GSGP is statistically significantly better than the best S-SGP.

Based on the previous performance analysis and statistical assessments, the best observed ensemble system for all problems is using: tuned base learners, S-GSGP as algorithm, EDDA as initialization, Tournament as selection and 100% as crossover probability. TIE offers the best average rank against other stopping criteria without statistical evidence. All the stopping criteria are statistically significantly better than any base learners.

5.2 Interpretation

Knowing that the base learners are tuned for the best ensemble system, we can assume that their predictions are closer to the global optima with a better generalisation.

By using S-GSGP as algorithm, the Geometric Semantic operators induce a unimodal fitness landscape (no local optima), which is not the case for S-SGP. The unimodal fitness landscape of GSGP provides an adequate environment to combine the best features that each base learner has to offer and construct an ensemble. The learning process with GSGP shows a more stable effect on the validation set than GP meaning that what is learned on the training set is relevant (Appendix D).

Geometric Semantic Crossover is more appropriate than Geometric Semantic Mutation since we want to combine different solutions and preserve the original behavior of base learners. With semantic crossover the structures of the parents are transmitted to the offspring which will never perform worse than the worst parent.

In the field of genetic programming, diversity is really important for natural selection. Where RHH only ensures diversity between the individuals of the population, EDDA also makes sure to preserve the ones with the bests features.

The tournament selection maintains diversity due to random selection then selects individuals with the best performance for the complete training set. In comparison, ϵ-LS selects individuals based on their expertise on parts/cases of the training set. That is why it would be harder for ϵ-LS to preserve individuals that present with low error throughout the training set during the evolutionary process.

5.3 Implications

Since this is the first time a benchmark on GSGP and GP for ensemble learning (S-GSGP and S-SGP) is being proposed, it is difficult to relate to previous research mainly focusing and relying on GP as the ensemble learner.

The experiment presented here provides new insight into the relationship between initialization, parent selection and genetic operators for both GSGP and GP. It also demonstrates the possibility of obtaining quality results combining different types of base learners.

5.4 Limitations

The statistical analysis does not allow to suggest a default stopping criterion as part of the best ensemble system. During our experimentation TIE gives the best results.

5.5 Recommendations

Feature selection could have an important role for the S-GSGP ensemble if base learners able to learn from each other can be identified as a initialization step. It would also make sense for them not to be correlated.

In the boxplots in Appendix D we can clearly see that overfitting base learners have a negative impact on the ensemble by way of adding confusion to other base learners with decent generability.

The learning curves show that the ensemble reaches a stable state after only a few generations after which the learning slows down. In order for the learning to continue at that speed we would need to include more base learners or expand the number of individuals in the population.

It would also be interesting to analyze and compare the behavior of a S-GSGP ensemble composed of both, tuned and not tuned base learners. For instance, the best base learner (e.g. random forest, tuned) could learn from a weaker base learner (linear regression, not tuned) aided by less correlation.

Chapter 6

Conclusion

Conclusion

In this thesis, the main questions answered are: By providing a benchmark composed of S-GSGP and S-SGP systems, what is the best observed ensemble system? What are the best hyper-parameters for S-GSGP and S-SGP in different context (All, real-world and synthetic problems)? How does perform S-GSGP against S-SGP in different contexts (different types of problems, with tuned or not tuned base learners)? How does the best ensemble performs perform against its own base learners?

By first establishing the benchmark through 11 different datasets with multiple hyper-parameters and finding the best performing system using S-GSGP we have been able to show that it performs better than S-SGP under multiple conditions.

The key finding of this thesis was establishing a new benchmark for ensemble learning using S-GSGP and S-SGP systems. The best performing system found within this benchmark is using S-GSGP, tuned base learners, EDDA as initialization, Tournament as selection and crossover as only GSO. It is statistically significantly better than its own base learners and the best performing S-SGP, for all problems.

Since there was no previous research in this area, we didn't have any clear expectations only the goal to try and obtain similar or better results with S-GSGP than what was previously found using S-SGP. By using S-GSGP we wanted to maintain the good features of the base learners through the offspring. Such a process was computationally expensive and time-consuming so optimization was required (3.10.1). The

methodology used allowed us to explore different techniques and obtain a clear comparison guiding us to the best setup for S-GSGP. Our results present S-GSGP as a viable solution to ensemble learning.

The next step would be feature selection to gain more insight into which base learners are able to learn from each other. Provided enough computing power, learning about base learner's features could further optimize experimentation, in worst case scenario producing results at least as good as those of the best base learner at any of the runs.

Additionally, the benchmark suggested using tuned base learners to obtain the best results, however, adding not tuned base learners could be beneficial to the learning process.

Thus far, In the field of ensemble learning, no other benchmark has been proposed using S-GSGP and other recent techniques with a significant number of datasets. This new approach helps us better understand how base learners can be combined to produce new solutions by generalizing the relevant features of each base learner.

Bibliography

[1] N. Acosta-Mendoza, A. Morales-Reyes, H. J. Escalante, and A. Gago-Alonso. "Learning to Assemble Classifiers via Genetic Programming." In: *International Journal of Pattern Recognition and Artificial Intelligence* 28 (Dec. 2014). DOI: 10.1142/S0218001414600052.

[2] I. Bakurov, L. Vanneschi, M. Castelli, and F. Fontanella. "EDDA-V2–An Improvement of the Evolutionary Demes Despeciation Algorithm." In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 185–196.

[3] I. Bakurov, M. Castelli, F. Fontanella, and V. Leonardo. "A Regression-Like Classification System for Geometric Semantic Genetic Programming." English. In: Germany: Springer Verlag, Sept. 2019.

[4] I. Bakurov, M. Castelli, L. Vanneschi, and M. J. Freitas. "Supporting medical decisions for treating rare diseases through genetic programming." English. In: *Applications of Evolutionary Computation*. Ed. by P. Kaufmann and P. A. Castillo. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Germany: Springer Verlag, Jan. 2019, pp. 187–203. ISBN: 978-3-030-16691-5. DOI: 10.1007/978-3-030-16692-2_13.

[5] P. Bartashevich, I. Bakurov, S. Mostaghim, and L. Vanneschi. "Evolving PSO algorithm design in vector fields using geometric semantic GP." In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. 2018, pp. 262–263.

[6] P. Bartashevich, I. Bakurov, S. Mostaghim, and L. Vanneschi. "PSO-Based Search Rules for Aerial Swarms Against Unexplored Vector Fields via Genetic Programming." In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2018, pp. 41–53.

[7] E. Bauer and R. Kohavi. "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants." In: *Machine Learning*. 1998, pp. 105–139.

[8] L. Beadle and C. Johnson. "Semantically Driven Crossover in Genetic Programming." In: July 2008, pp. 111 –116. ISBN: 978-1-4244-1822-0. DOI: 10.1109/CEC.2008.4630784.

[9] B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A training algorithm for optimal margin classifiers." In: *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152.

[10] L. Breiman. "Bagging Predictors." In: *Machine Learning* 24.2 (1996), pp. 123–140. ISSN: 1573-0565. DOI: 10.1023/A:1018054314350. URL: https://doi.org/10.1023/A:1018054314350.

[11] L. Breiman. "Random Forests." In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[12] V. Bukhtoyarov and O. Semenkina. "Comprehensive evolutionary approach for neural network ensemble automatic design." In: July 2010, pp. 1–6. ISBN: 978-1-4244-6909-3. DOI: 10.1109/CEC.2010.5586516.

[13] M. Castelli, D. Castaldi, I. Giordani, S. Silva, L. Vanneschi, F. Archetti, and D. Maccagnola. "An efficient implementation of geometric semantic genetic programming for anticoagulation level prediction in pharmacogenetics." In: *Portuguese Conference on Artificial Intelligence*. Springer. 2013, pp. 78–89.

[14] M. Castelli, L. Manzoni, I. Goncalves, L. Vanneschi, L. Trujillo, and S. Silva. "An Analysis of Geometric Semantic Crossover: A Computational Geometry Approach." In: *IJCCI (ECTA)*. 2016, pp. 201–208.

[15] M. Castelli, L. Vanneschi, and A. Popovic. "Parameter evaluation of geometric semantic genetic programming in pharmacokinetics." In: *International Journal of Bio-Inspired Computation* 8.1 (2016), pp. 42–50.

[16] M. Castelli, L. Vanneschi, L. Manzoni, and A. Popovic. "Semantic genetic programming for fast and accurate data knowledge discovery." In: *Swarm and Evolutionary Computation* 26 (2016), pp. 1 –7.

[17] C. Darwin. *On the origins of species by means of natural selection*. London: Murray, 1859.

[18] T. G. Dietterich. "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization." In: *Bagging, boosting, and randomization. Machine Learning*. 1998, pp. 139–157.

[19] R. Eberhart and J. Kennedy. "A new optimizer using particle swarm theory." In: *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. 1995, pp. 39–43. DOI: 10.1109/MHS.1995.494215.

[20] O. Flasch, M. Friese, M. Zaefferer, T. Bartz-Beielstein, and J. Branke. "Learning Model-Ensemble Policies with Genetic Programming." In: (Jan. 2015).

[21] Y. Freund and R. E. Schapire. "Experiments with a New Boosting Algorithm." In: *IN PROCEEDINGS OF THE THIRTEENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*. Morgan Kaufmann, 1996, pp. 148–156.

[22] *Genetic Programming and Evolvable Machines (8): Genetic programming for computational pharmacokinetics in drug discovery and development*. http://kdbio.inesc-id.pt/~sara/gptp2013/ppb.txt. Accessed: 05.09.2019. 2007.

[23] I. Goncalves, S. Silva, C. M. Fonseca, and M. Castelli. "Unsure when to Stop?: Ask Your Semantic Neighbors." In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '17. Berlin, Germany: ACM, 2017, pp. 929–936. ISBN: 978-1-4503-4920-8. DOI: 10.1145/3071178.3071328. URL: http://doi.acm.org/10.1145/3071178.3071328.

[24] I. Gonçalves, S. Silva, and C. Fonseca. "Semantic Learning Machine: A Feedforward Neural Network Construction Algorithm Inspired by Geometric Semantic Genetic Programming." In: Sept. 2015, pp. 280–285. ISBN: 978-3-319-23484-7. DOI: 10.1007/978-3-319-23485-4_28.

[25] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[26] U. Johansson, T. Löfström, R. König, and L Niklasson. "Building Neural Network Ensembles using Genetic Programming." In: Jan. 2006, pp. 1260 –1265. DOI: 10.1109/IJCNN.2006.246836.

[27] C. Johnson. "Deriving Genetic Programming Fitness Properties by Static Analysis." In: vol. 2278. Apr. 2002, pp. 298–307. DOI: 10.1007/3-540-45984-7_29.

[28] C. Johnson. "Genetic Programming with Fitness Based on Model Checking." In: vol. 4445. Apr. 2007, pp. 114–124. DOI: 10.1007/978-3-540-71605-1_11.

[29] P. Kordík and J. Cerny. "Building predictive models in two stages with meta-learning templates optimized by genetic programming." In: Dec. 2014. DOI: 10.1109/CIEL.2014.7015740.

[30] J. R. Koza. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Tech. rep. Stanford, CA, USA, 1990.

[31] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA, USA: MIT Press, 1994. ISBN: 0-262-11189-6.

[32] J. R. Koza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Norwell, MA, USA: Kluwer Academic Publishers, 2003. ISBN: 1402074468.

[33] J. R. Koza, D. Andre, F. H. Bennett, and M. A. Keane. *Genetic Programming III: Darwinian Invention & Problem Solving*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN: 1558605436.

[34] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[35] K. Krawiec and B. Wieloch. "Analysis of Semantic Modularity for Genetic Programming." In: *Foundations of Computing and Decision Sciences* 34 (Jan. 2009), pp. 265–285.

[36] W. La Cava, L. Spector, and K. Danai. "Epsilon-Lexicase Selection for Regression." In: July 2016, pp. 741–748. DOI: 10.1145/2908812.2908898.

[37] *Least Angle Regression, Lasso and Forward Stagewise in R: Diabetes dataset*. https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html. Accessed: 05.09.2019. 2013.

[38] A. Moraglio, K. Krawiec, and C. G. Johnson. "Geometric semantic genetic programming." In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2012, pp. 21–31.

[39] Q. U. Nguyen, N. Hoai, M. O'Neill, R. McKay, and E. Galván-López. "Semantically-based crossover in genetic programming: Application to real-valued symbolic regression." In: *Genetic Programming and Evolvable Machines* 12 (June 2011), pp. 91–119. DOI: 10.1007/s10710-010-9121-2.

[40] T. P. Pawlak and K. Krawiec. "Semantic Geometric Initialization." In: *Genetic Programming*. Ed. by M. I. Heywood, J. McDermott, M. Castelli, E. Costa, and K. Sim. Cham: Springer International Publishing, 2016, pp. 261–277. ISBN: 978-3-319-30668-1.

[41] T. P. Pawlak, B. Wieloch, and K. Krawiec. "Review and comparative analysis of geometric semantic crossovers." In: *Genetic Programming and Evolvable Machines* 16 (2015), pp. 351–386.

[42] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008. ISBN: 1409200736, 9781409200734.

[43] R. Polikar. "Polikar, R.: Ensemble based systems in decision making. IEEE Circuit Syst. Mag. 6, 21-45." In: *Circuits and Systems Magazine, IEEE* 6 (Oct. 2006), pp. 21 –45. DOI: 10.1109/MCAS.2006.1688199.

[44] A. Re. "Universal Genetic Programming: a Meta Learning Approach based on Semantics." Doctoral dissertation. NOVA Information Management School, Universidade Nova de Lisboa, 2018.

[45] L. Rokach. "Ensemble-Based Classifiers." In: *Artif. Intell. Rev.* 33 (Feb. 2010), pp. 1–39. DOI: 10.1007/s10462-009-9124-7.

[46] S. Samarasinghe. *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. Auerbach publications, 2016.

[47] L. Spector. "Assessment of Problem Modality by Differential Performance of Lexicase Selection in Genetic Programming: A Preliminary Report." In: *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '12. Philadelphia, Pennsylvania, USA: ACM, 2012, pp. 401–408. ISBN: 978-1-4503-1178-6. DOI: 10.1145/2330784.2330846. URL: http://doi.acm.org/10.1145/2330784.2330846.

[48] *StatLib Datasets Archive: Boston house prices dataset*. http://lib.stat.cmu.edu/datasets/boston. Accessed: 05.09.2019. 1980.

[49] A. Tsanas, M. A. Little, P. E. McSharry, and L. O. Ramig. "Accurate Telemonitoring of Parkinson's Disease Progression by Noninvasive Speech Tests." In: *IEEE Transactions on Biomedical Engineering* 57.4 (2010), pp. 884–893. ISSN: 1558-2531. DOI: 10.1109/TBME.2009.2036000.

[50] L. Vanneschi. "An Introduction to Geometric Semantic Genetic Programming." In: 663 (Aug. 2017), pp. 3–42. DOI: 10.1007/978-3-319-44003-3_1.

[51] L. Vanneschi, M. Castelli, L. Manzoni, and S. Silva. "A new implementation of geometric semantic GP and its application to problems in pharmacokinetics." In: *European Conference on Genetic Programming*. Springer. 2013, pp. 205–216.

[52] L. Vanneschi, M. Castelli, and S. Silva. "A survey of semantic methods in genetic programming." In: *Genetic Programming and Evolvable Machines* 15.2 (2014), pp. 195–214.

[53] L. Vanneschi, S. Silva, M. Castelli, and L. Manzoni. "Geometric semantic genetic programming for real life applications." In: *Genetic programming theory and practice xi*. Springer, 2014, pp. 191–209.

[54] L. Vanneschi, I. Bakurov, and M. Castelli. "An initialization technique for geometric semantic GP based on demes evolution and despeciation." In: *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE. 2017, pp. 113–120.

[55] D. H. Wolpert. "Stacked Generalization." In: *Neural Networks* 5 (1992), pp. 241–259.

[56] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Vol. 14. Jan. 2012. ISBN: 9781439830031. DOI: 10.1201/b12207.

Appendix A

Appendix - Ensemble Workflow

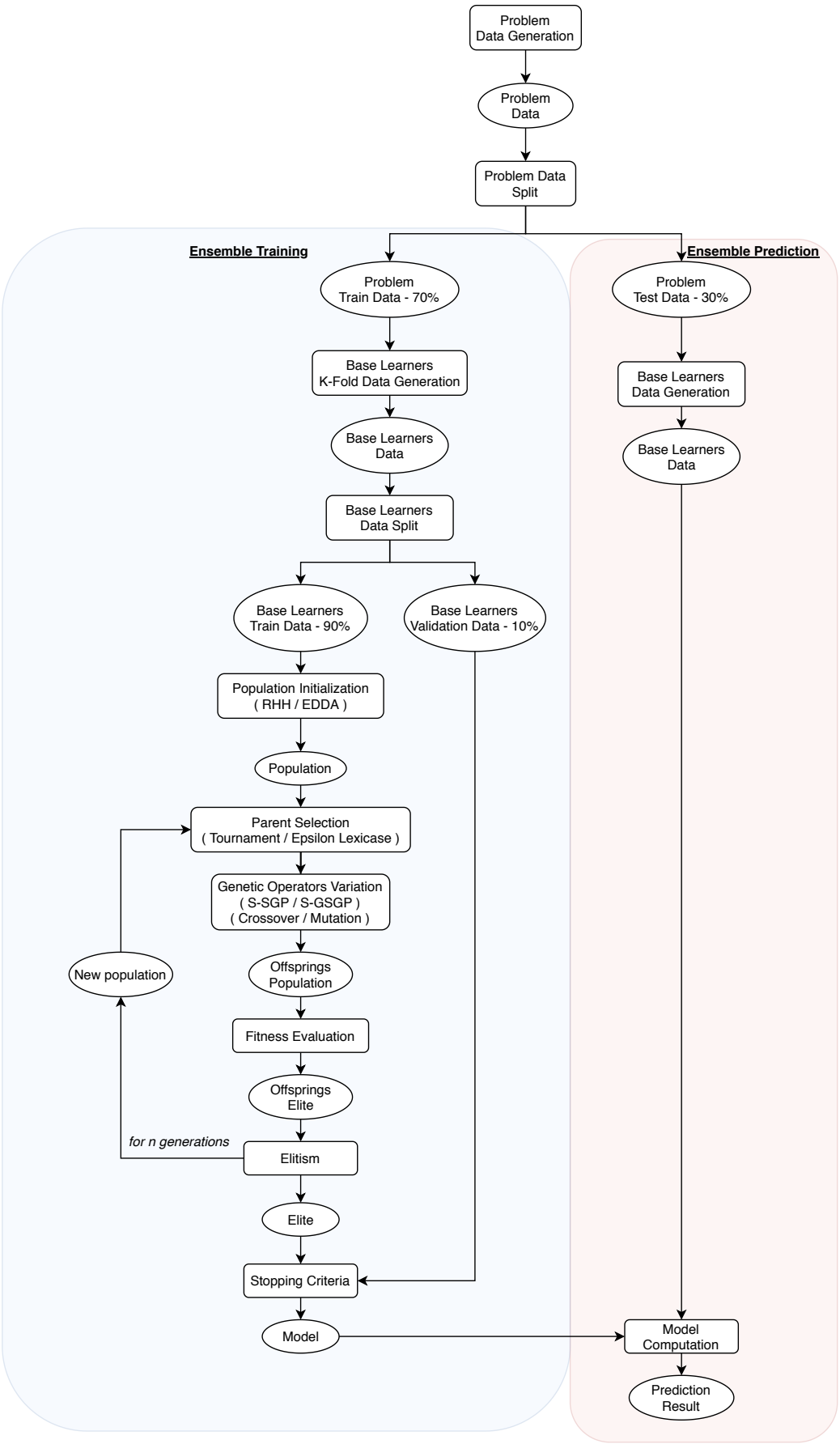

Figure A.1: Ensemble Workflow

Appendix B

Appendix - Performance by dataset

Problem	Algo	Initialization	Selection	Tuned BLs	P(Cr)	Stopping Crierion	RMSE
Branin	S-GSGP	RHH	ϵ-LS	TRUE	0.0	TIE	11.985212
	S-GSGP	RHH	ϵ-LS	TRUE	0.0	Nºgenerations	12.111562
	S-GSGP	RHH	TS	FALSE	0.8	TIE	12.240556
	S-GSGP	RHH	ϵ-LS	TRUE	0.0	EDV	12.311387
	S-GSGP	RHH	TS	FALSE	0.8	EDV	12.381343
Discus	S-GSGP	EDDA	ϵ-LS	TRUE	0.0	EDV	0.275822
	S-GSGP	EDDA	TS	TRUE	1.0	Validation	0.275966
	S-GSGP	EDDA	TS	FALSE	0.0	EDV	0.277260
	S-GSGP	EDDA	ϵ-LS	FALSE	0.0	EDV	0.277397
	S-GSGP	EDDA	TS	TRUE	1.0	TIE	0.277571
Griewank	S-GSGP	RHH	ϵ-LS	FALSE	0.0	TIE	0.512961
	S-SGP	RHH	ϵ-LS	FALSE	1.0	Nºgenerations	0.513833
	S-SGP	RHH	ϵ-LS	FALSE	1.0	Validation	0.513895
	S-GSGP	EDDA	TS	FALSE	0.0	EDV	0.514111
	S-SGP	EDDA	TS	FALSE	1.0	TIE	0.514439
Kotanchek	S-GSGP	EDDA	TS	FALSE	0.8	TIE	0.028731
	S-GSGP	EDDA	TS	TRUE	1.0	Nºgenerations	0.028976
	S-GSGP	EDDA	ϵ-LS	FALSE	1.0	Nºgenerations	0.029052
	S-GSGP	EDDA	TS	FALSE	1.0	TIE	0.029296
	S-GSGP	EDDA	TS	FALSE	1.0	Nºgenerations	0.029296
Mexicanhat	S-SGP	EDDA	TS	TRUE	1.0	Nºgenerations	0.000636
	S-SGP	EDDA	TS	TRUE	0.8	Nºgenerations	0.000640
	S-SGP	EDDA	TS	TRUE	0.8	Validation	0.000643
	S-SGP	EDDA	ϵ-LS	TRUE	0.2	Nºgenerations	0.000648
	S-SGP	EDDA	TS	TRUE	0.2	Nºgenerations	0.000649
Rastrigin	S-SGP	EDDA	ϵ-LS	TRUE	0.0	TIE	8.860859
	S-SGP	EDDA	ϵ-LS	TRUE	0.0	EDV	8.891697
	S-GSGP	EDDA	ϵ-LS	TRUE	0.0	Nºgenerations	8.906953
	S-SGP	EDDA	ϵ-LS	TRUE	0.2	EDV	8.907992
	S-GSGP	EDDA	ϵ-LS	TRUE	0.0	TIE	8.913248
Weierstrass	S-GSGP	EDDA	ϵ-LS	FALSE	1.0	TIE	0.428237
	S-GSGP	EDDA	ϵ-LS	FALSE	1.0	Validation	0.428718
	S-GSGP	EDDA	ϵ-LS	TRUE	0.0	TIE	0.429958
	S-GSGP	EDDA	ϵ-LS	TRUE	0.0	EDV	0.429958
	S-GSGP	EDDA	ϵ-LS	FALSE	1.0	Nºgenerations	0.431224

Table B.1: Avg RMSE - Top 5 - Synthetic datasets

Problem	Algo	Initialization	Selection	Tuned BLs	P(Cr)	Stopping Criteria	RMSE
Diabetes	S-SGP	edda	ϵ-LS	TRUE	1.0	TIE	56.456302
	S-GSGP	edda	TS	TRUE	0.0	EDV	56.456779
	S-GSGP	edda	TS	TRUE	0.0	Validation	56.630922
	S-GSGP	edda	TS	TRUE	1.0	Validation	56.699151
	S-GSGP	edda	TS	TRUE	0.0	TIE	56.713499
Ppb	S-GSGP	edda	ϵ-LS	FALSE	0.0	Validation	40.141465
	S-GSGP	edda	ϵ-LS	FALSE	0.0	EDV	40.149956
	S-GSGP	edda	ϵ-LS	FALSE	0.0	TIE	40.286283
	S-GSGP	edda	ϵ-LS	FALSE	0.0	Nºgenerations	40.318188
	S-SGP	edda	TS	TRUE	0.0	EDV	43.314997
Boston	S-GSGP	edda	ϵ-LS	TRUE	0.8	EDV	3.263451
	S-GSGP	edda	ϵ-LS	TRUE	0.8	Validation	3.263568
	S-GSGP	edda	ϵ-LS	TRUE	0.8	Nºgenerations	3.266919
	S-GSGP	edda	TS	TRUE	1.0	TIE	3.270275
	S-GSGP	edda	TS	TRUE	1.0	Validation	3.283117
Parkinson	S-GSGP	edda	TS	TRUE	1.0	Validation	18687.751581
	S-GSGP	edda	TS	TRUE	1.0	Nºgenerations	18711.120370
	S-GSGP	edda	TS	TRUE	1.0	TIE	18711.120370
	S-GSGP	edda	TS	TRUE	0.8	Nºgenerations	18724.871696
	S-GSGP	edda	TS	TRUE	0.8	TIE	18724.871696

Table B.2: Avg RMSE - Top 5 - Real-world datasets

Problem	TIE	Validation	EDV	Nºgenerations	RF	SVM	MLP	LR
Boston	3.270275	3.283117	3.298656	3.289512	3.525224	5.710099	4.287523	4.952785
Branin	12.852443	12.795747	12.964059	12.852443	16.205687	44.906725	48.227606	48.408719
Diabetes	57.156454	56.699151	57.121728	57.156454	63.707485	55.312513	60.313119	88.118727
Discus	0.277571	0.275966	0.286831	0.311625	1.450E+13	3.425E+14	4.862E+14	4.895E+14
Griewank	0.516116	0.516346	0.518276	0.517555	11.092327	17.255833	71.999513	38.512383
Kotanchek	0.030122	0.029670	0.029589	0.028976	0.042039	0.076905	0.144288	0.157398
Mexican Hat	0.000851	0.000858	0.000867	0.000860	0.002480	0.000964	0.195627	0.009385
Parkinson	1.871E+04	1.869E+04	1.882E+04	1.871E+04	1.854E+04	1.873E+04	2.176E+04	1.033E+05
Ppb	578.871932	1938.769171	288.064776	578.871936	28.908556	76.618709	136.099117	3.682E+05
Rastrigin	9.061723	9.103786	9.061723	9.061723	8.940424	10.877383	20.944968	14.734824
Weierstrass	0.440366	0.451444	0.449200	0.449200	0.481757	0.592406	0.574611	1.162233

Table B.3: Avg RMSE - Best performing ensemble system for all problems and BLs: [Tuned BLs, S-GSGP, EDDA, TS, P(C) 100%]

APPENDIX C

Appendix - Boxplots and Learning curves

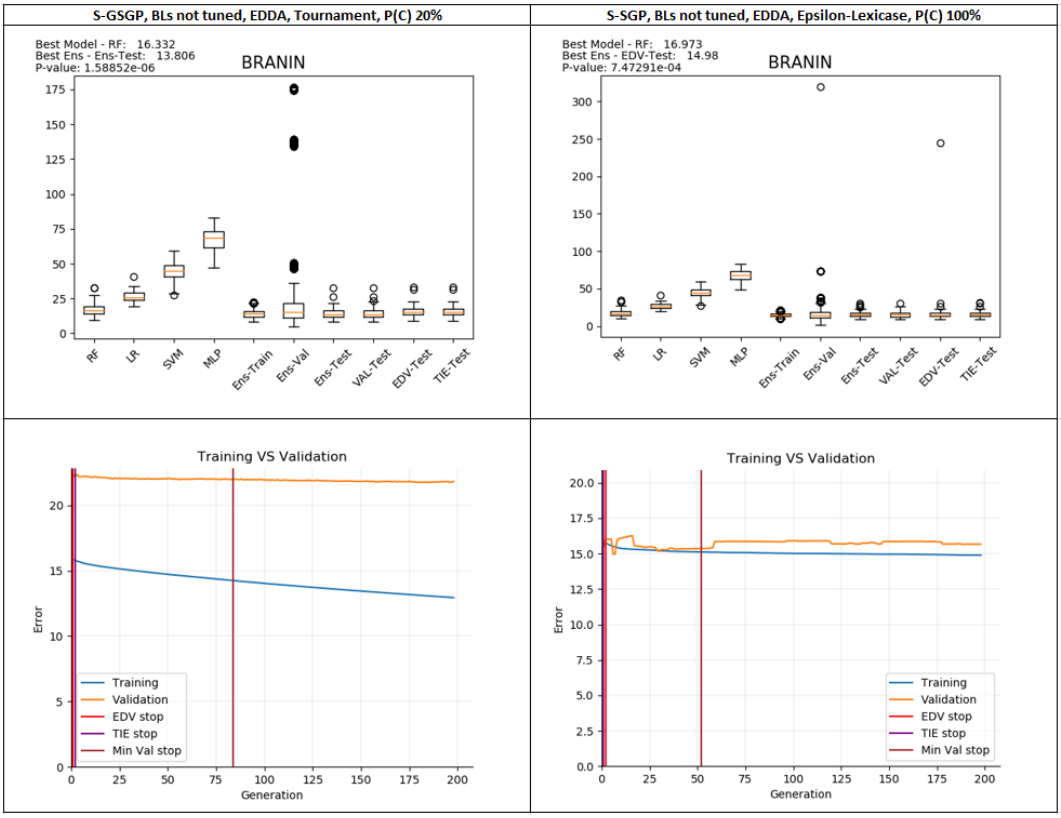

Figure C.1: Branin - BLs not tuned - Boxplots and Learning curves

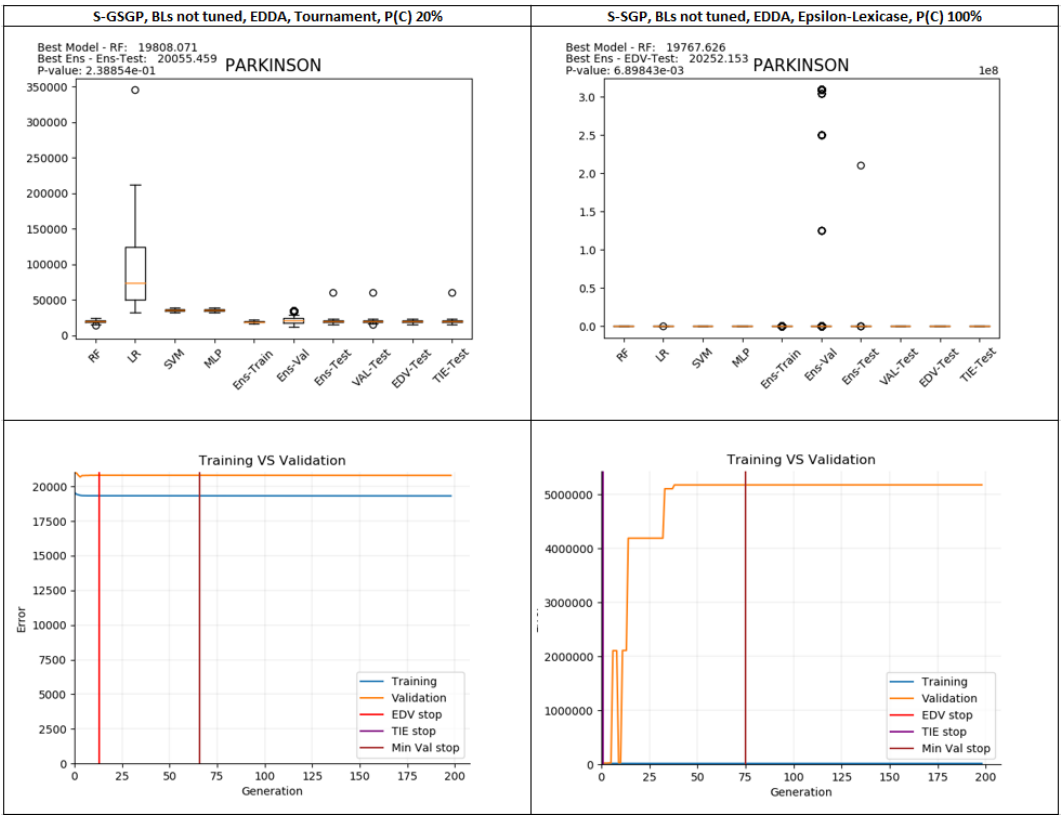

Figure C.2: Parkinson - BLs not tuned - Boxplots and Learning curves

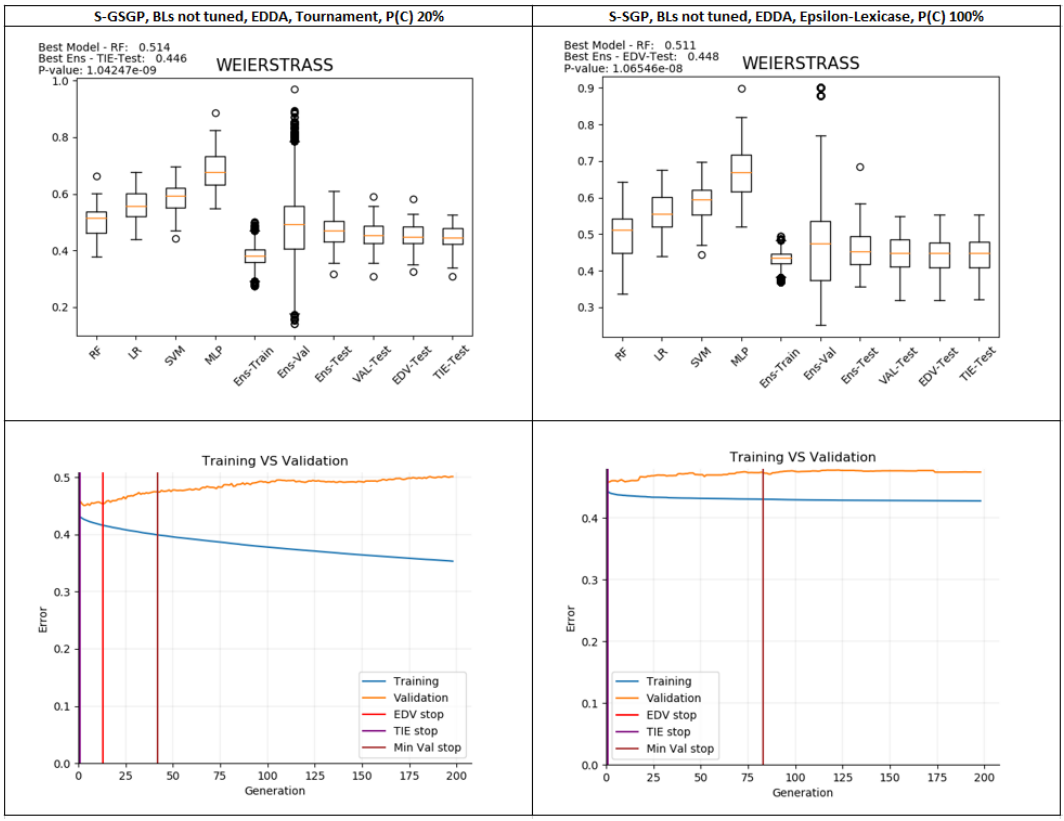

Figure C.3: Weierstrass - BLs not tuned - Boxplots and Learning curves

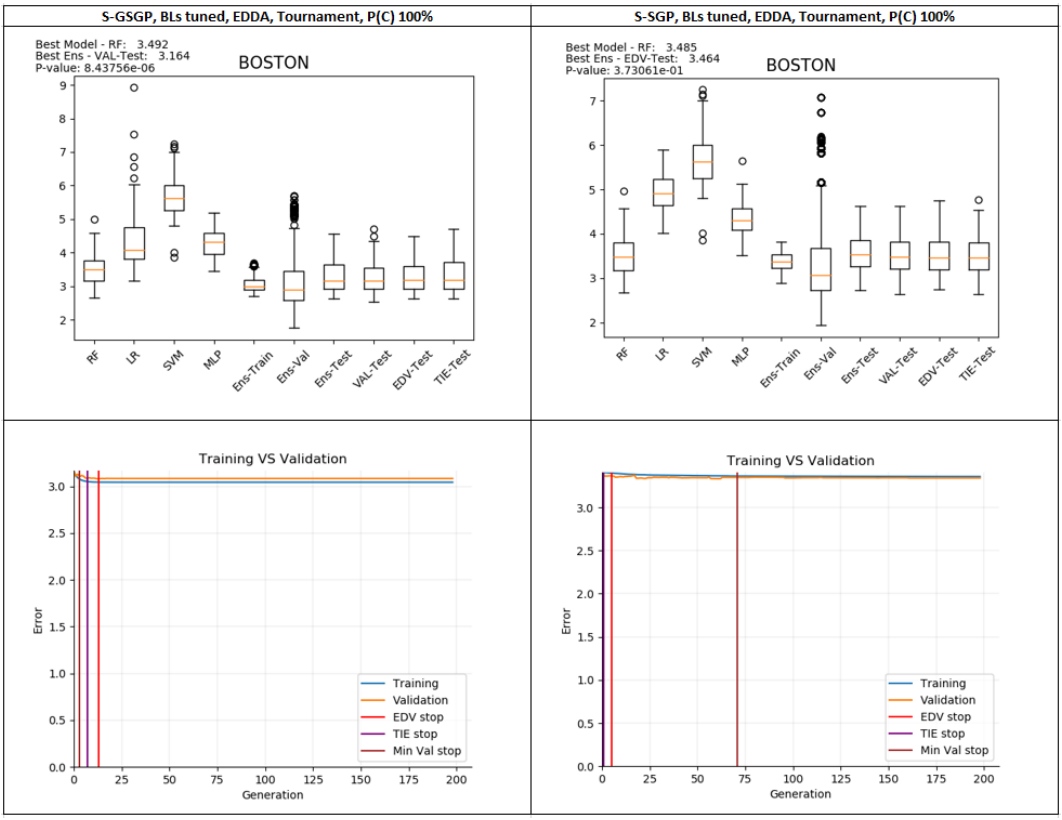

Figure C.4: Boston - BLs tuned - Boxplots and Learning curves

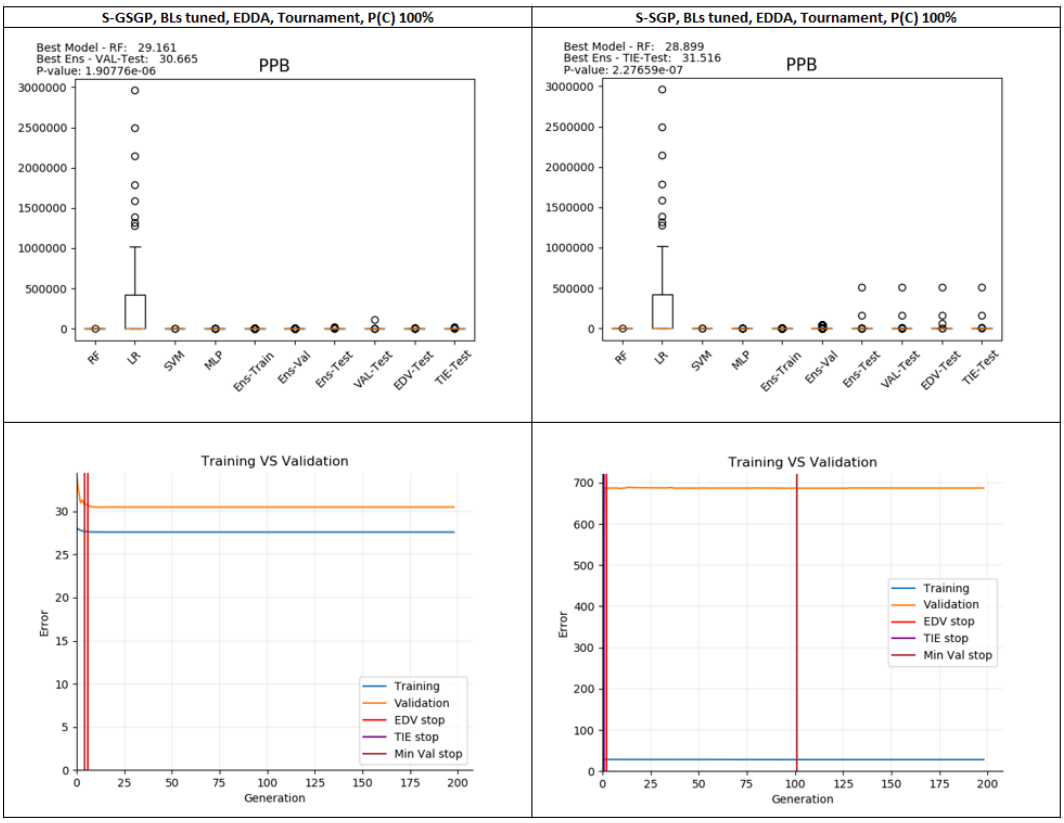

Figure C.5: Ppb - BLs tuned - Boxplots and Learning curves

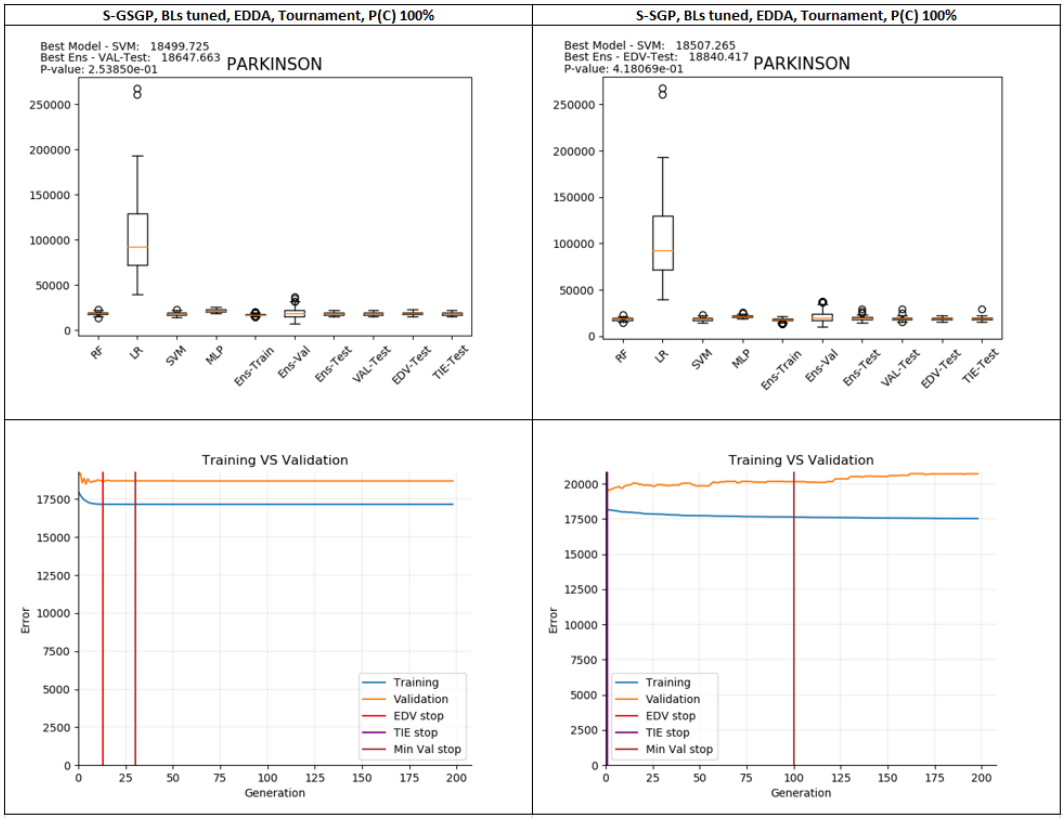

Figure C.6: Parkinson - BLs tuned - Boxplots and Learning curves

Appendix D

Appendix - Tuned Base Learners Hyper-parameters

Problems	bootstrap	max_depth	max_features	n_estimators
Ackley	TRUE	100	sqrt	10
Branin	TRUE	None	auto	50
Discus	TRUE	100	auto	100
Griewank	FALSE	100	sqrt	100
Kotanchek	TRUE	10	auto	100
Mexicanhat	FALSE	100	sqrt	100
Rastrigin	TRUE	50	sqrt	100
Rosenbrock	TRUE	10	auto	100
Sphere	FALSE	100	sqrt	100
Weierstrass	FALSE	50	sqrt	50
Boston	random	412	auto	193
Diabetes	random	918	sqrt	581
Ppb	best	826	log2	471
Energy	best	990	log2	256
Parkinson	random	348	sqrt	124
Ld50	random	827	sqrt	437

Table D.1: Random Forest Regression - Tuned Hyper-parameters by problem

Problems	degree	epsilon	kernel
Ackley	2	0.01	rbf
Branin	2	0.1	rbf
Discus	4	0.001	poly
Griewank	2	0.1	poly
Kotanchek	2	0.01	rbf
Mexicanhat	2	0.001	rbf
Rastrigin	2	0.1	poly
Rosenbrock	4	0.1	poly
Sphere	2	0.01	rbf
Weierstrass	2	0.01	rbf
Boston	2	0.01	rbf

Table D.2: Support Vector Regressor - Synthetic Problems - Tuned Hyper-parameters by problem

Problems	epsilon	C	loss	dual	fit_intercept	max_iter
Diabetes	0.09949	0.204525	squared_epsilon_insensitive	TRUE	TRUE	655
Ppb	0.089958	1.901082	epsilon_insensitive	TRUE	FALSE	243
Energy	0.007113	0.70489	squared_epsilon_insensitive	TRUE	TRUE	439
Parkinson	0.053021	1.912587	squared_epsilon_insensitive	TRUE	TRUE	688
Ld50	0.075916	0.962089	squared_epsilon_insensitive	TRUE	TRUE	911

Table D.3: Linear Support Vector Regressor - Real-world Problems - Tuned Hyper-parameters by problem

Problems	alpha	hidden_layer_sizes	learning_rate_init
Ackley	0.01	8	0.001
Branin	0.1	6	0.01
Discus	0.001	6	0.1
Griewank	0.0001	4	0.001
Kotanchek	0.01	10	0.01
Mexicanhat	0.0001	9	0.001
Rastrigin	0.01	9	0.001
Rosenbrock	0.0001	10	0.01
Sphere	0.001	10	0.001
Weierstrass	0.0001	8	0.1
Boston	0.067743	21	0.005255
Diabetes	0.01	42	0.01
Ppb	0.01	31	1.00E-05
Energy	0.0001	53	0.01
Parkinson	0.1	15	0.1
Ld50	0.0001	93	0.01

Table D.4: Multi-layer Perceptron Regressor - Tuned Hyper-parameters by problem

Problems	degree	normalize	fit_intercept
Ackley	2	TRUE	FALSE
Branin	4	FALSE	TRUE
Discus	2	TRUE	FALSE
Griewank	2	FALSE	TRUE
Kotanchek	4	TRUE	FALSE
Mexicanhat	4	FALSE	TRUE
Rastrigin	2	FALSE	TRUE
Rosenbrock	4	FALSE	TRUE
Sphere	2	FALSE	TRUE
Weierstrass	2	TRUE	TRUE
Boston	2	FALSE	TRUE
Diabetes	2	TRUE	TRUE
Ppb	2	FALSE	TRUE
Energy	2	TRUE	TRUE
Parkinson	3	FALSE	TRUE
Ld50	3	TRUE	TRUE

Table D.5: Linear Regression - Tuned Hyper-parameters by problem

Annex I

Annex S-GSGP and Base Learners Graphs

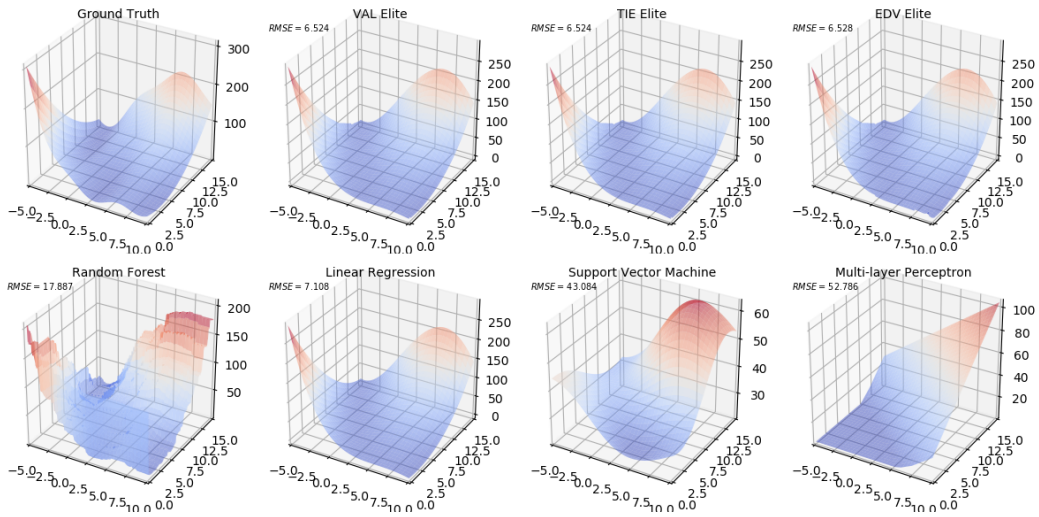

Figure I.1: Branin - S-GSGP and Base Learners 3D Graphs

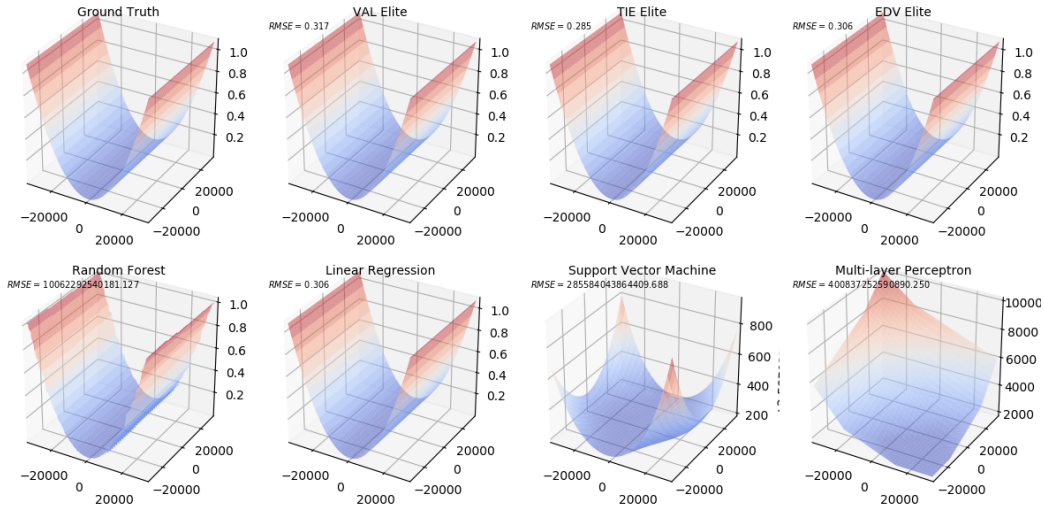

Figure I.2: Discus - S-GSGP and Base Learners 3D Graphs

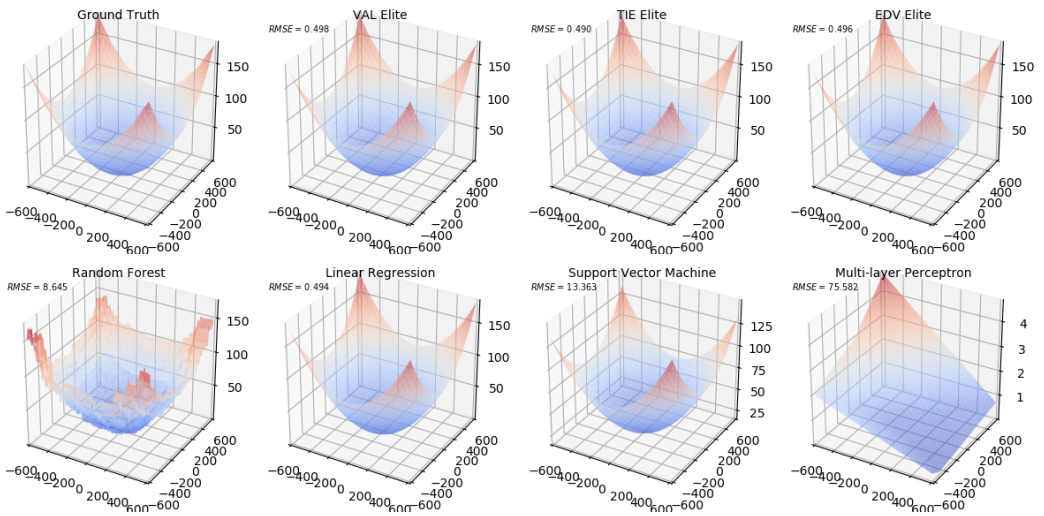

Figure I.3: Griewank - S-GSGP and Base Learners 3D Graphs

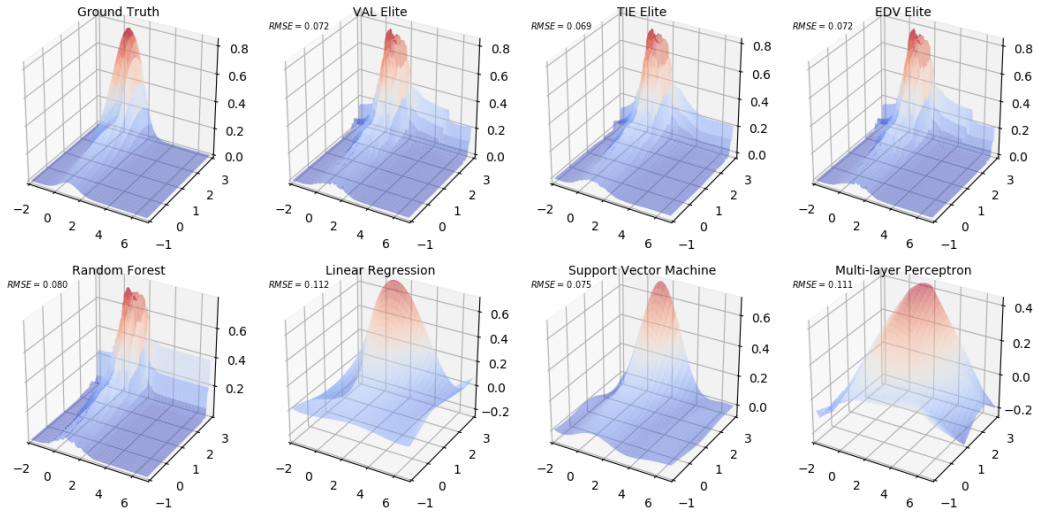

Figure I.4: Kotanchek - S-GSGP and Base Learners 3D Graphs

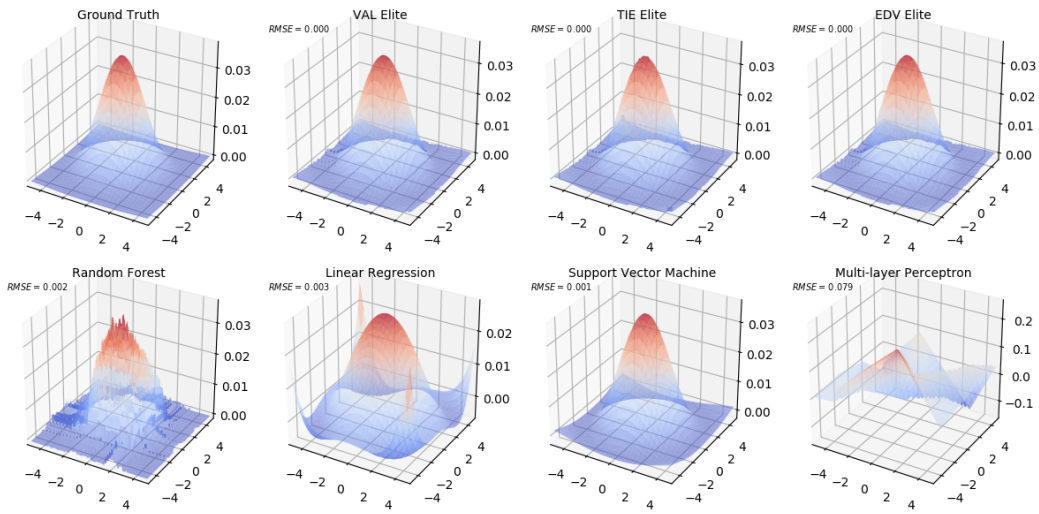

Figure I.5: Mexican Hat - S-GSGP and Base Learners 3D Graphs

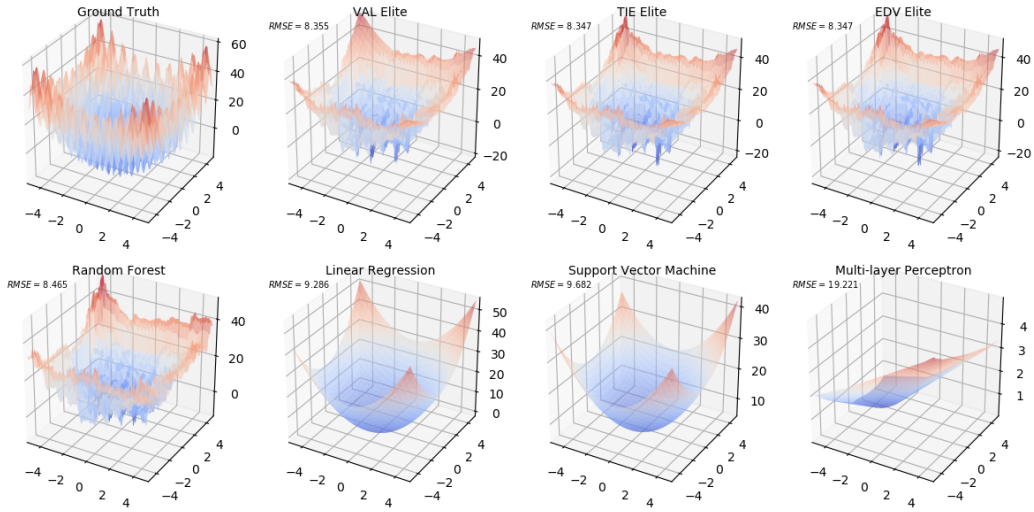

Figure I.6: Rastrigin - S-GSGP and Base Learners 3D Graphs

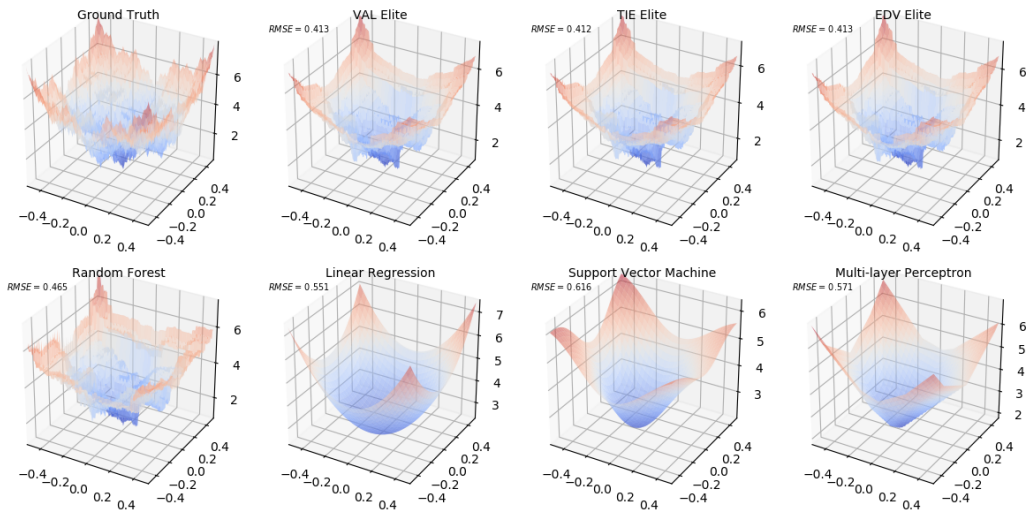

Figure I.7: Weierstrass - S-GSGP and Base Learners 3D Graphs

ANNEX II

Annex - Best ensemble solution's string

In this annex, few lines of each synthetic problem solution obtained with the best performing ensemble.

II.1 Branin - Solution's string

add(mul(lf(sub(mul(add(mlp, mlp), deci[-0.34](svm, -0.433, -0.480)), mul(rf, -0.230))), add(add(add(add(mul(lf(min(avg(deci[-0.34](sub(0.209, lr), avg(rf, -0.543), mul(rf, svm)), add(add(mlp, -0.599), mul(-0.683, -0.834))), sub(sub(mul(-0.820, -0.169), deci[-0.34](lr, rf, 0.549)), min(sub(rf, -0.619), avg(svm, lr))))), avg(avg(min(deci[-0.34](rf, rf, rf), deci[-0.34](-0.890, mlp, 0.804)), min(avg(-0.624, svm), avg(lr, svm))), deci[-0.34](avg(mul(-0.364, rf), deci[-0.34](-0.443, rf, mlp)), deci[-0.34](deci[-0.34](mlp, lr, svm), sub(-0.605, mlp), add(lr, svm)), deci[-0.34](min(svm, svm), min(-0.346, 0.807), avg(rf, lr))))), mul(sub(1.000, lf(min(avg(deci[-0.34](sub(0.209, lr), avg(rf, -0.543), mul(rf, svm)), add(add(mlp, -0.599), mul(-0.683, -0.834))), sub(sub(mul(-0.820, -0.169), deci[-0.34](lr, rf, 0.549)), min(sub(rf, -0.619), avg(svm, lr)))))), sub(avg(lr, lr), min(svm, 0.307)))), ...

II.2 Discus - Solution's string

add(mul(lf(deci[0.6](mul(mul(rf, deci[-0.14](-0.857, svm, -0.311)), min(svm, avg(svm, lr))), mul(min(deci[-0.14](0.180, lr, rf), min(rf, svm)), sub(mlp, sub(mlp, mlp))), sub(-0.577, min(min(lr, rf), -0.600)))), sub(lr, 0.121)), mul(sub(1.000, lf(deci[0.6](mul(mul(rf, deci[-0.14](-0.857, svm, -0.311)), min(svm, avg(svm, lr))), mul(min(deci[-0.14](0.180, lr, rf), min(rf, svm)), sub(mlp, sub(mlp, mlp))), sub(-0.577, min(min(lr, rf), -0.600))))),

add(deci[-0.14](sub(0.534, -0.567), add(mlp, 0.695), sub(lr, -0.396)), min(mlp, -0.423))))

II.3 Griewank - Solution's string

add(mul(lf(sub(avg(svm, lr), add(-0.736, svm))), add(add(add(mul(lf(deci[0.68](deci[-0.42](sub(-0.702, lr), avg(-0.204, 0.646), deci[-0.42](0.461, mlp, -0.757)), mul(mul(lr, -0.760), min(rf, -0.820)), deci[-0.42](lr, avg(svm, rf), deci[-0.42](rf, -0.074, svm)))), sub(deci[-0.42](min(rf, svm), lr, lr), mul(min(mlp, 0.733), sub(svm, svm)))), mul(sub(1.000, lf(deci[0.68](deci[-0.42](sub(-0.702, lr), avg(-0.204, 0.646), deci[-0.42](0.461, mlp, -0.757)), mul(mul(lr, -0.760), min(rf, -0.820)), deci[-0.42](lr, avg(svm, rf), deci[-0.42](rf, -0.074, svm))))), deci[0.75](deci[-0.42](avg(mlp, lr), min(mlp, lr), deci[-0.42](mlp, svm, rf)), sub(deci[-0.42](rf, lr, rf), avg(0.614, -0.756)), deci[-0.42](min(svm, -0.618), min(mlp, lr), add(lr, 0.805))))), mul(0.181, sub(lf(avg(sub(svm, 0.314), min(rf, mlp))), lf(mul(rf, avg(mul(mlp, svm), -0.792)))))), mul(0.924, sub(lf(avg(mul(svm, add(svm, svm)), avg(add(lr, lr), mlp))), ...

II.4 Kotanchek - Solution's string

add(mul(lf(sub(deci[-0.53](avg(avg(rf, lr), 0.175), mlp, mlp), add(mul(min(-0.692, 0.356), deci[-0.53](svm, -0.760, lr)), min(deci[-0.53](0.963, mlp, 0.055), avg(-0.597, svm))))), add(mul(lf(deci[0.07](mul(sub(mlp, 0.186), rf), add(svm, avg(0.231, lr)), -0.063)), add(add(mul(lf(mul(avg(min(svm, sub(-0.259, -0.662)), add(sub(0.936, lr), mul(0.074, lr))), sub(mlp, avg(rf, 0.659)))), mul(rf, add(svm, 0.785))), mul(sub(1.000, lf(mul(avg(min(svm, sub(-0.259, -0.662)), add(sub(0.936, lr), mul(0.074, lr))), sub(mlp, avg(rf, 0.659))))), mul(rf, add(svm, 0.785)))), mul(0.647, sub(lf(mul(sub(svm, -0.534), deci[-0.22](svm, svm, 0.671))), lf(deci[-0.69](deci[-0.22](min(lr, 0.983), lr, lr), rf, min(sub(-0.178, -0.362), svm))))))), mul(sub(1.000, lf(deci[0.07](mul(sub(mlp, 0.186), rf), add(svm, avg(0.231, lr)), -0.063))), add(mul(lf(add(deci[-0.22](deci[-0.22](0.901, -0.300, mlp), lr, add(svm, rf)), rf)), add(add(mul(lf(sub(mul(add(-0.948, mlp), deci[-0.22](-0.600, rf, rf)), avg(add(0.956, mlp), ...

II.5 Mexican Hat - Solution's string

add(mul(lf(deci[0.18](deci[-0.26](deci[-0.26](deci[-0.26](lr, -0.554, mlp), min(-0.543, 0.836), deci[-0.26](-0.609, rf, 0.140)), min(deci[-0.26](-0.946, 0.268, rf), deci[-0.26](mlp, mlp, 0.139)), 0.612), mlp, add(mlp, rf))), add(mul(lf(avg(svm, sub(avg(rf, 0.445), avg(lr, rf)))), add(mul(lf(add(svm, svm)), add(mul(lf(deci[0.5](avg(avg(avg(-0.315, 0.568), deci[-0.49](-0.675, mlp, svm)), sub(lr, -0.105)), rf, sub(deci[-0.49](lr, avg(0.570, mlp), min(rf, lr)), deci[-0.49](svm, 0.867, add(0.769, lr))))), add(mul(lf(deci[0.39](mul(0.705, -0.028), add(mlp, svm), min(0.482, svm))), add(mul(lf(deci[-0.32](min(mul(-0.481, lr), mul(-0.977, lr)), sub(add(svm, rf), mul(mlp, rf)), avg(min(svm, lr), mul(svm, mlp)))),

add(svm, mul(mul(lr, sub(svm, 0.626)), 0.715))), mul(sub(1.000, lf(deci[-0.32](min(mul(-0.481, lr), mul(-0.977, lr)), sub(add(svm, rf), mul(mlp, rf)), avg(min(svm, lr), mul(svm, mlp))))), add(avg(lr, svm), svm)))), mul(sub(1.000, lf(deci[0.39](mul(0.705, -0.028), add(mlp, svm), min(0.482, svm)))), ...

II.6 Rastrigin - Solution's string

add(mul(lf(sub(min(sub(mul(0.706, rf), add(mlp, 0.198)), mlp), avg(0.649, sub(sub(svm, mlp), mul(lr, mlp))))), add(mul(lf(add(avg(mul(lr, mlp), sub(svm, svm)), min(avg(lr, mlp), mul(-0.880, lr)))), add(mul(lf(mul(svm, min(svm, rf))), add(mul(lf(deci[-0.59](avg(mlp, min(lr, deci[0.38](rf, lr, rf))), mul(deci[0.38](deci[0.38](mlp, mlp, lr), 0.900, min(rf, lr)), mul(mul(0.419, rf), deci[0.38](rf, 0.614, 0.308))), sub(mul(deci[0.38](rf, lr, 0.901), min(mlp, rf)), add(add(mlp, lr), min(mlp, lr))))), add(mul(lf(add(deci[0.38](add(deci[0.38](0.466, -0.545, mlp), add(lr, 0.785)), mul(mul(svm, 0.471), min(svm, svm)), -0.032), 0.705)), sub(add(deci[0.38](svm, 0.308, -0.111), rf), mlp)), mul(sub(1.000, lf(add(deci[0.38](add(deci[0.38](0.466, -0.545, mlp), add(lr, 0.785)), mul(mul(svm, 0.471), min(svm, svm)), -0.032), 0.705))), avg(sub(rf, 0.021), deci[0.38](-0.843, mlp, lr))))), mul(sub(1.000, lf(deci[-0.59](avg(mlp, min(lr, deci[0.38](rf, lr, rf))), ...

II.7 Weierstrass - Solution's string

add(mul(lf(min(add(mul(rf, -0.632), avg(mlp, lr)), add(add(svm, 0.658), min(lr, lr)))), avg(rf, deci[0.25](svm, rf, lr))), mul(sub(1.000, lf(min(add(mul(rf, -0.632), avg(mlp, lr)), add(add(svm, 0.658), min(lr, lr))))), add(mul(lf(sub(add(mlp, svm), min(mlp, deci[0.25](0.854, lr, svm)))), add(mul(lf(add(mul(mlp, deci[0.25](mlp, avg(mlp, -0.294), mul(0.253, -0.650))), min(min(deci[0.25](0.953, lr, svm), mul(-0.653, mlp)), add(add(mlp, lr), deci[0.25](mlp, rf, svm))))), add(add(mul(lf(mul(add(mlp, min(deci[0.25](rf, -0.295, rf), sub(mlp, svm))), min(sub(deci[0.25](0.832, lr, lr), deci[0.25](svm, rf, svm)), deci[0.25](deci[0.25](-0.153, 0.125, mlp), sub(svm, -0.967), -0.237)))), sub(sub(min(rf, mlp), min(0.954, mlp)), min(avg(-0.067, svm), sub(-0.850, 0.004)))), mul(sub(1.000, lf(mul(add(mlp, min(deci[0.25](rf, -0.295, rf), sub(mlp, svm))), min(sub(deci[0.25](0.832, lr, lr), deci[0.25](svm, rf, svm)), ...