NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Improving Tree-based Pipeline Optimization Tool with Geometric Semantic Genetic Programming

Helena Hetal Chhotobhai

Dissertation

presented as partial requirement for obtaining the Master Degree Program in Data Science and Advanced Analytics

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

[this page should not be included in the digital version. Its purpose is only for the printed version]

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

IMPROVING TREE-BASED PIPELINE OPTIMIZATION TOOL WITH GEOMETRIC SEMANTIC GENETIC PROGRAMMING

by

Helena Hetal Chhotobhai

Dissertation report presented as partial requirement for obtaining the Master's degree in Advanced Analytics, with a Specialization in Data Science

Supervisor: Leonardo Vanneschi

July 2022

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisbon, 25 July 2022

Improving Tree-based Pipeline Optimization Tool with Geometric Semantic Genetic Programming

ABSTRACT

Machine Learning (ML) is becoming part of our lives, from face recognition to sensors of the latest cars. However, the construction of its pipelines is a time-consuming and expensive process, even for experts that have the knowledge in ML algorithms, due to the several options for each step. To overcome this issue, Automated ML (AutoML) was introduced, automating some steps of this process. One of its recent algorithms is Tree-Based Pipeline Optimization Tool (TPOT), an Evolutionary Algorithm (EA) that automatically designs and optimizes ML pipelines using Genetic Programming (GP). Another recent algorithm is Geometric Semantic Genetic Programming (GSGP), an EA characterized by using the semantics, the vector of outputs of a program on the different training data, and by searching directly in the space of semantics of the program through geometric semantic operators, leading to a unimodal fitness landscape. In this work, a new version of TPOT was created, called TPOT-GSGP, where GSGP is one of the options for model selection. This new algorithm was implemented in Python, only for regression problems and using Negative Mean Absolute Error as measurement error. Five case studies were used to compare the performance of three algorithms: TPOT-GSGP, the original TPOT, and GSGP. Additionally, the statistical significance of the difference on the last generation's score for each combination of two algorithms was checked with Wilcoxon tests. There was not a single algorithm that outperformed the others in all datasets, sometimes it was TPOT-GSGP and others TPOT, depending on the case study and on the score that was analysed (learning or test). It was concluded that every time GSGP is chosen as root 50% of the times or more, TPOT-GSGP outperformed TPOT on the test set. Therefore, the advantages of this new algorithm can be extraordinary with its development and adjustment in future work.

KEYWORDS

Automated Machine Learning; Genetic Programming; Geometric Semantic Genetic Programming; Tree-based Pipeline Optimization Tool; Regression

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

2SEGP	Simple Simultaneous Ensemble Genetic Programming
AI	Artificial Intelligence
ANN	Artificial Neural Network
AutoML	Automated Machine Learning
BIO	Bioavailability
BP	Back-propagation
CNN	Convolutional Neural Networks
CONC	Concrete
CSEL-Algs	Complex Simultaneous Ensemble Learning Algorithms
CS	Cuckoo Search
CV	Cross Validation
DCFL	Divide-and-conquer feature learning
DEAP	Distributed Evolutionary Algorithms in Python
DNN	Deep Neural Network
DT	Decision Tree
eGP	Ensemble GP
EA	Evolutionary Algorithm
EC	Evolutionary Computations
EP	Evolutionary Programming
ES	Evolution Strategies
FF	Forest Fires
FWI	Fire Weather Index
GA	Genetic Algorithms
GP	Genetic Programming
GSGP	Geometric Semantic Genetic Programming
GWO	Grey Wolf Optimization
HPC	High-performance concrete
FWI	Fire Weather Index
KT	Knowledge transfer
LD50	Median Lethal Dose
LM	Levenberg Marquardt
LSTM	Long Short-Term Memory
M3GP	Multidimensional Multiclass GP with Multidimensional Populations

MFO	Moth-flame optimization
ML	Machine Learning
MLP	Multilayer perceptron
MLR	Multi-Linear Regression
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MLPR	Multi-Layer Perceptron Regression
MVO	Multi-Verse Optimizer
NMAE	Negative Mean Absolute Error
NN	Neural Networks
NSGA-II	Nondominated Sorting Genetic Algorithm II
TPOT	Tree-based Pipeline Optimization Tool
PCA	Principal Component Analysis
PPB	Plasma Protein Binding Level
PSO	Particle Swarm Optimization
RFR	Random Forests Regression
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
S-GP	Stacking GP
SASEGASA	Self Adaptive SEgregative Genetic Algorithm with Simulated Annealing aspects
SAGP	Self-Adaptive Genetic Programming
SEGA	Segregative Genetic Algorithm
SIEL-Apps	Simple Independent Ensemble Learning Approaches
SSA	Salp Swarm Algorithm
SVM	Support Vector Machines
SVR	Support-Vector Regression
TPOT	Tree-based Pipeline Optimization Tool
TOX	Toxicity
XGBoost	eXtreme Gradient Boosting

1. INTRODUCTION

Our lives are gaining a new component that is becoming more important and more popular, ML. The study of algorithms that automatically improve by means of experience, that can also be considered a try and error process, lead to the construction of computer programs capable of learning autonomously, improving their performance on each task or problem. In order to have a well-defined learning problem, it is crucial to define well the tasks, the performance metric (or measurement of error), and the source of training experience.

With a broad range of applications in several areas, such as the analysis of the outcomes of medical treatments or face recognition images, this subarea of AI can be divided into supervised and unsupervised problems according to the existence or not of a target. In supervised learning, the representation of the data can be in a matrix $n * p$ that has n instances and p features, and a n-dimensional vector to represent the target variable. On the other hand, in unsupervised learning, the target does not exist. Additionally, the partition of data can be done also in different ways, being some adequate to small datasets and another for larger datasets.

There are unlimited combinations of options for each one of the usual steps of supervised learning, data preparation, data transformation, model selection, parameter optimization and model validation. It is important to check if the model is not overfitting, using the parameters chosen. So, designing a ML system and implementing it is a time-consuming and error-prone process, usually requiring experience, expert knowledge, and use of brute force techniques, for saving time and avoiding testing all the possibilities if knowledge about which choice is adequate for a specific task exist. To overcome this issue, AutoML was created to automatize some parts of ML process as the name suggests. It automates the steps that are iterative and repetitive, such as the comparison of the models, tunning the parameters, among others. One example is the TPOT, an AutoML tool based on EA more concretely on GP. TPOT automatically designs and optimizes some steps of ML pipelines using GP, being the steps of feature selection, feature preprocessing, feature construction, model selection and parameter optimization automatized.

EAs are based on the Theory of Evolution of Charles Darwin, being characterized by having a specific recombination (or crossover), mutation and selection operators, that can be applied in every cycle or only with a probability, considering that each consecutive cycle is a generation. In a nutshell, an EA initialises a population with random candidate solutions and evaluates them. Then, until the termination conditions defined a priori are satisfied, it selects parents, recombines (or performs the crossover of) pairs of parents, mutates the resulting offspring, evaluates the new candidates, and selects the individuals for the next generation. When the termination conditions are satisfied, it returns the best solution found. Being one type of EA, GP is “a domain-independent problem-solving approach in which computer programs are evolved to solve, or approximately solve, problems”(Koza, 1997), where there is a population composed by computer programs with different sizes and shapes. In tree-based GP, these computer programs are based on the set of terminals and on the set of primitive functions. Along with these two sets, the fitness measure, the parameters for controlling the run and the method for designating a result and the criterion for terminating the run are the definitions that need to be made to apply GP to a problem.

The semantics of a GP individual or program are the resulting outputs after applying a function for each input vector, in other words, the vectors of outputs of a program on different training data. In traditional GP, the search operators are applied to the syntactic representation of the programs, without considering the actual semantics, the indicator of success of the search at solving the problem. To overcome this issue, (Moraglio et al., 2012) introduced Geometric Semantic Genetic Programming (GSGP) with the objective of searching directly in the space of semantics of the program, using geometric semantic operators, resulting in a unimodal fitness landscape. However, this approach causes an exponential growth in the size of the individuals. So, (Vanneschi et al., 2013) proposed a new implementation of geometric semantic operators that consist of saving the operations needed to create new individuals without actually building them by using tables that store the initial population, the random trees used for crossover and mutation and the new population, making use of semantics for knowing the performance of the algorithms. In this sense, only the best individual (solution) is constructed at the end.

TPOT and GSGP are recent algorithms that are being studied, although the results obtained until here are satisfactory. In this sense, integration of ML with GP, ensemble learning with GP or evolving NN with GP can be considered related work. In the first topic, there is an application of integration of ML with GP for the identification of "nonlinear model structures for mechatronic system" (Winkler et al., 2007), another for autonomous image pre-processing and feature extraction when dealing with low-quality image classification (Bi et al., 2021a) and another for overcoming the problem of black-box based ML algorithms (Oh et al., 2021). Regarding the second topic, with the objective of automatically find the right setting for the ensemble learning, Ensemble GP was created by (Rodrigues et al., 2020), (Bakurov et al., 2021) studied GP as a meta-learner in the area of stacked generalization and (Virgolin, 2021) proposed a new algorithm in the area of bagging. In practical terms, (Roknizadeh & Naeen, 2021) proposed a method to solve the challenges of imbalanced data, (Bi et al., 2021b) for the limitation of high computational cost when using large-scale images and (Sahu et al., 2022) to predict software faults. In the third topic, there is a study that reviews the existing methods for evolving ANN (Floreano et al., 2008), (Jalali et al., 2020) performs a comparative study using four EAs, (Galvan & Mooney, 2021) did a comprehensive study about using neuroevolutionary approaches in DNN, and (Al-Badarneh et al., 2020) introduced three algorithms for imbalanced classification problems.

The objective of this work is to present a new version of TPOT that incorporates GSGP as one of the options for model selection, called TPOT-GSGP, for regression problems. In order to achieve this objective, an implementation of TPOT in Python that used the DEAP package and also an implementation of GSGP in Python will be used as the basis for the implementation of TPOT-GSGP. The necessary changes that will be performed include for example have the same measurement of error in TPOT and GSGP, the existence of elitism in TPOT and creation of at least one individual with only GSGP on the step of initialization of TPOT. After making the necessary changes to incorporate GSGP into TPOT, the performance of this new version will be compared with the original TPOT and with the GSGP alone, using the Negative Mean Absolute Error (NMAE). The case studies that will be used are Forest Fires, Concrete, Bioavailability, Plasma Protein Binding Level and Toxicity, being the last three pharmacokinetic parameters analysed for drug discovery and development. These datasets will be divided into learning (70%) and test (30%), and a 5-fold cross validation will be used (where the learning set will be divided into five parts, being four of them considered as the training set and the other as the validation set). First, two grid searches with 10 runs each will be performed to know the best value for GSGP's mutation step and tournament size. Then, using the best values for these

parameters, each of the three algorithms, TPOT-GSGP, TPOT and GSGP, will be run 30 times with different random seeds. The analysis of the learning and test scores for each problem will be analysed to know if one algorithm outperforms the others. Additionally, a Wilcoxon test will be performed to check if the differences on the last generation's score (learning and test scores) between each combination of two algorithms for each case study are statistically significant or not. It is expected that TPOT-GSGP will be an improved version of the original TPOT and GSGP since TPOT and GSGP are powerful algorithms by themselves. However, this may not be true for every dataset since the performance of the algorithms depend on the characteristics of each dataset.

This work is organized as follows: Section 2 describes the theoretical background necessary for understanding this work, presenting concepts from ML to GSGP or even TPOT; Section 3 reviews previous and related work, that can be divided into three areas: integration of ML with GP, ensemble learning with GP and evolving NN with GP; Section 4 presents the methodology and hence, the new algorithm TPOT-GSGP; Section 5 describes the experimental study, including the experimental setting, the case studies, the experimental results and discussion; Section 6 presents the conclusions; and Section 7 presents the limitations and recommendations for future work.

2. THEORETICAL BACKGROUND

2.1. MACHINE LEARNING

Machine Learning (ML) is a field of Artificial Intelligence and its the most accepted definition is:

> "Machine Learning is the study of algorithms that automatically improve by means of experience." (Mitchell, 1997)

It makes use of several areas, like mathematics, statistics, biology, computational complexity, among others, to build computer programs capable of learning autonomously through a try and error process, knowing what is doing right or wrong, and therefore improving their performance on each task or problem. This computer program capable of learning can be defined more precisely as:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997)

where the identification of the class of tasks T, the performance metric to be improved P, and the source of training experience E is crucial to have a well-defined learning problem. So, designing a ML approach involves diverse options for choosing the type of training experience, the target function to be learned and its representation, and an algorithm for learning this target function from training examples.

There is a broad range for its applications, such as data mining problems with large amounts of data (for example, learn general rules for credit worthiness from financial databases or analyse outcomes of medical treatments from patient databases); domains where human capacity is not enough to develop algorithms (for example, face recognition from images); and, domains that require changes through time according to certain conditions (for example, adapt the reading interests of individuals) (Mitchell, 1997). As the years pass, more applications for algorithms that can learn autonomously and create insights from large amounts of data have been discovered.

There are two types of learning in this area, unsupervised and supervised. Unsupervised learning has the objective of discovering hidden patterns in data by analysing and clustering unlabelled data sets, giving insights from large volumes of data. On the other hand, in supervised learning the datasets are labelled, and the goal is to train a model so it can classify the data or predict the target accurately. This type of learning can be divided into two types of problems: if the target is discrete (having specific categories), then it is a classification problem; if the objective is to predict a continuous target, then it is a regression problem.

The data can be represented in a tabular format, with a matrix $n * p$ that has n instances and p features, and a n-dimensional vector to represent the target variable, as can be seen in Figure 2.1. It is important to recall that the target only exists for supervised problems.

x_1	x_2	x_3	...	x_p
x_{11}	x_{12}	x_{13}	...	x_{1p}
x_{21}	x_{22}	x_{23}	...	x_{2p}
...	...	...	...	...
x_{n1}	x_{n2}	x_{n3}	...	x_{np}

t	t_1	t_2	t_3	...	t_n

Figure 2.1 - Representation of data in ML.

The usual steps of ML supervised learning are:

- Data preparation - initial exploratory analysis (to check if there is missing or mislabelled data) and data cleaning (check for inconsistencies and errors in data) (Brownlee, 2020);
- Data transformation - feature preprocessing (normalizing the features), feature selection (removing features that are not helpful) (Cai et al., 2018) and/or feature construction (creating new features from the existing ones) (Dong & Liu, 2018) (Kuhn & Johnson, 2019);
- Model selection – choose the ML algorithm to fit to the data;
- Parameter optimization – tune the model's parameters to get the most accurate classification of the data;
- Model validation – check if the model has generalization ability, that is, if it can predict data that was not used to fit the model accurately.

(Olson et al., 2016)

Besides these steps, one important choice is required, which will be the measurement of error. The error is generally calculated in each generation to know how the model is behaving. There are classification metrics for classification problems, such as accuracy, precision, recall, F1-score, among others, (Hossin & Sulaiman, 2015), and regression metrics for regression problems, such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), among others, (Hoffmann et al., 2019).

Focusing on regression problems, the error is characterized by the difference between the real value of the target and the predicted value by the model. The Mean Absolute Error (MAE) gives an idea of the magnitude of the errors, and it is computed as:

$$MAE = \frac{\sum_{i=1}^{N} | Y_i - Y_i^p |}{N}$$

where N is the total number of instances, Y_i is the true value and Y_i^p is the predicted value. Since this is a measure of error, the closer to zero the better. Being the possible values of MAE positive ones, the objective is to minimize it. In the extreme case in which it is zero, it means that the model predicts the target perfectly.

There is another measurement of error that is an extension of the MAE, the Negative Mean Absolute Error (NMAE). The difference is that it changes the value of the error from positive to negative and it is computed as:

$$NMAE = - \frac{\sum_{i=1}^{N} | Y_i - Y_i^p |}{N}$$

where N is the total number of instances, Y_i is the true value and Y_i^p is the predicted value, as in MAE. When using NMAE as the measurement of error, instead of a minimization problem as it is with MAE, the objective is to maximize it.

Different error measures also exist, for example the Root Mean Squared Error (RMSE) that is the square root of the mean of the squared differences between the real value and the predicted value.

Another crucial step in supervised learning is the partition of data. As mentioned before, the data in this type of learning is composed by features or independent variables and one target or dependent variable, different nomenclatures exist. The data can be split into a training, a validation and a test set, which is the common partition for datasets that are not too small. Otherwise, if the dataset has a small number of instances, it is usual to split the data into a training and test set only, not having a validation set.

The training set, as the name indicates, is used to train and fit the model. The validation set is used to compare different models when more than one is being considered, to check how the model is behaving with different data and to see which parameters can be changed to improve the model. Lastly but also important, the test set is used to check the performance and generalization ability of the model, how does the model behave in unseen data, if it can predict the target precisely.

The situation where the model can predict very well the train and validation data, but the predictions for the test data are far from the real values is called overfitting. That is, having a low train and validation errors and a huge test error means that the model is too shaped to the data used for training and it cannot adjust to the unseen data. In other words, “model does not generalize well from observed data to unseen data” which can be caused by different reasons (Ying, 2019).

One way to avoid overfitting is to use a k-fold cross validation (CV) procedure (Rodríguez et al., 2010), an example of a 5-fold cross validation approach represented in Figure 2.2. The data is divided into k subsets with the same size. For each subset, the model is training using the other k-1 subsets and evaluated on that subset (considered as validation set). Therefore, this process is repeated k times, having k validation scores. Finally, the validation score reported is the average of all k validation scores. Although it has the disadvantage of being computationally expensive, the advantage usually overcomes it. When comparing to the approach of split the data into train, validation and test sets, the k-fold CV does not restrict data because it does not fix an arbitrary validation set and therefore does not waste too much data.

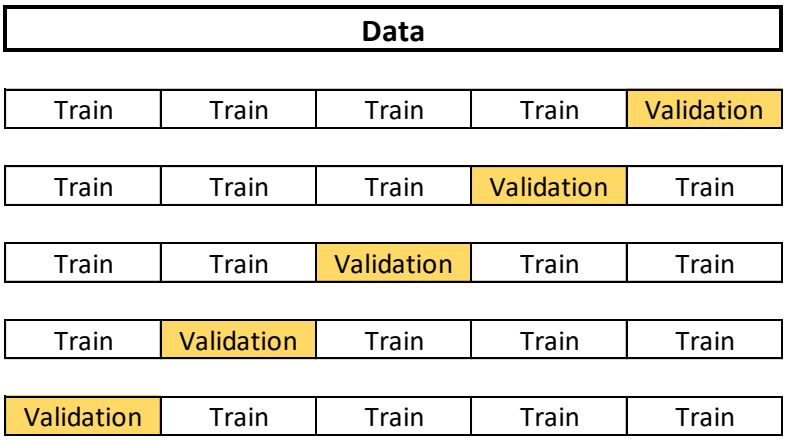


Figure 2.2 - Example of a 5-fold cross validation approach.

2.1.1. Automated Machine Learning (AutoML)

Given a specific task, designing the ML system and implementing it is a time-consuming and error-prone process, even for experts that have substantial skills (Vaccaro et al., 2021). Additionally, the effective application of ML tools commonly requires experience with ML algorithms, expert knowledge of tool and of the problem domain, and use of the existing brute force techniques.

As mentioned before, the main steps of ML lead to several possibilities. For instance, which treatment should be applied to preprocess data (remove or correct the instances that have inconsistencies, use feature selection or use PCA, normalizing or not the data), which algorithm should be used to fit the data (Random Forest, Decision Tree, K-Neighbors), which values are ideal for model parameters (how many trees in random forest, how many neighbours in K-Neighbors), among others (Olson et al., 2016). In this sense, achieve a good ML pipeline is a time-consuming task because it is necessary to test different combinations and check which one is the best, knowing that there are infinite possibilities.

Automated Machine Learning (AutoML) overcomes this issue by automating some parts of the process in ML that are iterative and repetitive, for instance the comparison of different models, the tunning of the parameters, among others. Different tools of AutoML exist, like PyCaret (https://pycaret.org/sampling /) or Auto-SKLearn (https://www.automl.org/automl/auto-sklearn/).

Recently it was developed an EA that automatically designs and optimizes ML pipelines, the Tree-based Pipeline Optimization Tool (TPOT). This is one example of an AutoML algorithm based on Genetic Programming. Therefore, in order to present this algorithm, it is necessary to first explain Genetic Programming.

2.2. EVOLUTIONARY ALGORITHMS

Evolutionary Algorithms (EAs), a subarea of Evolutionary Computations (EC), represent the algorithms that are inspired by the Theory of Evolution of Charles Darwin. Our planet evolved and improved from the initial configuration, where there only exist unicellular units/human beings, to nowadays, where there is a wide variety of human beings. The idea of EA is that the laws that guided to these improvements could be captured into a computer.

In the last years, EA have been used to solve complex real-world problems, like robotics, operation research, decision making, bioinformatics, machine learning, data mining, where using heuristic solutions (Abbass et al., 2002) is not possible and may lead to inadequate results. So, EA become a very popular tool to search, optimize and provide solutions to complex problems (Vikhar, 2017).

EA is composed by four variants: Evolutionary Programming (EP), Genetic Algorithms (GA), Evolution Strategies (ES) and Genetic Programming (GP) (Bartz-Beielstein et al., 2014), that have the same underlying idea "given a population of individuals within some environment that has limited resources, competition for those resources causes natural selection (survival of the fittest)" (Eiben & Smith, 2003). The individuals are considered as solution candidates. Each candidate has a fitness value, a measure to quantify the quality of the solution. In addition to the usual requirements of an algorithm (an initialization and a termination condition), three crucial components can be considered for EAs: a set of search operators, usually implemented as recombination and mutation; a specific control flow (Figure 2.3); and a representation mapping the adequate variables to implementable solution candidates, also called genotype-phenotype mapping (Bartz-Beielstein et al., 2014). One of the most used definitions for EA is:

> "EA: collective term for all variants of (probabilistic) optimization and approximation algorithms that are inspired by Darwinian evolution. Optimal states are approximated by successive improvements based on the variation-selection-paradigm. Thereby, the variation operators produce genetic diversity and the selection directs the evolutionary search."
>
> (Beyer et al., 2002)

Each variant of EAs gives different levels of importance to the search operators, recombination and mutation, but the overall effect is the same. The first one's effect is to reorganize existing information in the "inner space" while the second one explores the "outer space" that is not covered by the population since it indicates neighbourhood-based movement in the search space. Additionally, selection introduces a bias towards better fitness values in two possible moments, when the parents are chosen to generate offspring - mating selection; or when the new solutions created compete to be inserted in the population - environmental selection or survival selection.

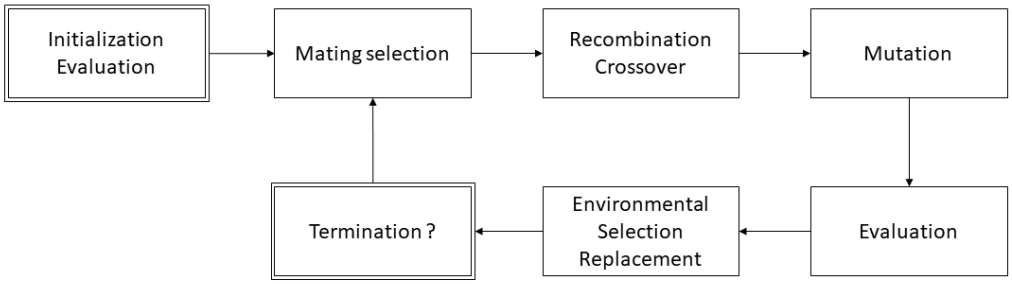


Figure 2.3 - Basic working scheme of all EAs. (Bartz-Beielstein et al., 2014)

A typical EA should have a specific recombination, mutation and selection operators, and these operators can be applied in every cycle or applied only with a specific probability, maintaining the control flow unaffected always. Each consecutive cycle is called generation.

It is important to say that several moments of the evolutionary process are stochastic (Eiben & Smith, 2003). In the selection phase, weak individuals can also be chosen and the best individuals are not chosen deterministically. In the recombination phase, the parent's part that will be recombined is chosen randomly, just like in mutation, where the part of candidate solution that will be changed and the new parts that will replace them are also chosen randomly. Being EAs stochastic, there is no guarantee that the optimal solution will be reached (Vikhar, 2017). Additionally, the solution founded by the algorithm in one run may be slightly different to another run (Bartz-Beielstein et al., 2014).

The general scheme of an EA using pseudocode is:

```
BEGIN
    INITIALISE population with random candidate solutions;
    EVALUATE each candidate;
    REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO
        1  SELECT parents;
        2  RECOMBINE pairs of parents;
        3  MUTATE the resulting offspring;
        4  EVALUATE new candidates;
        5  SELECT individuals for the next generation;
    OD
END
```

(Eiben & Smith, 2003)

2.2.1. Genetic Programming

As mentioned before, Genetic Programming (GP) is based in Darwinian principle of reproduction and survival of the fittest, using genetic operators like crossover (sexual recombination) and mutation. It can be defined as:

> "Genetic programming is a domain-independent problem-solving approach in which computer programs are evolved to solve, or approximately solve, problems."
>
> (Koza, 1997)

So, in GP, the population is composed by hierarchical computer programs with different sizes and shapes, contrary to Genetic Algorithm (GA) which is characterized by fixed-length size string representation of the individuals. However, GP can be considered an extension of GA because it is based on the efficiency of learning of GA (Holland, 1992).

In this sense, GP executes the following steps to breed programs:

1. Generate an initial population of computer programs, i.e. individuals.
2. Iteratively perform the following steps until the termination criteria is satisfied:
   2.1. Execute each program in the population and assign it a fitness value according to its capability of solving the problem.
   2.2. Create a new population by applying the following operators:
      2.2.1. Select a set of computer programs probabilistically, according to their fitness – selection.
      2.2.2. Copy some of the selected computer programs without modifications into the new population – reproduction.
      2.2.3. Recombine genetically randomly chosen parts of two selected individuals to create new computer programs – crossover.
      2.2.4. Substitute randomly chosen parts of some selected individuals with new randomly generated individuals to create new computer programs – mutation.
3. Return the best computer program existing in any generation, that is designated as the result for the run. This result may be a solution (or an approximate solution) to the problem.

Contrary to GA, where crossover and mutation are applied to the same individuals, in GP crossover and mutation are not applied sequentially. So, the individuals that were created from crossover are not submitted to mutation and vice-versa. Therefore, based on prefixed probability's parameter, a set of individuals undergo reproduction, another set of individuals undergo crossover and another set of individuals undergo mutation. The individuals resulting from these operators are usually inserted in the new population.

In order to apply GP to a problem, five definitions need to be made:

- the set of terminals,
- the set of primitive functions,
- the fitness measure,
- the parameters for controlling the run, and
- the method for designating a result and the criterion for terminating a run.

(Koza, 1992)

In tree-based GP, the representation of a GP individual is based on the set of terminals and on the set of primitive functions. Therefore, a computer program (i.e. individual) is a composition of functions from the function set $\mathcal{F}$ and terminals from terminal set $\mathcal{T}$. Functions can include arithmetic operations ($+, -, *, /$, among others), other mathematical functions (like sin, cos, log, exp), boolean operators (like AND, OR, NOT), conditional operators (like $IF - THEN - ELSE$), iterative operations (like $WHILE - DO$), and/or any other domain-specific functions that may be defined. Terminals are usually

variables or constants, defined on the problem domain. An example of a GP individual is represented in Figure 2.4, considering the function set $\mathcal{F}$ and terminal set $\mathcal{T}$ presented below.

$$\mathcal{F} = \{+,0\}, \mathcal{T} = \{x, 1\}$$

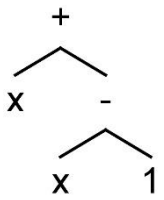


Figure 2.4 - One possible tree that can be built with the sets $\mathcal{F} = \{+,0\}, \mathcal{T} = \{x, 1\}$.

When choosing $\mathcal{F}$ and $\mathcal{T}$, the closure property needs to be satisfied. It requires that each function in $\mathcal{F}$ should allow any value and data type that can be returned by any function in $\mathcal{F}$ and that can be assumed by any terminal in $\mathcal{T}$. In this sense, there is the guarantee that every combination of functions and terminals results in an executable computer program. For example, return zero when the division by zero is attempted if division is one of the functions. The property of sufficiency also needs to be satisfied, that is, the set of functions and the set of terminals should be capable of expressing a solution to a problem. For example, if the problem is a logic problem, the functions cannot be only arithmetic operations.

Every computation program is evaluated after the execution to check how it is performing in the specific problem through the fitness measure. Generally, this fitness measure is the difference between the computer program's result and the true value, so the closer to zero, the better.

The parameters for controlling the run are the population size and the maximum number of generations.

The method for designating a result usually is the best individual obtained in any generation of the population during the run, also known as the best-so-far individual. (Koza, 1992)

GP is characterized by breeding a vast amount of population for a large number of generations which is caused by the Darwinian principle of survival and reproduction of the fittest in addition to a genetic crossover operation capable of mating computer programs. So, this combination of Darwinian natural selection and genetic operations lead to a computer program able to solve (or approximately solve) a specific problem.

The initial population, in generation 0, is randomly generated from the determined set of functions and terminals, usually with a limit for the size equal to a maximum number of points (total number of functions and terminals) or a maximum depth of the program tree. These initial individuals usually have poor fitness, but some of them have better fitness than the others. After, these discrepancies will be exploited by the Darwinian principle.

There is the possibility of using elitism, where the best individual from the current generation is automatically copied to the next generation, to guarantee that the best individual is not lost and that some improvement will be tried. (Poli et al., 2008)

A new offspring population will be created from the reproduction and survival of the fittest along with the genetic operations.

The reproduction operation selects an individual from the current population with probability based on fitness and the selected individual is copied into the new population.

The crossover operation creates new offspring individuals from two parents chosen with probability based on their fitness. The parents usually have different sizes and shapes. By choosing some parts of each parent through the crossover point, subexpressions (like subtrees, subprograms) compose these new offspring computer programs, that are also of different types and shapes. Since different parts of each parent are selected, this genetic operation always produces offspring that are syntactically and semantically valid.

The mutation operation creates new offspring individuals from one parent chosen with probability based on its fitness. Using the same growth process as in the initial population, a new individual is generated randomly to replace some part of the selected parent, also selected randomly.

The new offspring population (new generation) created from these genetic operations replaces the old population (old generation).

GP has different important features, such as the hierarchical character and dynamic variability of computer programs, and the needless to preprocess data or postprocess outputs (Koza, 1992).

2.2.1.1. Geometric Semantic Genetic Programming

Considering that $\mathrm{X} = \{\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n}\}$ is the set of input data (or fitness cases) of a symbolic regression problem, and $\vec{t} = [t_1, t_2, \ldots, t_n\ ]$ the vector of target values, that is for each $i = 1,2,\ldots,n$, t_i is the expected output corresponding to input $\overrightarrow{x_i}$, the semantics of P (a GP individual or program) is given by the vector

$$\overrightarrow{s_p} = [P(\overrightarrow{x_1}), P(\overrightarrow{x_2}), \ldots, P(\overrightarrow{x_n})]$$

where P is a function that, for each input vector $\overrightarrow{x_i}$ returns the scalar value $P(\overrightarrow{x_i})$. So, semantics can be defined as the vector of outputs of a program on the different training data.

Therefore, the space of all these semantics can be identified as semantic space, which is composed by vectors of prefixed length of typically real numbers $(\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n})$ where the optimum is known (since it is a supervised problem).

In the traditional GP, the search operators are applied to the syntactic representation of the programs, regardless of their actual semantics. However, it is the semantics of the program that indicates how successful is the search at solving the specific task.

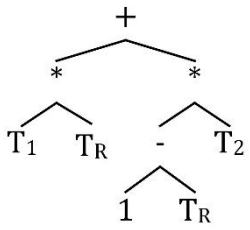
Geometric Semantic Genetic Programming (GSGP) was introduced in order to "search directly the space of the underlying semantics of the program" (Moraglio et al., 2012), using GP with geometric semantic operators.

The geometric semantic crossover is defined as:

> "Given two parent functions $\mathcal{T}_1, \mathcal{T}_2 : \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic crossover returns the real function $\mathcal{T}_{XO} = (\mathcal{T}_1 \cdot \mathcal{T}_R) + ((1 - \mathcal{T}_R) \cdot \mathcal{T}_2)$, where $\mathcal{T}_R$ is a random real function whose output values range in the interval $[0,1]$."

(Moraglio et al., 2012)

$\mathcal{T}_{XO} =$



$\mathcal{T}_R = Random\ function\ with\ codomain\ [0,1]$

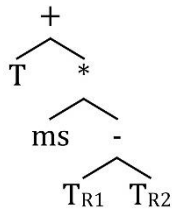
So, the semantics of the offspring generated by this type of crossover is a linear combination of the parents' semantics with random coefficients included in $[0,1]$ and whose sum is equal to 1 (Vanneschi, 2017).

The geometric mutation is defined as:

> "Given a parent function $\mathcal{T}: \mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic mutation with mutation step ms returns the real function $\mathcal{T}_M = \mathcal{T} + ms \cdot (\mathcal{T}_{R1} - \mathcal{T}_{R2})$, where $\mathcal{T}_{R1}$ and $\mathcal{T}_{R2}$ are random real functions."

(Moraglio et al., 2012)

$\mathcal{T}_M =$



$\mathcal{T}_R = Random\ function\ with\ codomain\ [0,1]$

This type of mutation generates offspring with a semantic vector where each element is a "weak" (since $(\mathcal{T}_{R1} - \mathcal{T}_{R2})$ is centered in 0) perturbation of the corresponding element in the semantics of the parent, having the possibility of tune its performance through ms parameter. So, this operator corresponds to a box mutation on the semantic space, inducing a unimodal fitness landscape (Vanneschi, 2017).

Although these operators induce a unimodal fitness landscape on every problem consisting in matching input data into known targets, the disadvantage is that it causes an exponential growth in

size of the individuals, proven in (Moraglio et al., 2012). To overcome this situation, a new implementation of these geometric semantic operators was proposed by (Vanneschi et al., 2013).

Firstly, as in standard GP, the initial population of individuals is created typically randomly. Then, these individuals are stored in a table, that will be called P (Figure 2.5a)), and evaluated. The evaluations will be stored in a new table, table V, that include the semantics for each individual in P. So, table V will have n rows and k columns for a population of n individuals and a training set of k instances (fitness cases).

Then, a new empty table V' is created at each generation. Every time crossover between selected parents $\mathcal{T}_1$ and $\mathcal{T}_2$ generates a new individual $\mathcal{T}$, $\mathcal{T}$ is represented by a triplet $\mathcal{T} = \langle ID(\mathcal{T}_1), ID(\mathcal{T}_2), ID(R)\rangle$, where for any tree $\mathcal{T}$, $ID(\mathcal{T})$ is a reference (or memory point) to $\mathcal{T}$, and R is a random tree. This triplet $\mathcal{T}$ is stored in P in an appropriate structure, that will be called $\mathcal{M}$ from now, including the name of the operators used (Figure 2.5c)). The random tree R is created, also stored in P and evaluated in order to get its semantics for each fitness case. The evaluations of $\mathcal{T}$ to obtain its semantics can be calculated by applying the formula $(\mathcal{T}_1 \cdot R) + ((1 - R) \cdot \mathcal{T}_2)$ to each fitness case, as the definition of geometric semantic crossover states, and then stored in V'. As in crossover, every time mutation applied to an individual $\mathcal{T}_1$ generates a new individual $\mathcal{T}$, $\mathcal{T}$ is represented by a triplet $\mathcal{T} = \langle ID(\mathcal{T}_1), ID(R_1), ID(R_2)\rangle$, stored in $\mathcal{M}$, where R_1 and R_2 are two random trees (newly created, then stored in P and evaluated to obtain their semantics). The evaluations of $\mathcal{T}$ to obtain its semantics can be calculated by applying the formula $\mathcal{T}_1 + m_s \cdot (R_1 - R_2)$ to each fitness case, as the definition of geometric semantic mutation states, and then stored in V'.

In the end of each generation, a copy of table V' to table V is performed. Table V' is erased along with all the individuals in P and $\mathcal{M}$ that are not parents or random trees of new individuals of the new population created by applying the operators crossover and mutation. It is important to say that the size of V is independent from the number of generations but $\mathcal{M}$ grows at each generation, since it stores the individual's semantics separated.

This new implementation follows the idea that the individuals generated by semantic operators can be totally described by its semantics, being the syntactic representation not so important as in standard GP. In this sense, the table V is updated with the semantics of the new individuals, and it has the information needed to build the syntactic structure of these new individuals without actually building them. This way of storing the semantics and updating the table V is very efficient in terms of computational time because there is no need to evaluate the entire trees.

The last step performed at the end of the final generation is to reconstruct the individuals (usually only the best individual found), that is ,"unwind" the compact representation and obtain the syntactic representation of it. This final step can be a disadvantage, except when the solution can be presented as a "black box" and therefore there is no need to get the syntactic representation of the best individual.

Although there is still this disadvantage of large tree-size individual(s) to report in the end, during the evolutionary process less computational effort is required. This new implementation of geometric semantic operators outperforms the standard one.

An example will be explained using Figure 2.5 (Vanneschi et al., 2013). Table (a) represents the initial population P and table (b) that is composed by random trees added to P as needed. Just for a reason of simplicity, the individuals of the new population P' will be generated using only crossover. In addition to the infix notation for individual's representation, an identifier (Id) for each individual ($\mathcal{T}_1, \ldots, \mathcal{T}_5$, and $R_1, \ldots, R_5$) is used to represent different individuals, being the new individuals of the new population represented by the identifiers $\mathcal{T}_6, \ldots, \mathcal{T}_{10}$.

Id	Individual
$\mathcal{T}_1$	$x_1 + x_2 x_3$
$\mathcal{T}_2$	$x_3 + x_2 x_4$
$\mathcal{T}_3$	$x_3 + x_4 - 2x_1$
$\mathcal{T}_4$	$x_1 x_3$
$\mathcal{T}_5$	$x_1 - x_3$

(a)

Id	Individual
R_1	$x_1 + x_2 - 2x_4$
R_2	$x_2 - x_1$
R_3	$x_1 + x_4 - 3x_3$
R_4	$x_2 - x_3 - x_4$
R_5	$2x_1$

(b)

Id	Operator	Entry
$\mathcal{T}_6$	crossover	$\langle ID(\mathcal{T}_1), ID(\mathcal{T}_4), ID(R_1) \rangle$
$\mathcal{T}_7$	crossover	$\langle ID(\mathcal{T}_4), ID(\mathcal{T}_5), ID(R_2) \rangle$
$\mathcal{T}_8$	crossover	$\langle ID(\mathcal{T}_3), ID(\mathcal{T}_5), ID(R_3) \rangle$
$\mathcal{T}_9$	crossover	$\langle ID(\mathcal{T}_1), ID(\mathcal{T}_5), ID(R_4) \rangle$
$\mathcal{T}_{10}$	crossover	$\langle ID(\mathcal{T}_3), ID(\mathcal{T}_4), ID(R_5) \rangle$

(c)

Figure 2.5 - Example of implementation of GSGP, where table (a) represents the initial population P, table (b) the random trees that can be used for crossover and mutation, and table (c) the representation of memory of the new population P'. (Vanneschi et al., 2013)

The individuals of P' are represented by the set of entries as shown in table (c), where each individual has a reference to the parents or random tress and the operator (crossover or mutation) used to generate it. For instance, the individual $\mathcal{T}_{10}$ is generated by crossover of individuals $\mathcal{T}_3$ and $\mathcal{T}_4$ and using the random tree R_5.

The information needed to reconstruct the genotype of an individual in P', for instance $\mathcal{T}_{10}$, is all shown in Figure 2.5. From table (c), it can be seen that $\mathcal{T}_{10}$ is generated by crossover of $\mathcal{T}_3$ and $\mathcal{T}_4$ and using the random tree R_5 (as mentioned above). Using the definition of geometric semantic crossover, the structure will be $(\mathcal{T}_3 \cdot R_5) + ((1 - R_5) \cdot \mathcal{T}_4)$. Using the syntactic structures represented in the other tables, the syntactic structure of $\mathcal{T}_{10}$ can be reconstructed, which will be $((x_3 + x_4 - 2x_1) \cdot (2x_1)) + ((1 - (2x_1)) \cdot (x_1 x_3))$ and after simplified, $-x_1(4x_1 - 3x_3 - 2x_4 + 2x_1 x_3)$.

2.2.1.2. Tree-based Pipeline Optimization Tool

Tree-based Pipeline Optimization Tool (TPOT) is an example of AutoML, as mentioned before, that automatically design and optimize machine learning pipelines using GP. In this way, most steps in a ML pipeline are automated, such as feature selection, feature preprocessing, feature construction, model selection and parameter optimization, as can be seen in Figure 2.6.

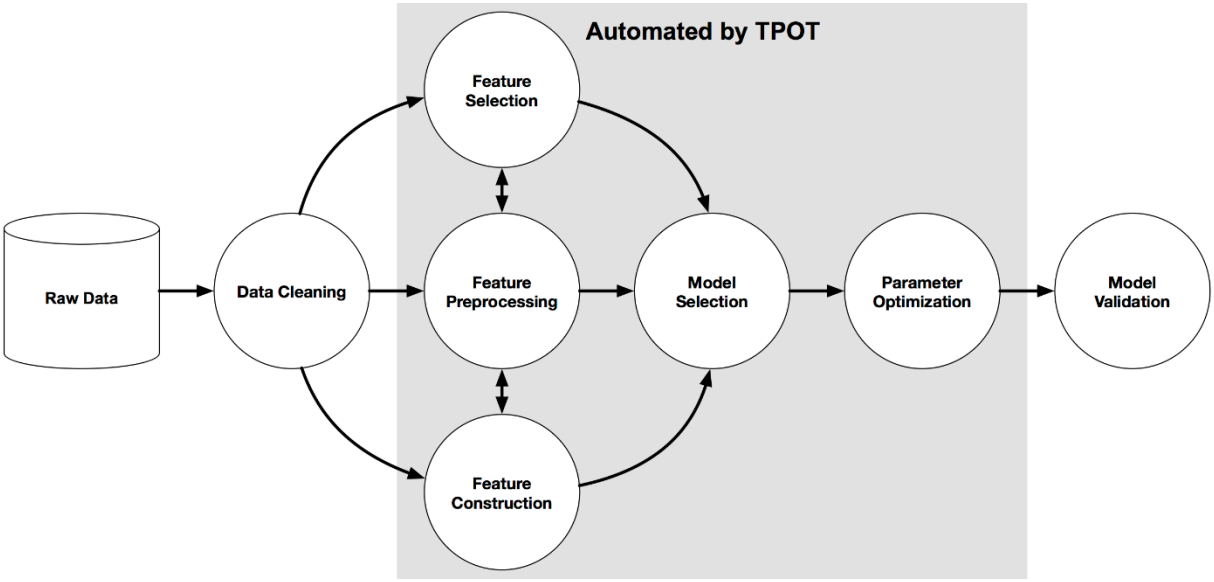


Figure 2.6 - Typical supervised ML pipeline, where the grey area demonstrates the steps that are automatized by TPOT. (Olson et al., 2016)

There is an implementation of this algorithm in Python using the DEAP package, generating an optimizing tree-based pipelines in an automated way (Olson et al., 2016). The tree-based pipelines have input data as leaves of the tree and operators as nodes of the tree. The operators can receive a dataset as input and return the modified dataset as output, making possible arbitrary shaped pipelines that can act on multiple copies of dataset, as it can be seen in Figure 2.7, an example of a tree-based ML pipeline. In this sense, flexible representation of pipelines is one advantage of GP trees.

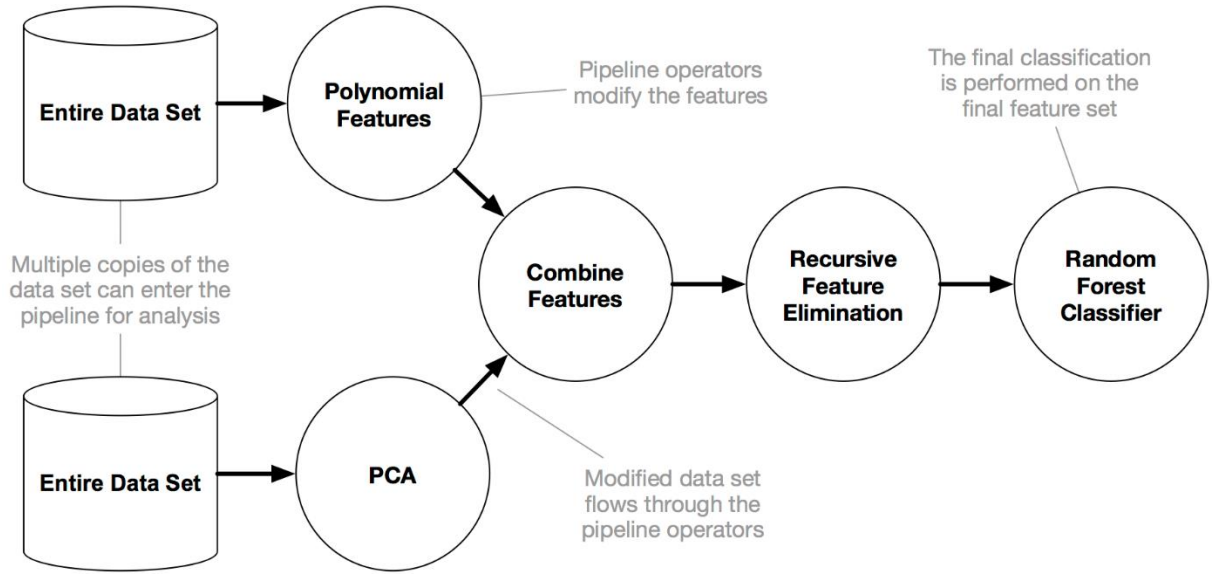


Figure 2.7 - Example of a tree-based ML pipeline. (Olson et al., 2016)

The pipeline starts with one or more copies of the input datasets as leaves of the tree, as mentioned before. Then, it is fed into one class of pipeline operators: Feature Preprocessing, Feature Selection and Supervised Learning Models, being these operators implemented in scikit-learn (Pedregosa et al., 2012). The data is modified by the operator of each node as it is passed up the tree. If there are multiple

copies of the dataset, there is the possibility of combining it into a single dataset through a dataset combination operator.

When the data passes through a Supervised Learning operator, the resulting classifications are stored and the previous algorithm's predictions are stored as a new feature, meaning that the new classifications overwrite the earlier ones (that will be stored as a new feature). After the data has passed all the processing steps (after passing the Random Forest Classifier in Figure 2.7), the final predictions are evaluated to determine the performance of the pipeline.

In this sense, the pipeline operators and its parameters are evolved using GP with the objective of minimizing the error. In the final step, the pipeline that optimizes the error is returned.

3. LITERATURE REVIEW

TPOT and GSGP are recent algorithms yet in research. Good results are being achieved, however these algorithms are still in study to deepen the knowledge of its performance and characteristics according to different types of problems.

Related work can be considered by three topics: the integration of ML with GP, the ensemble learning with GP or even evolving Neural Networks (NN) with GP.

Firstly, considering the integration of ML with GP, one example is a GP approach capable of "identifying nonlinear model structures for mechatronic systems" (Winkler et al., 2007), where the solution candidates are tree structure representations of symbolic expressions without fixed length and without distinguish the nonterminal and terminal nodes (since each node can be either of one type or another). This GP approach combined some algorithmic parts of SAGEGASA, a development of SEGA, with GP. Several operators and parameter settings were used and tested to see the differences. The results according to a specific dataset Thyroid are explained in this paper (Winkler et al., 2007), including some examples for analysing the classifiers and their quality through graphics.

Another use case of GP is for low-quality image classification, an important task in ML and computer vision. The learning process of discriminative features from low-quality images has a broad range of applications such as autonomous driving and safety surveillance, which has been increasing the attention on this subject. However, the low-quality of the images is a problem because of their variance in scale, illumination, rotation and distortions for example blur, noise and low contrast. Although image pre-processing can improve the quality of the images, it usually requires human intervention and knowledge domain. In this sense, (Bi et al., 2021a) proposed a new evolutionary learning approach, EFLGP, using GP with a new individual representation, a new function set and a new terminal set, capable of autonomously perform image pre-processing and feature extraction. More specifically, it is capable of learning discriminative features from images with blur, low contrast and noise for classification. After comparing with other methods and using eight datasets with different difficulties, the experimental study showed that this new approach had better or similar results and it was more invariant than the benchmark methods.

One last example of this topic, integration of ML with GP, is related to manufacturing big data analysis. The data is increasing exponentially and the right analysis of it contributes to the reduction of production cost, increase of productivity and profit, among many other advantages. So, it is important to be able to analyse it based on the manufacturing principles, which does not happen with black-box based ML algorithms. In this sense, (Oh et al., 2021) presented an algorithm for this area called Self-Adaptive Genetic Programming (SAGP) overcoming this issue of the black-box based ML algorithms. SAGP applies the self-adaptive technique into GP to obtain solutions that are highly accurate and highly interpretable in a symmetrical balanced way. It was concluded that this algorithm outperformed the other ML algorithms in addition to the possibility of intuitive analysis based on manufacturing principles that were not able with the other ML algorithms.

Regarding the second topic, ensemble learning (Sagi & Rokach, 2018) with GP, an explanation of what is ensemble is needed to facilitate the understanding of the examples that will follow. An ensemble is

a model that combines several base learners, individual models, outperforming these individual models by themselves and having a better capacity of generalization. According to the objective, there are different types of ensembles: bagging for decreasing variance, boosting for decreasing bias, and stacking for improving predictions. The first two, bagging and boosting, are characterized by considering mainly homogenous weak learners while stacking considers heterogenous weak learners.

The first example regarding this topic is Ensemble GP (eGP) (Rodrigues et al., 2020) that was created with the objective of automatically find the right settings for ensemble learning, given that the choice of parameters' values relies on each dataset's characteristics, and it is important to find the right values since the wrong choice of the values lead to bad results. Some of the elements of this eGP are M3GP inspired. In a nutshell, trees and forest compose the population. "The output model is a forest, built as an ensemble of trees. Each tree may be part of many different forests, and some trees may be part of none." (Rodrigues et al., 2020). Regarding fitness functions, the standard methods are used, difference between the expected and predicted output. Taking into account operators, trees and forests use different ones because the restrictions need to be satisfied. Different variants of eGP were tested against GP, M3GP, Random Forest and XGBoost using eight datasets. Although M3GP and XGBoost outperformed the others, in terms of generalization ability, eGP achieved a good performance in comparison with all the others to generalize.

The second example is the study of GP as a meta-learner in the context of stacked generalization called S-GP (Bakurov et al., 2021). The representation of the individuals of S-GP differs from traditional GP in the set of terminals T because in S-GP it is "composed of the predicted outputs ($\hat{y}$) of the base learners trained to solve a given Supervised Machine Learning problem" (Bakurov et al., 2021). The base learners used in this study were Multi-Linear Regression (MLR), Multi-Layer Perceptron Regression (MLPR), Random Forests Regression (RFR) and Support-Vector Regression (SVR). The function set $\mathcal{F}$ is composed by the usual arithmetic operators, admitting two operands. Additionally, it was also included the decision expression introduced by (Flasch et al., 2015), resembling the Decision Trees binary split, however with random feature and threshold selection. In terms of base learners' manipulation, there is the pre-requisite of training the base learners in order to extract their semantics to build $\mathcal{T}$. Regarding the geometric properties of the semantic space, the convex hull of the initial GP population's semantics is at least closer to the global optima, "containing" the target's semantics in the best case. For these reasons, this work had the objective of assessing Geometric Semantic Operators, different crossover-mutation probabilities, Evolutionary Demes Despeciation Algorithm initialization and $\varepsilon - Lexicase$ selection (designed specifically for regression problems). The experimental study was conducted using eleven problems and some of the conclusions were that the S-GP system achieved equivalent or better performance than the base learners in isolation, being some of these base learners already ensembles as it happens with Random Forest, and that this system is as good as the state-of-art ensemble approaches that include boost methods and other stacked generalization approaches.

The third example is related to bagging. The ensemble learning GP-based approaches that have been produced can be divided into two classes: one class that forms an ensemble of estimators producing one estimator at each time, called Simple Independent Ensemble Learning Approaches (SIEL-Apps); and another class that obtain an ensemble in one go using several novel mechanisms and respective hyper-parameters, called Complex Simultaneous Ensemble Learning Algorithms (CSEL-Algs). Since the SIEL-Apps is simple but inefficient and CSEL-Algs are very efficient but complex and difficult to adopt

in practical cases, a new algorithm with the good of each one was proposed by (Virgolin, 2021): “GP algorithm that learns ensembles efficiently (e.g., without repeating multiple evolutions) and is simple enough to be thought as a possible minimal/natural extension of classic GP”. In this new algorithm called Simple Simultaneous Ensemble Genetic Programming (2SEGP), there were only modifications in fitness evaluations and selection in order to achieve two aspects considered the essence of this algorithm: “(i) Evaluate a same individual according to different realizations of the training set (i.e., bootstrap samples); and (ii) Let the population improve uniformly across these realizations” (Virgolin, 2021). With this, it was concluded that small changes to the classic GP led to the evolution of bagging ensembles efficiently and effectiveness and therefore, it was argued that GP is naturally suitable for evolving bagging ensembles, having almost zero computational cost.

The fourth example shows that the use of GP can improve imbalanced data classification, a usual problem. So, to overcome the unsolved challenges of imbalanced data, (Roknizadeh & Naeen, 2021) proposed a method to improve the efficiency of the ensemble method in the sampling of training sets, with focus on minority classes, and to determine better basic classifiers for combining classifiers than the already existing ones. This approach is a hybrid ensemble algorithm based on GP for two classes of imbalanced data classification, being the details explained in (Roknizadeh & Naeen, 2021). One of the conclusions of this study was that Bagging and Boosting, Standard GP, NB and Support Vector Machines (SVM) are vulnerable to bias learning because of the majority class, while the proposed method is not.

The fifth example is a feature learning framework for image classification (Bi et al., 2021b) with the objective of overcoming the limitation of high computational cost when using GP-based methods on large-scale image classifications. The main characteristics of this approach, called divide-and-conquer feature learning (DCFL), are the reduction of computational time by splitting the training set into nonoverlapping subsets and the population into several small populations, with the possibility of using different GP methods; the improvement of the learning process by the knowledge transfer (KT) across these multiple subsets and populations; a new log-loss-based fitness function that leads to more accurate information on the effectiveness of the learned features in guiding the GP search; and a new ensemble formulation strategy that achieves high generalization performance by selecting better trees and calculating the weights of the classifiers built from the selected trees. Nonetheless, this new approach is also called DCFL-FGP, referring to the use of DCFL framework along with FGP tree representation (tree representation of the GP approach with image-related operators). The conclusions of this study were satisfactory since this new framework improved the computational efficiency of GP-based feature learning algorithms without losing the generalization performance in image classification.

The sixth example refers to a topic that has been growing in the last years, the software fault prediction. In addition to a review of the literature that uses ML to find faults and errors to establish a level of quality in software, (Sahu et al., 2022) also propose a hybrid approach using GP and ensemble learning techniques to predict software faults. The comparison of this new approach with standard ensemble techniques (such as Adaboost Classifier with Decision Tree (DT) as base estimator or even with Ridge, Bagging Classifier with DT, among others) revealed that this new approach outperformed all the others. However, some limitations include the increased time and space complexity along with the range of predictive accuracy.

Considering the last topic, related work can be evolving NN with GP. Artificial Neural Network (ANN) is one type of ML model inspired in the behavioural and adaptive features of biological nervous systems. It is composed by a set of interconnected neurons.

(Floreano et al., 2008) reviews the existing methods for evolving ANNs, from evolving the network architecture and its parameters to neuroevolution experiments, focusing on advances in the synthesis of learning architectures. It is necessary to define the architecture or topology of the NN by choosing the number of neurons and the possible interconnections, which is a challenging task. To facilitate and achieve better results, learning algorithms are used to find satisfactory parameters for these ANNs. In this sense, EA has the advantage of the NN's defining features be genetically encoded and co-evolved at the same time along with the flexibility that using a performance measure gives instead of defining an error function.

Neuroevolution can be described as the process of training ANNs in an effectively and efficiently way using EAs and it has become more and more popular. Although different applications exist and different evolutionary optimization techniques lead to good results, it is not obvious which technique is the best. In this sense, (Jalali et al., 2020) used four EAs algorithms, Moth-flame optimization (MFO), Multi-Verse Optimizer (MVO), Cuckoo Search (CS) and Particle Swarm Optimization (PSO), to train a multilayer perceptron (MLP), which is one type of ANN. Additionally, to validate the results obtained by evolutionary-based MLP trainers, two gradient descent algorithms are used, Back-propagation (BP) and Levenberg Marquardt (LM). After comparing these models in terms of optimization of weights and bias for a classification dataset regarding navigation tasks of autonomous mobile robot, it was concluded that MFO-based MLP trainers achieve the best results.

Neuroevolution can also refer to the processes of automated configuration and training of Deep Neural Network (DNN) using EAs. However, the need of a comprehensive survey about using neuroevolution approaches in DNN with its advantages and disadvantages led to the comprehensive survey, discussion and evaluation of the state-of-the-art in using EAs for automated configurations and training of EAs (Galvan & Mooney, 2021). First, the authors present the evolution of DNNs architectures with EAs, mentioning Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNNs), among others. Secondly, the use of EAs for training DNNs is analysed.

As mentioned before, imbalance problems are usual and when combined with ANN, lead to more training biases and variances, decreasing the performance of the algorithm. So, to overcome this problem of imbalance classifications, (Al-Badarneh et al., 2020) introduced three stochastic and metaheuristic algorithms for training MLP NN, Grey Wolf Optimization (GWO), Particle Swarm Optimization (PSO) and Salp Swarm Algorithm (SSA), and trained them on different objective functions, accuracy, f1-score and g-mean, using ten datasets. Although there was not an algorithm that outperformed the others, f1-score and g-mean give an advantage over accuracy for imbalanced datasets. If the minor class is more important, then f-measure should be used as the objective function. If the objective is to improve recall of major and minor classes, then g-mean should be used.

4. METHODOLOGY

In this work, a new version of TPOT is presented, in short TPOT-GSGP, focused on regression. It differs from the original TPOT in the models available in model selection's step since there is the possibility of choosing GSGP. Figure 4.1 shows where GSGP was added in TPOT pipeline.

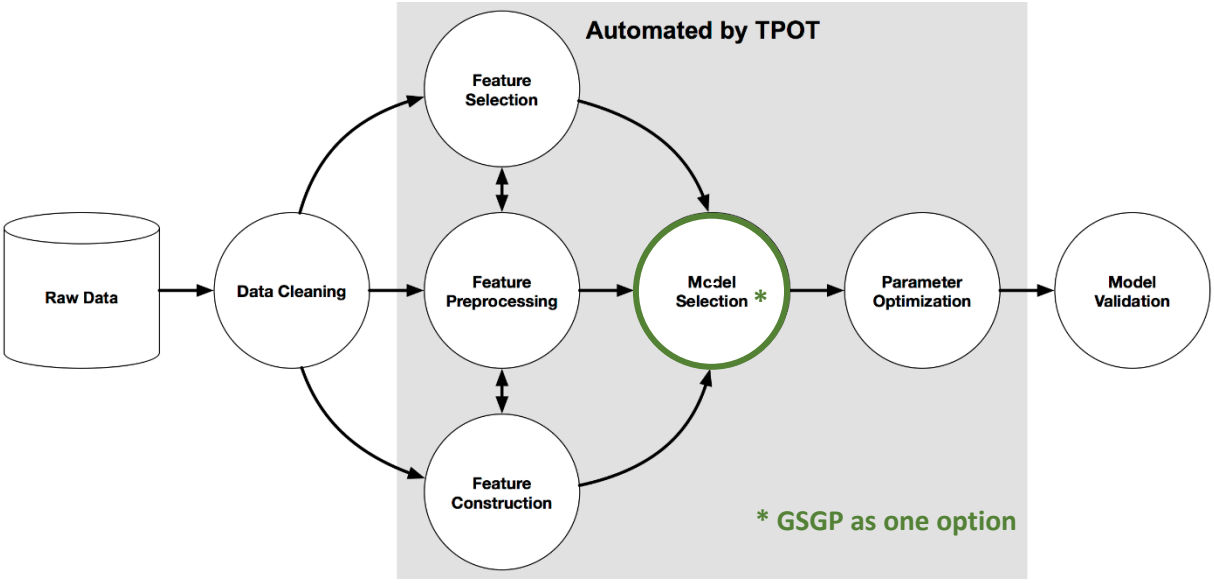


Figure 4.1 - New version of TPOT pipeline with GSGP as one possible model, also called TPOT-GSGP.

This new algorithm TPOT-GSGP was implemented in Python using already existing versions of TPOT (Olson et al., 2016), more concretely version 0.11.7 ((*GitHub - EpistasisLab/Tpot at 6448bdb71ba08b4a0447c640d2f05a05e1affc21*, n.d.)), and GSGP (*GitHub - Jespb/Python-GSGP: My Implementarion of the Geometric Semantic Genetic Programming (GSGP) Algorithm.*, n.d.).

The library TPOT has TPOTClassifier() for classification problems and TPOTRegressor() for regression problems. Being this work focused on regression, the last one was used to implement the new algorithm TPOT-GSGP.

The library GSGP was created for classification problems, so in order to use it in this work, adjustments were made to support regression problems. Additionally, before incorporating GSGP into TPOT and with the objective of coherence, the following changes were made in GSGP algorithm:

- measure of error of GSGP from RMSE to NMAE, to be the same of TPOT;
- since using RMSE as measure of error lead to minimization problem, changing the measure of error to NMAE it turns into a maximization problem;
- elimination of the file that reads the dataset and calls GSGP with posterior implementation of these actions in another file to simplify it;
- copy of the class Population into the file of GSGP;
- creation of several methods to return specific details of the solution, such as the tree or the score according to each generation;
- possibility to export statistics and details when running the algorithm.

After these changes and being all details coherent, GSGP was incorporated in TPOT. However, TPOT was also changed in the following aspects:

- on the step of initialization of the population, creating at least one individual with only GSGP since in the original TPOT this was not guaranteed;
- how the statistics of each individual and population along the generations are saved;
- creation of several methods to return specific details of the solution, such as the tree or the score according to each generation;
- possibility to export statistics and details when running the algorithm.

The main characteristics of TPOT that were checked include the maximum depth of trees and consideration of solution's tree size when evaluating the solutions. It was also assured that elitism was implemented in TPOT, which happen due to the NSGA-II Selection method (Deb et al., 2002), and that only uses the accuracy and does not consider the size.

Figure 4.2 represents one possible initial population with 5 individuals of TPOT-GSGP, having at least one individual with GSGP only (Individual 5). Then, this population is evolved using crossover or mutation. At the end, after fitting the model with a certain number of generations, such as 10, the final solution is found. Figure 4.3 shows three possible solutions, one without FeatureUnion and the other two with FeatureUnion that differ in having or not the StackingEstimator.

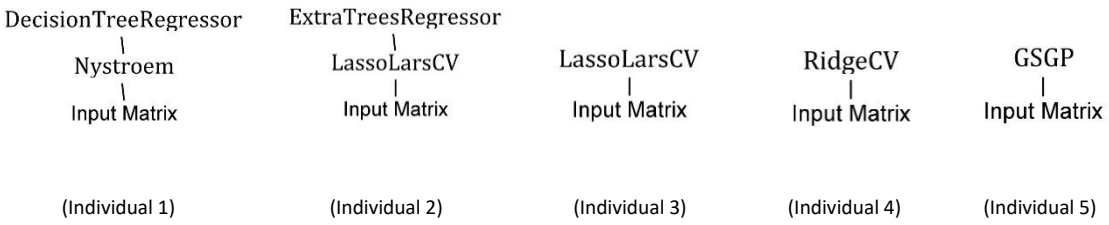


Figure 4.2 - Example of a possible initial population with 5 individuals in TPOT-GSGP.

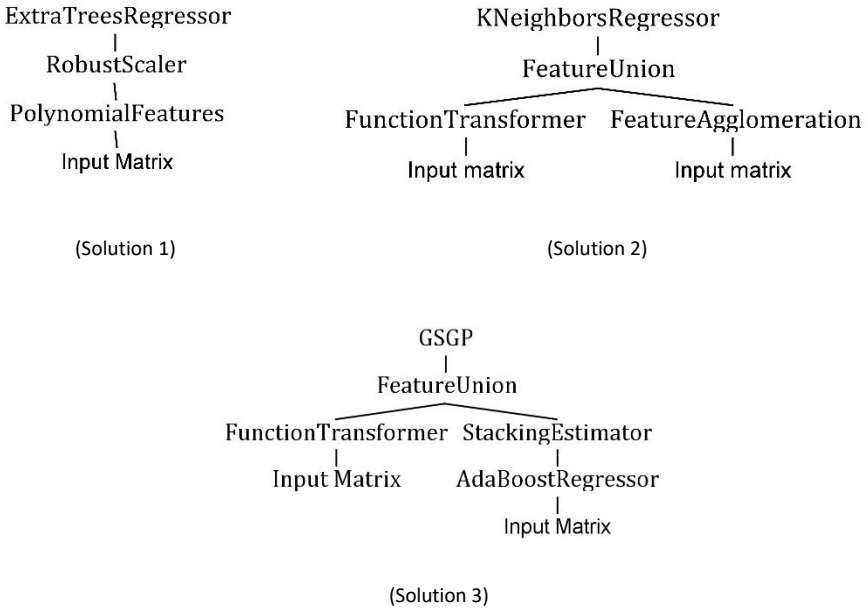


Figure 4.3 - Example of three possible solutions using TPOT-GSGP.

5. EXPERIMENTAL STUDY

5.1. EXPERIMENTAL SETTING

Each dataset was split into a learning set (70%) and a test set (30%). The process of fitting the model included a 5-fold cross validation, where the learning set was divided into five parts, four of those were considered as the training set and the other as the validation set. For each one of the five possible combinations, each model was trained on the training set and evaluated on the training and validation sets, returning the training and validation scores, respectively. Additionally, at each generation, the best individual (model) of the population considering the mean validation score was fitted on the learning set. Then it was tested on the learning and test sets, resulting on the learning and test scores, respectively.

The measure of error used was the Negative Mean Absolute Error (NMAE), which is computed as:

$$NMAE = -\frac{\sum_{i=1}^{N} | Y_i - Y_i^p |}{N}$$

Where N is the total number of instances, Y_i is the true value and Y_i^p is the predicted value. The objective is to maximize this score. So, the closer to zero the better.

Firstly, with the objective of optimizing the GSGP's mutation step in the root, it was performed a grid search, using a 5-fold cross validation, for each dataset with the values 0.05, 0.2, 0.5 and 1.0. This grid search was repeated 10 times, with different random seeds. The parameters used in TPOT-GSGP are reported in Table 5.1, where most of them were the default ones, and some were changed, for instance the number of generations to 10, the population size to 5 and the config_dict that was the parameter that will be used in grid search. The parameters of GSGP, reported in Table 5.2, were the default ones, except for the population size and maximum generations that were 40 and 20, respectively, and the mutation step. Moreover, the mean of validation scores and the mean of training time were analysed and the best value for each problem was found.

Default parameters		Changed parameters	
offspring_size	None	generations	10
mutation_rate	0.9	population_size	5
crossover_rate	0.1	scoring	neg_mean_absolute_error
cv	5	n_jobs	-1
subsample	1	random_state	i (according to each run)
max_time_mins	None	verbosity	3
max_eval_time_mins	5	config_dict	(grid search)
template	None		
warm_start	FALSE		
memory	None		
use_dask	FALSE		
periodic_checkpoint_folder	None		
early_stop	None		
disable_update_check	FALSE		

Table 5.1 - Parameters used in TPOT-GSGP algorithm, more concretely for TPOT, in grid searches.

Default parameters		Changed parameters	
operators	["+","-","*","/"]	population_size	40
max_depth	6	max_generation	20
tournament_size	10	mutation_step	(grid search)
elitism_size	1		

Table 5.2 - Parameters used in TPOT-GSGP algorithm, more concretely for GSGP's root, in GSGP's mutation step grid search.

Secondly, in order to optimize the GSGP's tournament size in the root, another grid search was performed, using also a 5-fold cross validation, with the values 2, 5 and 10. It was also repeated 10 times with different random seeds. The parameters used in TPOT-GSGP were the same used in the first grid search, Table 5.1. The parameters of GSGP, reported in Table 5.3, were also the same used before, except for the mutation step that was equal to the best value achieved in the first grid search considering each dataset. As in the first grid search, the same analysis was made.

Default parameters		Changed parameters	
operators	["+","-","*","/"]	population_size	40
max_depth	6	max_generation	20
mutation_step	(varies)	tournament_size	(grid search)
elitism_size	1		

Table 5.3 - Parameters used in TPOT-GSGP algorithm, more concretely for GSGP's root, in GSGP's tournament size grid search.

The values for the GSGP's mutation step and tournament size that led to a better score according to each dataset will be reported in the Experimental Results section. Considering these best values for each problem, the TPOT-GSGP algorithm was applied to each dataset 30 times with different random seeds.

5.2. CASE STUDIES

The datasets used in this work to check the performance of the TPOT-GSGP algorithm were Forest Fires (FF), Concrete (CONC), Bioavailability (BIO), Plasma Protein Binding Level (PPB) and Toxicity (TOX).

The Forest Fires dataset was retrieved from Machine Learning Repository (Cortez & Morais, 2007), and it is composed by 510 instances and 10 features (two of them discrete and the others continuous variables) related to real-time and non-costly meteorological data from the Portugal's northeast region. The features are reported in Table 5.4 and include x and y-axis spatial coordinates, four indexes from the FWI (Fire Weather Index) system, temperature, relative humidity, wind speed and rain. The target variable is "area", the burned area of the forest, in hectares, being this a regression dataset.

Variable	Description	Type	Values
X	x - axis spatial coordinate within the Montesinho park map	discrete	1 to 9
Y	y - axis spatial coordinate within the Montesinho park map	discrete	2 to 9
FFMC	FFMC index from the FWI system	continuous	18.7 to 96.20
DMC	DMC index from the FWI system	continuous	1.1 to 291.3
DC	DC index from the FWI system	continuous	7.9 to 860.6
ISI	ISI index from the FWI system	continuous	0.0 to 56.10
temp	temperature (in Celsius degrees)	continuous	2.2 to 33.30
RH	relative humidity (in %)	continuous	15.0 to 100
wind	wind speed (in km/h)	continuous	0.40 to 9.40
rain	outside rain (in mm/m2)	continuous	0.0 to 6.4
area	the burned area of the forest (in ha)	continuous	0.00 to 1090.84

Table 5.4 - Description of the attributes in Forest Fires dataset.

The Concrete dataset (Vanneschi et al., 2018) refers to the concrete strength used to build houses. Concrete is made from a mixture of several components (broken stone or gravel, sand, cement, water and other possible aggregates) that can be spread or poured into moulds to form a stone-like mass on hardening. There are various ways of mixture these components and the choice of the right one is made according to the objectives and needs of the project because each combination provides different characteristics and performance. The requirements of each project (for example the resistance, aesthetics and the local legislations and building codes) need to be fulfilled to guarantee that the right mixture can support the weather conditions, the required strength of the material, the cost of different aggregations, among others. Recently, in this area of concrete construction industry, high-performance concrete (HPC) has been become important. The difference between the conventional concrete and this HPC is that the latter is a very complex material since it integrates the basic ingredients plus supplementary cementitious materials (fly ash, blast furnace slag and chemical admixture, like superplasticizer). Therefore, finding the right combination of each material that composes HPC is a hard task. This concrete dataset is composed by 1030 instances and 8 features, including cement, fly ash, blast furnace slag and superplasticizer, being these mixtures a fair representative group for all of the major parameters that influence the strength of HPC. Therefore, the objective of this regression dataset is to predict the strength of HPC.

The other three datasets, Bioavailability (BIO), Plasma Protein Binding Level (PPB) and Toxicity (TOX) (Archetti et al., 2007), are pharmacokinetic parameters that need to be analysed for drug discovery and development, considering a set of molecular structures of a new potential drug.

The Bioavailability (BIO) dataset refers to the human oral bioavailability, the parameter that calculates the percentage of the initial drug dose that effectively reaches the systemic blood circulation after passing the liver, being a representative measure of the quantity of active principle that effectively can actuate its therapeutic effect. Additionally, since the preferred way to supply drugs to patients is the oral way, this parameter is relevant. This dataset is composed by 359 instances and 241 features.

Plasma Protein Binding Level (PPB) is the percentage of the initial drug dose that reaches blood circulation and binds the proteins of plasma, being essential for good pharmacokinetics for two reasons. The first one is that the drug distribution is commonly distributed into human body through blood circulation. And the second one is that only free (un-bound) drugs are able to permeate the membranes in order to reach their targets. The binding level is determined by several proteins in the

blood (for example, albumin, lipoproteins, among others) that are bound by pharmacological molecules. Therefore, the plasma protein binding level in this dataset is defined as the binding levels between the drug and all the proteins contained in the plasma. This dataset is composed by 234 instances and 627 features.

Toxicity (TOX) dataset corresponds to Median Lethal Dose (LD50), one of the parameters that measures the toxicity of a given compound. This LD50 is the amount of compound required to kill 50% of the test organisms, usually expressed as the number of milligrams of drug per one kilogram of mass of the model organism (mg/kg). There is a spectrum of LD50 protocols according to the choice of the specific organism (rat, mice, dog, monkey and rabbit are the usual ones) and the precise way of supplying (intravenous, subcutaneous, intraperitoneal, oral generally) in the experimental design. The most used protocol, LD50 measured in rats with the compound supply via oral, is used in this work. This dataset is composed by 131 instances and 627 features.

In this sense, all these regression datasets (BIO, PPB and TOX) have the objective of predicting the value of human oral bioavailability, plasma protein binding level and the toxicity of a candidate new drug using features that are a set of molecular structures of a new potential drug.

5.3. Experimental Results/Discussion

The results of the first grid search regarding GSGP's mutation step for each problem are represented in Table 5.5, where it can be seen the number of runs in which the value(s) led to a best score. For the datasets which had not a single best mutation step value, the training time was analysed. In this sense, it was checked which mutation step value led to a lower training time.

	Mutation step			
	Mean validation score		Mean training time	
	Value	# MAX	Value	# MIN
FF	0.05, 0.2, 0.5, 1	9	**1**	8
	0.2, 1	1	0.5	2
CONC	0.05, 0.2, 0.5,1	9	**1**	6
	0.05	1	0.05	4
BIO	**0.2**	4		
	0.05	2		
	0.05, 0.2, 0.5,1	2		
	0.05, 0.5,1	1		
	1	1		
PPB	**1**	3		
	0.05	2		
	0.05, 0.2, 0.5,1	2		
	0.5	2		
	0.2	1		
TOX	0.05, 0.2, 0.5,1	5	**1**	4
	0.5	2	0.05	3
	1	2	0.2	2
	0.2	1	0.5	1

Table 5.5 - Results of the GSGP's mutation step grid search. # MAX is the number of runs in which the value(s) of mutation step led to the best mean validation score. # MIN is the number of runs in which the value(s) of mutation step led to the best mean training time.

From the analysis of Table 5.5 regarding the GSGP's mutation step in TPOT-GSGP, it can be concluded that a single value did not lead to the best mean validation score for the problems FF, CONC and TOX. Therefore, the mean training time was used to choose the best value for GSGP's mutation step, which was 1. Considering the other problems, the best value achieved was 0.2 for BIO and 1 for PPB.

Using the GSGP's mutation step values mentioned before, the second grid search regarding GSGP's tournament size was performed and the results are represented in Table 5.6, where the analysis has the same approach as Table 5.5.

	Tournament size			
	Mean validation score		Mean training time	
	Value	# MAX	Value	# MIN
FF	10	4	**10**	6
	5	4	2	2
	2	2	5	2
CONC	**10**	6		
	2	1		
	5	3		
BIO	**2**	6		
	10	3		
	5	1		
PPB	**2**	6		
	5	3		
	10	1		
TOX	**10**	5		
	2	1		
	5	4		

Table 5.6 - Results of the GSGP's tournament size grid search. # MAX is the number of runs in which the value(s) of tournament size led to the best mean validation score. # MIN is the number of runs in which the value(s) of tournament size led to the best mean training time.

From Table 5.6, it can be seen that only the FF problem had a tie between values 10 and 5 for the GSGP's tournament size, which led to the mean training time analysis, choosing 10 as the best value. Moreover, for the other problems, the value for GSGP's tournament size was 10, 2, 2 and 10 for CONC, BIO, PPB and TOX, respectively.

Table 5.7 summarizes the results of both grid searches, the values for the GSGP's mutation step and tournament size according to each case study.

	Best parameters	
	Mutation Step	Tournament Size
FF	1	10
CONC	1	10
BIO	0.2	2
PPB	1	2
TOX	1	10

Table 5.7 - Result of both grid searches: best results for the GSGP's mutation step and tournament size.

Considering the values in Tables 5.1 and 5.2, with the exception of the ones presented in Table 5.7, the TPOT-GSGP algorithm was applied to each problem 30 times changing the random seed. In each run at each generation, the best individual was achieved according to the validation score. Then, it was fitted on the learning data. After that, it was evaluated on the learning and test data to get the learning score and test score, moving then to the next generation and so on.

The TPOT algorithm was also applied with the same number of runs, by changing the random seed and with the same steps in each generation.

Moreover, the GSGP algorithm was also applied 30 times by changing the random seed and with the same values for the mutation step and the tournament size as the ones used on the GSGP's root of TPOT-GSGP (represented in Table 5.7), being the values of the other parameters represented in Table 5.2 or 5.3. With the objective of making the comparison fair, the number of generations used for GSGP was 1000, because if TPOT-GSGP always had GSGP on the root it would lead to 10 generations with 5 individuals that would have 20 generations and 40 individuals, what comes down to 10x5x20x40, that is 1000 generations.

The results of all of these runs are showed on Figures 5.1 to 5.5, where the mean and the 95% confidence interval of learning and test scores for each problem were calculated. In each legend, the number of pipelines where the root of TPOT-GSGP was GSGP is also reported. Additionally, to compare the 3 algorithms it was necessary to consider the GSGP's generations which corresponds to the multiples of 100, i.e., generations 100, 200, 300 until 1000. Each one of these generation will correspond to the 10 generations of TPOT-GSGP.

Moreover, the statistics of learning and test scores at the final generation (that are also showed in the figures mentioned before) are represented on the Tables 5.8 and 5.9, respectively.

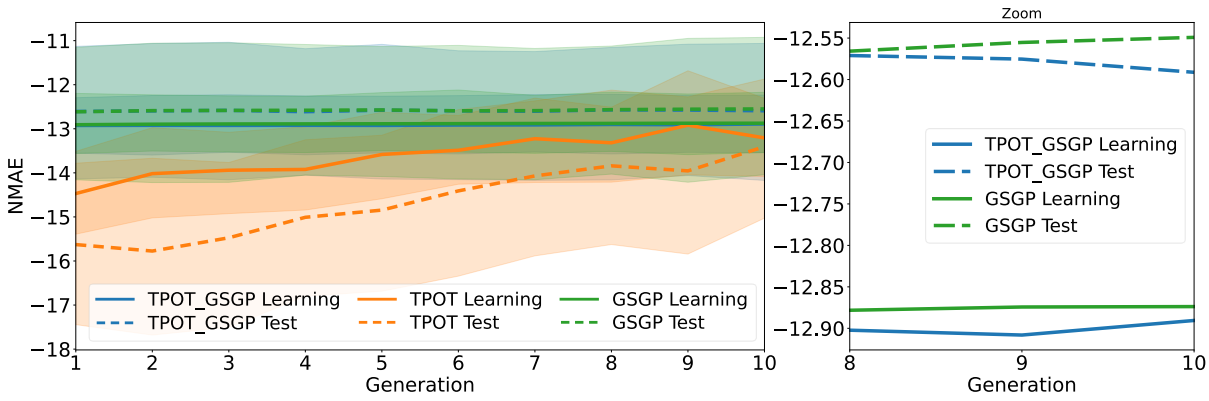


Figure 5.1 - Experimental comparison between TPOT-GSGP, TPOT and GSGP for FF problem. TPOT-GSGP had 27 pipelines with GSGP as the root.

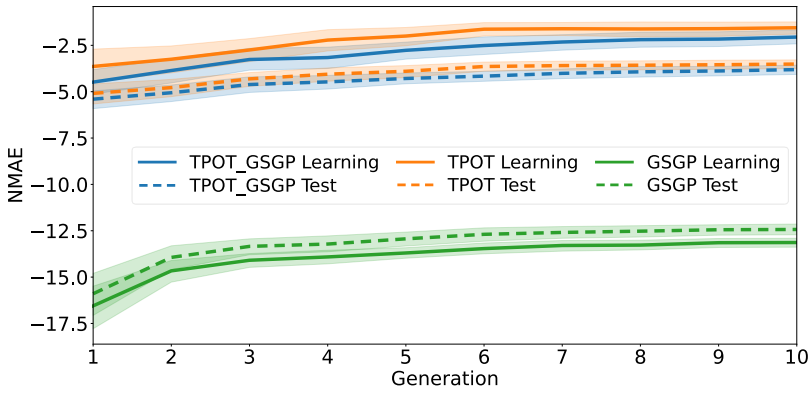


Figure 5.2 - Experimental comparison between TPOT-GSGP, TPOT and GSGP for CONC problem. TPOT-GSGP had 5 pipelines with GSGP as the root.

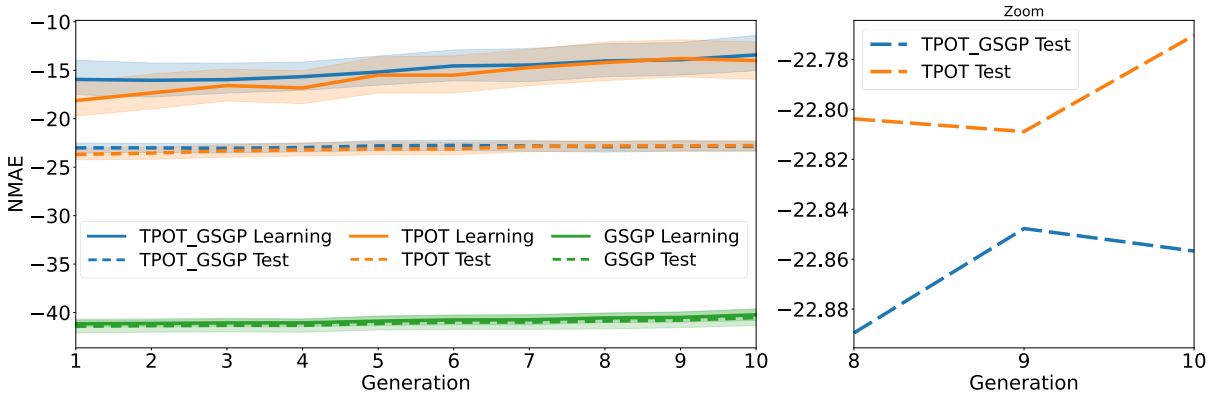


Figure 5.3 - Experimental comparison between TPOT-GSGP, TPOT and GSGP for BIO problem. TPOT-GSGP had 2 pipelines with GSGP as the root.

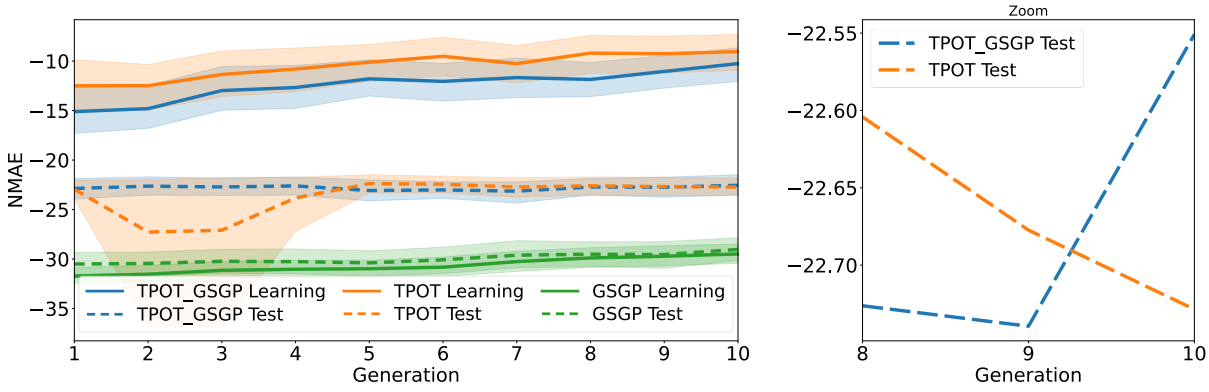


Figure 5.4 - Experimental comparison between TPOT-GSGP, TPOT and GSGP for PPB problem, TPOT-GSGP had 0 pipelines with GSGP as the root.

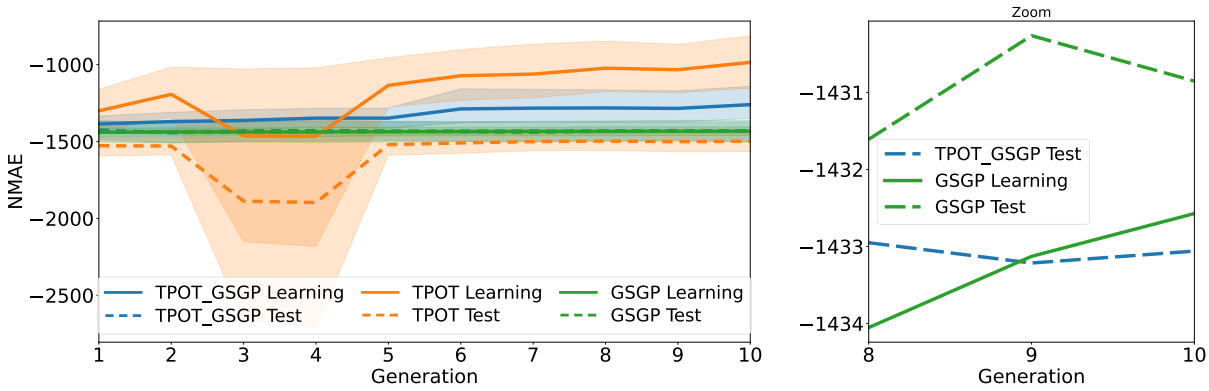


Figure 5.5 - Experimental comparison between TPOT-GSGP, TPOT and GSGP for TOX problem, TPOT-GSGP had 15 pipelines with GSGP as the root.

Problem	TPOT_GSGP			TPOT			GSGP		
	mean	min	max	mean	min	max	mean	min	max
FF	-12.89	-16.05	-8.28	-13.21	-19.98	-8.83	-12.87	-16.02	-8.31
CONC	-2.05	-4.65	-0.24	-1.55	-4.34	-0.12	-13.14	-14.58	-12.30
BIO	-13.41	-19.51	-0.94	-14.00	-22.49	-0.14	-40.24	-42.93	-36.16
PPB	-10.25	-19.66	-2.37	-9.04	-19.06	0.00	-29.49	-34.10	-23.41
TOX	-1259.88	-1576.28	0.00	-985.20	-1493.43	0.00	-1432.57	-1612.47	-1256.93

Table 5.8 - Statistics (mean, minimum and maximum) of learning score in the last generation of each algorithm, TPOT-GSGP, TPOT and GSGP.

Problem	TPOT_GSGP			TPOT			GSGP		
	mean	min	max	mean	min	max	mean	min	max
FF	-12.59	-23.03	-5.39	-13.41	-24.62	-5.38	-12.55	-23.07	-5.30
CONC	-3.81	-5.49	-2.18	-3.51	-4.91	-2.09	-12.43	-14.01	-11.20
BIO	-22.86	-25.64	-19.42	-22.77	-26.68	-19.97	-40.53	-44.41	-34.83
PPB	-22.55	-28.21	-14.04	-22.73	-28.82	-17.96	-29.03	-35.93	-23.61
TOX	-1433.06	-1872.99	-1016.45	-1497.32	-2010.99	-1142.31	-1430.85	-1900.41	-1013.09

Table 5.9 - Statistics (mean, minimum and maximum) of test score in the last generation of each algorithm, TPOT-GSGP, TPOT and GSGP.

Furthermore, for each combination of the three algorithms, TPOT-GSGP, TPOT and GSGP, a Wilcoxon test was realized with the objective of checking the statistical significance of the difference on the last generation's score, that is a two-tailed test with a 95% of confidence. Therefore, the hypothesis used are:

$$H_0: A = B \; vs \; H_1: A \neq B$$

where A and B are the combinations mentioned before that are also explicit in Table 5.10. Since these statistical tests have the objective of comparing three algorithms, the Bonferroni correction was

applied, changing the p-value from 0.05 to (0.05/3). Therefore, it was considered that the difference between two algorithms is statistically significant if the p-value is smaller than 0.01(6).

Data	Learning/Test	A	B	Statistic	P-value	P-value<0.01(6)
FF	Learning	TPOT_GSGP	TPOT	0.28	0.778784	FALSE
FF	Learning	TPOT_GSGP	GSGP	-0.15	0.8824657	FALSE
FF	Learning	TPOT	GSGP	-0.37	0.7116726	FALSE
FF	Test	TPOT_GSGP	TPOT	0.62	0.5346352	FALSE
FF	Test	TPOT_GSGP	GSGP	-0.10	0.9175733	FALSE
FF	Test	TPOT	GSGP	-0.77	0.4420179	FALSE
CONC	Learning	TPOT_GSGP	TPOT	-2.04	0.0413254	FALSE
CONC	Learning	TPOT_GSGP	GSGP	6.65	2.872E-11	TRUE
CONC	Learning	TPOT	GSGP	6.65	2.872E-11	TRUE
CONC	Test	TPOT_GSGP	TPOT	-1.76	0.0785187	FALSE
CONC	Test	TPOT_GSGP	GSGP	6.65	2.872E-11	TRUE
CONC	Test	TPOT	GSGP	6.65	2.872E-11	TRUE
BIO	Learning	TPOT_GSGP	TPOT	0.47	0.6361407	FALSE
BIO	Learning	TPOT_GSGP	GSGP	6.65	2.872E-11	TRUE
BIO	Learning	TPOT	GSGP	6.65	2.872E-11	TRUE
BIO	Test	TPOT_GSGP	TPOT	-0.34	0.7338251	FALSE
BIO	Test	TPOT_GSGP	GSGP	6.65	2.872E-11	TRUE
BIO	Test	TPOT	GSGP	6.65	2.872E-11	TRUE
PPB	Learning	TPOT_GSGP	TPOT	-1.08	0.2804705	FALSE
PPB	Learning	TPOT_GSGP	GSGP	6.65	2.872E-11	TRUE
PPB	Learning	TPOT	GSGP	6.65	2.872E-11	TRUE
PPB	Test	TPOT_GSGP	TPOT	0.06	0.9528424	FALSE
PPB	Test	TPOT_GSGP	GSGP	5.88	4E-09	TRUE
PPB	Test	TPOT	GSGP	5.80	6.812E-09	TRUE
TOX	Learning	TPOT_GSGP	TPOT	-2.81	0.0049689	TRUE
TOX	Learning	TPOT_GSGP	GSGP	3.20	0.0013703	TRUE
TOX	Learning	TPOT	GSGP	5.17	2.285E-07	TRUE
TOX	Test	TPOT_GSGP	TPOT	1.14	0.2549532	FALSE
TOX	Test	TPOT_GSGP	GSGP	-0.11	0.911709	FALSE
TOX	Test	TPOT	GSGP	-1.14	0.2549532	FALSE

Table 5.10 - Final generation statistical tests of each combination of the algorithms, TPOT-GSGP, TPOT and GSGP.

From the information reported in the Figures 5.1 to 5.5 and in the Tables 5.8, 5.9 and 5.10, regarding the scores and the statistical tests, two analyses were made to determine which was the algorithm with the best performance in the learning set and in the test set: the first one (Table 5.11) considers three algorithms (TPOT-GSGP, TPOT and GSGP), and the second one (Table 5.12) only two algorithms (TPOT-GSGP and TPOT). In these two analyses, if the scores have the same integer value, it is considered that the scores are the same.

Problem	Learning	Test
FF	TPOT-GSGP/GSGP	TPOT-GSGP/GSGP
CONC	TPOT	TPOT-GSGP/TPOT
BIO	TPOT-GSGP	TPOT-GSGP/TPOT
PPB	TPOT	TPOT-GSGP/TPOT
TOX	TPOT	GSGP

Table 5.11 - Best algorithm after comparison between the mean of learning and test scores of the three algorithms in the last generation, TPOT-GSGP, TPOT and GSGP.

Firstly, it will be presented the analysis that compares the performance of the three algorithms in the learning set.

The Figure 5.1 shows that, regarding FF dataset, TPOT-GSGP and GSGP outperform TPOT. Comparing TPOT-GSGP and GSGP, the NMAE integer part is the same. Regarding the statistical tests, none of the differences is statistically significant.

Taking into consideration the CONC problem, represented in Figure 5.2, the TPOT and TPOT-GSGP algorithms outperform the other one, GSGP, being the best score achieved by TPOT. Additionally, the differences between TPOT-GSGP and GSGP, and TPOT and GSGP are statistically significant. In opposition, TPOT-GSGP and TPOT do not have differences that are statistically significant.

For BIO dataset, it can be seen from Figure 5.3 that, as in CONC dataset, the algorithm that lead to the worst results was GSGP. All the statistical tests that included GSGP were statistically signficant. Comparing TPOT-GSGP and TPOT, the first one outperform the second one but these two algorithms had not statistical significant differences.

Taking into account the PPB problem, which results are represented in Figure 5.4, the GSGP was also the algorithm with the worst performance, and the results of statistical tests were the same as the BIO data. However, in PPB dataset, the TPOT algorithm outperform TPOT-GSGP in contrary to the BIO dataset.

Regarding the Figure 5.5, concerning the results of TOX dataset, the TPOT had the best learning score, TPOT-GSGP the second-best and GSGP the worst. All the statistical tests on learning data revealed that the differences were statistically significant.

Secondly, it will be analysed the test score taking into consideration also the three algorithms.

From Figure 5.1 it can be seen that for FF dataset, TPOT had the worst performance on the test set. Moreover, from Table 5.8, it can be considered that the other two algorithms, TPOT-GSGP and GSGP, had the same results since the integer part of NMAE is the same, where 27 out of 30 pipelines of TPOT-GSGP had GSGP as root. Although none of these differences is statistically significant, it can be said that GSGP itself and also as root of TPOT-GSGP outperform TPOT in this case study.

For CONC problem, the Figure 5.2 shows that TPOT-GSGP and TPOT outperform GSGP on test set, being the differences between GSGP and the other two algorithms statistical significant. However, it can be considered that the two best algorithms had the same performance, being the difference not statistical significant. For this dataset, the addition of GSGP as one possible root of TPOT-GSGP and

GSGP by itself did not improved the results, however GSGP by itself had worst performance while GSGP being a possible root led to the same performance, having TPOT-GSGP only 5 pipelines with GSGP as root.

For BIO dataset, it can be seen from Figure 5.3 that the algorithm that lead to the worst results was GSGP. Additionally, Table 5.10 shows that all the statistical tests that included GSGP were statistically signficant. Comparing TPOT-GSGP and TPOT perfomances, the difference is only at the first decimal place which can be considered as the same result and TPOT-GSGP had only 2 pipelines with GSGP as root. Therefore, GSGP by itself have worst performance but when is a possible root of TPOT-GSGP led to the same result, as in CONC problem.

Taking into account the PPB problem, which results are represented in Figure 5.4, GSGP was also the algorithm with the worst performance and the other two algorithms had also the same performance, leading to the same results of statistical tests as the CON and BIO datasets. Moreover, TPOT-GSGP had 0 pipelines with GSGP as root. As the previous two mentioned problems, the same can be said about the performance of GSGP by itself, that had worst results, and about TPOT-GSGP with GSGP as one possible root, that led to the same result.

Regarding the Figure 5.5 concerning the results of TOX dataset, GSGP had the best performance on test set, followed by TPOT-GSGP that had the second best with 15 pipelines where GSGP was in root, and TPOT being the worst. Contrary to what happened in the learning set, all the differences on the test set are not statistically significant as it can be seen in Table 5.10. This dataset is the only one where all the algorithms had different results when considered the integer value of NMAE (Table 5.9). It can be said that for this case study GSGP by itself and as one possible root of TPOT-GSGP improved the results, being GSGP the best.

The best algorithm or algorithms considering the comparison between the three can be seen in Table 5.11. For the learning set, TPOT-GSGP and GSGP are the best ones for FF dataset, TPOT-GSGP for BIO dataset and TPOT for the other problems, CONC, PPB and TOX. For the test set, TPOT-GSGP and GSGP are the best algorithms for FF problem, TPOT-GSGP and TPOT for CONC, BIO and PPB problems, and GSGP for TOX problem.

Problem	Learning	Test
FF	TPOT-GSGP	TPOT-GSGP
CONC	TPOT	TPOT-GSGP/TPOT
BIO	TPOT-GSGP	TPOT-GSGP/TPOT
PPB	TPOT	TPOT-GSGP/TPOT
TOX	TPOT	TPOT-GSGP

Table 5.12 - Best algorithm after comparison between the mean of learning and test scores of two algorithms in the last generation, TPOT-GSGP and TPOT.

Thirdly, another analysis will be presented with the objective of comparing two algorithms, TPOT-GSGP and TPOT, regarding the learning score.

For the FF and BIO problems, TPOT-GSGP outperform TPOT in the learning score. Contrary, for the other problems, CONC, PPB and TOX, TPOT outperform TPOT-GSGP in the learning set. The differences are not statistically significant for all datasets, except for TOX where the difference between TPOT-GSGP and TPOT is statistically significant.

Fourthly, the same analysis of comparing two algorithms, TPOT-GSGP and TPOT, will be presented but this time evaluating the test score.

All the differences between TPOT-GSGP and TPOT on test set are not statistically significant, as reported in Table 5.10.

Additionally, from Table 5.9, it can be seen that for CONC, BIO and PPB problems, the test score between TPOT-GSGP and TPOT had the same integer value, meaning that the results were the same and that there is not one best algorithm between TPOT-GSGP and TPOT, as reported in Table 5.12.

However, taking into consideration FF and TOX datasets, TPOT-GSGP outperform TPOT. As mentioned before, the number of pipelines with GSGP as root in TPOT-GSGP were 27 for FF and 15 for TOX. So, it can be said that these datasets are the ones with the large number of pipelines with GSGP as root. Therefore, it can be concluded that every time GSGP is chosen as root 50% of the times or more, TPOT-GSGP outperform TPOT on test set.

There are two differences between the best algorithm(s) when comparing the three algorithms (Table 5.11) and when comparing two algorithms (Table 5.12). The first one is that in Table 5.11 when there were two best algorithms and one of them was GSGP, in Table 5.12 the other one became the best algorithm. The second one is that when GSGP was considered the best algorithm in Table 5.11, which happened only once, TPOT-GSGP become the best algorithm in Table 5.12.

6. CONCLUSION

Nowadays, ML algorithms are becoming part of our daily life. Being present on the recommendation systems on online stores, on face recognition for unlock our phones, on the sensors of latest cars, on the analysis of the outcomes of medical treatments, among many other applications, these algorithms can be divided into supervised or unsupervised learning. The former is characterized by having a dependent variable, also called target, while the latter is characterized by not having it. In supervised learning, the data can be represented by a matrix $n * p$ with n instances and p features, and a n-dimensional vector to represent the target variable. There are different ways of partitioning the data and the choice is usually based on the size of the dataset.

Each step of the ML pipelines can be performed using several options, which makes this process a time-consuming and expensive one, even for those with experience in ML algorithms, expert knowledge of the tool and problem domain, and use of existing brute force techniques. Knowing which choice can be the best for each type of problem avoids testing all the possibilities, although still maintaining this process a time-consuming and expensive one. In this sense, AutoML was introduced to overcome this issue by automating some parts of this ML process. One of the recent algorithms in this area is TPOT, an EA that automatically designs and optimizes ML pipelines using GP.

An EA is an algorithm based on the Theory of Evolution of Charles Darwin, where the individuals need to compete to survive and therefore, it is characterized by having a specific recombination (also called crossover), mutation and selection operators. These operators can be applied with a specific probability or in every cycle, being each consecutive cycle a generation. In a brief way, the EA starts with the initialization of the population with random candidate solutions (also called individuals) and their evaluation. Then, until the defined termination conditions are satisfied, it selects parents, performs the crossover of the pair of parents, mutates the resulting offspring, evaluates the new individuals created and selects the individuals for the next generation. When the termination conditions are satisfied, the best solution (individual) is returned. EA can be divided into different categories, being one of them GP, that is “a domain-independent problem-solving approach in which computer programs are evolved to solve, or approximately solve, problems”(Koza, 1997). In GP, the population is composed by computer programs, as can be seen by the definition, that can have different sizes and shapes. In tree-based GP, these computer programs are based on the set of terminals and on the set of primitive functions. Adding to these two sets, the fitness measure, the parameters for controlling the run and the method for designating a result and the criterion for terminating the run, they compose the necessary definitions to apply GP to a problem. In this traditional GP, the search operators are applied to the syntactic representation of the programs.

There is another EA that was recently developed, the GSGP (Moraglio et al., 2012), characterized by using the semantics, the vector of outputs of a program on the different training data. It uses geometric semantic operators, which means that the search is doing directly in the space of the semantics of the program, that actually represents the performance of the program, leading to a unimodal fitness landscape. Since the operators proposed by (Moraglio et al., 2012) cause an exponential growth of the individuals, (Vanneschi et al., 2013) proposed a new implementation, saving the necessary operations to create new individuals without actually building them by using tables for saving the initial population, the random trees used for crossover and mutation and the new population, always using the semantics to know the performance of each individual. At the end, only the best individual

(solution) is constructed, decreasing the computational effort and time during the runs since the length of individuals increase along the generations.

In this work, a new version of TPOT was created, called TPOT-GSGP, in which GSGP is one of the options for model selection. This new algorithm, only created for regression problems, was implemented in Python using already existing versions of TPOT and GSGP and making the necessary changes in order to incorporate GSGP into TPOT. One of the changes implemented was the creation of at least one individual with only GSGP on the step of the initialization. It was checked if elitism was already implemented in the original TPOT, which was due to the NSGA-II selection method (Deb et al., 2002). The measurement of error used in this work was NMAE. The datasets were split into a learning set (70%) and a test set (30%). Additionally, a 5-fold cross validation was also used where the learning set was divided into five parts, being four of those considered as the training set and the other as the validation set. Two grid searches with 10 runs each were performed with the objective of optimizing GSGP's mutation step and tournament size for each case study. The case studies used were FF, CONC, BIO, PPB and TOX, being the last three pharmacokinetic parameters, which analysis is required for drug discovery and development.

After getting the best value for GSGP's mutation step and tournament size for each dataset through the grid searches, each algorithm (TPOT-GSGP, TPOT and GSGP) was applied to each problem 30 times with different random seeds. The number of generations of TPOT-GSGP and TPOT was 10 and the population size 5. To have a fair comparison between these algorithms, the number of generations for GSGP were 1000 because if TPOT-GSGP always had GSGP on the root it would lead to 1000 generations, and the population size 40. The learning and test scores of these three algorithms were analysed and Wilcoxon tests were also performed with the objective of checking the statistical significance of the difference on the last generation's score (learning and test scores) for each combination of two algorithms.

Comparing the three algorithms, for FF, TPOT-GSGP and GSGP outperformed TPOT, although none of the differences were statistically significant. For CONC, PPB and TOX, TPOT outperformed the other two algorithms in the learning score, while for BIO, TPOT-GSGP outperformed the others. In terms of test score, TPOT-GSGP and TPOT outperformed the other one for CONC, BIO and PPB, while for TOX, GSGP outperformed the others. In terms of statistical significance, CONC, BIO and PPB had the same result, the differences between TPOT-GSGP and GSGP, and TPOT and GSGP were statistically different, while the other difference was not. For TOX, the differences on the learning score were all statistically significant while on the test score were not.

When comparing only TPOT-GSGP and TPOT, for FF and BIO, TPOT-GSGP outperformed TPOT in the learning score, while for CONC, PPB and TOX, TPOT outperformed the other. Regarding the test score, for FF and TOX, TPOT-GSGP outperformed TPOT and for the other problems, CONC, BIO and PPB, it was not possible to choose only one algorithm as the best.

There are two differences between the first comparison (of three algorithms) and the second one (of two algorithms, TPOT-GSGP and TPOT). One is that when there were two best algorithms being one of them GSGP in the first comparison, that other became the best algorithm in the second comparison. The other difference is that the only time GSGP was considered the best algorithm in the first comparison, it was replaced by TPOT-GSGP in the second comparison.

Additionally, it can be also said that every time GSGP is chosen as root 50% of the times or more, TPOT-GSGP outperformed TPOT on the test set. The base of this conclusion is that TPOT-GSGP outperformed TPOT for FF and TOX, being these datasets the ones with the large number of pipelines with GSGP as root, 27 and 15 respectively.

7. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORK

One of the limitations of this work was the limited time and resources to explore different combinations of parameters for TPOT and also for GSGP, such as the measurement of error, the maximum/minimum depth of the trees, among others. Additionally, there were more two limitations of this work, this time related to computer resources. The process of debugging when implementing the new algorithm (for example when changing the code to incorporate GSGP into TPOT), and the running time of the algorithms, as already expected being TPOT an algorithm that automates some steps of ML pipelines and GSGP an EA that achieves good results by using the power of semantics.

One possibility for future work is to incorporate GSGP in a similar way as TPOT was created for better understanding and cohesion because the versions of TPOT and GSGP used in this work had different bases and ways of thinking since each one had a different author.

Moreover, having the limitations of this work in mind, future work also includes the exploration of a broad space of parameters to know if there are some more adequate that improve the performance of TPOT-GSGP. New algorithms and techniques should also be added to the possible choices in TPOT. Additionally, the implementation of parallel computation using GPU to increase the execution time of the algorithm can be also considered. With more powerful computer resources, the advantages of this new algorithm can be extraordinary.

Improvements of this work can also be done in the part of grid search. The way it is implemented forces the user to modify the code in the files that compose the library. First, the user needs to download the files of this library to create a new dictionary (that is a new 'config_dict') with the algorithms and values that want to be analysed. Then, must add this new dictionary to the possible options in the base code to avoid an error when calling class TPOTRegressor() with the new 'config_dict'. So, future work passes by applying grid search to the parameters of GSGP in a way that the user only needs to call a function instead of downloading and changing the code in the files that compose the library.

Another possible way of improving the TPOT-GSGP can be using GSGP as a feature selection, that is, in the step of feature selection, run GSGP in order to extract the variables that compose the best solution and use them in the following steps of TPOT because GSGP can be seen in terms of dimensionality reduction.

8. REFERENCES

Abbass, H. A., Sarker, R. A., & Newton, C. S. (2002). *Data mining : a heuristic approach*. Idea Group.

Al-Badarneh, I., Habib, M., Aljarah, I., & Faris, H. (2020). Neuro-evolutionary models for imbalanced classification problems. *Journal of King Saud University - Computer and Information Sciences*. https://doi.org/10.1016/j.jksuci.2020.11.005

Archetti, F., Lanzeni, S., Messina, E., & Vanneschi, L. (2007). Genetic programming for computational pharmacokinetics in drug discovery and development. *Genetic Programming and Evolvable Machines*, *8*(4), 413–432. https://doi.org/10.1007/s10710-007-9040-z

Bakurov, I., Castelli, M., Gau, O., Fontanella, F., & Vanneschi, L. (2021). Genetic programming for stacked generalization. *Swarm and Evolutionary Computation*, *65*. https://doi.org/10.1016/j.swevo.2021.100913

Bartz-Beielstein, T., Branke, J., Mehnen, J., & Mersmann, O. (2014). Evolutionary Algorithms. In *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (Vol. 4, Issue 3, pp. 178–195). Wiley-Blackwell. https://doi.org/10.1002/widm.1124

Beyer, H.-G., Brucherseifer, E., Jakob, W., Pohlheim, H., Sendhoff, B., & To, T. B. (2002). *Evolutionary algorithms—terms and definitions*. https://homepages.fhv.at/hgb/ea-glossary/ea-terms-engl.html

Bi, Y., Xue, B., & Zhang, M. (2021a). Genetic Programming-Based Discriminative Feature Learning for Low-Quality Image Classification. *IEEE Transactions on Cybernetics*. https://doi.org/10.1109/TCYB.2021.3049778

Bi, Y., Xue, B., & Zhang, M. (2021b). A Divide-And-Conquer Genetic Programming Algorithm with Ensembles for Image Classification. *IEEE Transactions on Evolutionary Computation*, *25*(6), 1148–1162. https://doi.org/10.1109/TEVC.2021.3082112

Brownlee, J. (2020). *Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python*. Machine Learning Mastery.

Cai, J., Luo, J., Wang, S., & Yang, S. (2018). Feature selection in machine learning: A new perspective. *Neurocomputing*, *300*, 70–79. https://doi.org/10.1016/j.neucom.2017.11.077

Cortez, P., & Morais, A. (2007). *A Data Mining Approach to Predict Forest Fires using Meteorological Data* (pp. 512–523).

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, *6*(2).

Dong, G., & Liu, H. (Eds.). (2018). *Feature Engineering for Machine Learning and Data Analytics*. CRC Press. https://ink.library.smu.edu.sg/sis_research/4362

Eiben, A. E., & Smith, J. E. (2003). *Introduction to Evolutionary Computing* (Second, Vol. 53). springer. https://doi.org/10.1007/978-3-662-44874-8

Flasch, O., Friese, M., Zaefferer, M., Bartz-Beielstein, T., & Branke, J. (2015). *Learning Model-Ensemble Policies with Genetic Programming*.

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: From architectures to learning. In *Evolutionary Intelligence* (Vol. 1, Issue 1, pp. 47–62). https://doi.org/10.1007/s12065-007-0002-4

Galvan, E., & Mooney, P. (2021). Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges. *IEEE Transactions on Artificial Intelligence*, *2*(6), 476–493. https://doi.org/10.1109/tai.2021.3067574

GitHub - EpistasisLab/tpot at 6448bdb71ba08b4a0447c640d2f05a05e1affc21. (n.d.). Retrieved August 5, 2021, from https://github.com/EpistasisLab/tpot/tree/6448bdb71ba08b4a0447c640d2f05a05e1affc21

GitHub - jespb/Python-GSGP: My implementarion of the Geometric Semantic Genetic Programming (GSGP) algorithm. (n.d.). Retrieved August 13, 2022, from https://github.com/jespb/Python-GSGP

Hoffmann, F., Bertram, T., Mikut, R., Reischl, M., & Nelles, O. (2019). Benchmarking in classification and regression. In *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (Vol. 9, Issue 5). Wiley-Blackwell. https://doi.org/10.1002/widm.1318

Holland, J. H. (1992). *Genetic Algorithms*. *267*(1), 66–73. https://www.jstor.org/stable/10.2307/24939139

Hossin, M., & Sulaiman, M. N. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), 01–11. https://doi.org/10.5121/ijdkp.2015.5201

Jalali, S. M. J., Ahmadian, S., Khosravi, A., Mirjalili, S., Mahmoudi, M. R., & Nahavandi, S. (2020). Neuroevolution-based autonomous robot navigation: A comparative study. *Cognitive Systems Research*, *62*, 35–43. https://doi.org/10.1016/j.cogsys.2020.04.001

Koza, J. R. (1992). *Genetic programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Koza, J. R. (1997). *Future Work and Practical Applications of Genetic Programming*. Handbook of Evolutionary computation,H1.

Kuhn, M. , & Johnson, K. (2019). *Feature engineering and selection: A practical approach for predictive models*. CRC Press.

Mitchell, T. M. (Tom M. (1997). *Machine Learning*. McGraw-Hill Education.

Moraglio, A., Krawiec, K., & Johnson, C. G. (2012). Geometric semantic genetic programming. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7491 LNCS*(PART 1), 21–31. https://doi.org/10.1007/978-3-642-32937-1_3

Oh, S., Suh, W.-H., & Ahn, C.-W. (2021). Self-Adaptive Genetic Programming for Manufacturing Big Data Analysis. *Symmetry*, *13*, 709. https://doi.org/10.3390/sym13040709

Olson, R. S., Bartley, N., Urbanowicz, R. J., & Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. *GECCO 2016 - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, 485–492. https://doi.org/10.1145/2908812.2908918

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2012). *Scikit-learn: Machine Learning in Python*. http://arxiv.org/abs/1201.0490

Poli, R., McPhee, N. F., & Vanneschi, L. (2008). Elitism Reduces Bloat in Genetic Programming. *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, 1343–1344. https://doi.org/10.1145/1389095.1389355

Rodrigues, N. M., Batista, J. E., & Silva, S. (2020). Ensemble Genetic Programming. In N. and M. E. and D. F. Hu Ting and Lourenço (Ed.), *Genetic Programming* (pp. 151–166). Springer International Publishing.

Rodríguez, J. D., Pérez, A., & Lozano, J. A. (2010). Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(3), 569–575. https://doi.org/10.1109/TPAMI.2009.187

Roknizadeh, M., & Naeen, H. M. (2021). *Hybrid Ensemble optimized algorithm based on Genetic Programming for imbalanced data classification*.

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. In *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (Vol. 8, Issue 4). Wiley-Blackwell. https://doi.org/10.1002/widm.1249

Sahu, S. P., Reddy, B. R., Mukherjee, D., Shyamla, D. M., & Verma, B. S. (2022). A hybrid approach to software fault prediction using genetic programming and ensemble learning methods. *International Journal of Systems Assurance Engineering and Management*. https://doi.org/10.1007/s13198-021-01532-x

Vaccaro, L., Sansonetti, G., & Micarelli, A. (2021). An Empirical Review of Automated Machine Learning. *Computers*, *10*, 11. https://doi.org/10.3390/computers

Vanneschi, L. (2017). An introduction to geometric semantic genetic programming. *Studies in Computational Intelligence*, *663*, 3–42. https://doi.org/10.1007/978-3-319-44003-3_1

Vanneschi, L., Castelli, M., Manzoni, L., & Silva, S. (2013). A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7831 LNCS*, 205–216. https://doi.org/10.1007/978-3-642-37207-0_18

Vanneschi, L., Castelli, M., Scott, K., & Popovič, A. (2018). Accurate High Performance Concrete Prediction with an Alignment-Based Genetic Programming System. *International Journal of Concrete Structures and Materials*, *12*(1). https://doi.org/10.1186/s40069-018-0300-5

Vikhar, P. A. (2017). Evolutionary algorithms: A critical review and its future prospects. *Proceedings - International Conference on Global Trends in Signal Processing, Information Computing and Communication, ICGTSPICC 2016*, 261–265. https://doi.org/10.1109/ICGTSPICC.2016.7955308

Virgolin, M. (2021). Genetic programming is naturally suited to evolve bagging ensembles. *GECCO 2021 - Proceedings of the 2021 Genetic and Evolutionary Computation Conference*, 830–839. https://doi.org/10.1145/3449639.3459278

Winkler, S., Affenzeller, M., & Wagner, S. (2007). Advanced genetic programming based machine learning. *Journal of Mathematical Modelling and Algorithms*, *6*(3), 455–480. https://doi.org/10.1007/s10852-007-9065-6

Ying, X. (2019). An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, *1168*(2). https://doi.org/10.1088/1742-6596/1168/2/022022

APPENDIX

Results of Grid Search for GSGP's Mutation Step

FF

Run	MS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	0.05	560.24	423.33	0.00	0.01	-12.82	5.58	4.00	-12.71	1.39
1	0.5	888.55	497.32	0.01	0.00	-12.79	5.58	3.00	-12.70	1.39
1	0.2	340.28	94.43	0.00	0.00	-12.79	5.59	1.00	-12.70	1.38
1	1	310.74	76.21	0.00	0.01	-12.79	5.59	1.00	-12.70	1.38
2	0.05	313.38	71.85	0.00	0.00	-13.64	7.20	1.00	-13.58	1.80
2	0.5	316.09	81.59	0.00	0.00	-13.64	7.20	1.00	-13.58	1.80
2	0.2	297.84	68.51	0.01	0.01	-13.64	7.20	1.00	-13.58	1.80
2	1	261.77	60.30	0.00	0.00	-13.64	7.20	1.00	-13.58	1.80
3	0.05	403.40	71.54	0.00	0.00	-12.51	6.49	1.00	-12.19	1.61
3	0.5	400.56	68.85	0.00	0.00	-12.51	6.49	1.00	-12.19	1.61
3	0.2	415.60	71.21	0.00	0.00	-12.51	6.49	1.00	-12.19	1.61
3	1	416.42	76.47	0.00	0.00	-12.51	6.49	1.00	-12.19	1.61
4	0.05	228.84	55.62	0.00	0.00	-12.99	7.67	1.00	-12.93	1.90
4	0.5	231.38	56.87	0.00	0.00	-12.99	7.67	1.00	-12.93	1.90
4	0.2	220.62	54.20	0.00	0.00	-12.99	7.67	1.00	-12.93	1.90
4	1	197.92	51.22	0.00	0.00	-12.99	7.67	1.00	-12.93	1.90
5	0.05	268.09	86.62	0.00	0.00	-12.58	5.70	1.00	-12.48	1.41
5	0.5	267.84	85.60	0.00	0.00	-12.58	5.70	1.00	-12.48	1.41
5	0.2	269.01	85.53	0.00	0.00	-12.58	5.70	1.00	-12.48	1.41
5	1	262.63	88.91	0.00	0.00	-12.58	5.70	1.00	-12.48	1.41
6	0.05	212.45	84.70	0.00	0.00	-14.01	11.65	1.00	-14.00	2.90
6	0.5	211.65	84.26	0.00	0.00	-14.01	11.65	1.00	-14.00	2.90
6	0.2	215.03	85.82	0.00	0.00	-14.01	11.65	1.00	-14.00	2.90
6	1	217.36	81.84	0.00	0.00	-14.01	11.65	1.00	-14.00	2.90
7	0.05	235.19	17.18	0.00	0.00	-13.19	7.00	1.00	-13.11	1.75
7	0.5	230.04	15.38	0.00	0.00	-13.19	7.00	1.00	-13.11	1.75
7	0.2	231.65	15.61	0.00	0.00	-13.19	7.00	1.00	-13.11	1.75
7	1	220.40	19.73	0.00	0.00	-13.19	7.00	1.00	-13.11	1.75
8	0.05	106.66	56.58	0.00	0.00	-12.43	5.94	1.00	-12.41	1.50
8	0.5	107.17	55.87	0.00	0.00	-12.43	5.94	1.00	-12.41	1.50
8	0.2	106.41	56.06	0.00	0.00	-12.43	5.94	1.00	-12.41	1.50
8	1	92.99	40.00	0.00	0.00	-12.43	5.94	1.00	-12.41	1.50
9	0.05	303.57	109.44	0.01	0.01	-11.86	7.23	1.00	-11.64	1.89
9	0.5	323.21	113.57	0.01	0.01	-11.86	7.23	1.00	-11.64	1.89
9	0.2	309.50	108.45	0.01	0.01	-11.86	7.23	1.00	-11.64	1.89
9	1	286.36	95.32	0.01	0.01	-11.86	7.23	1.00	-11.64	1.89
10	0.05	357.83	105.50	0.01	0.01	-11.80	4.64	1.00	-11.74	1.16
10	0.5	335.77	97.33	0.01	0.00	-11.80	4.64	1.00	-11.74	1.16
10	0.2	339.76	104.56	0.01	0.01	-11.80	4.64	1.00	-11.74	1.16
10	1	301.73	68.66	0.00	0.01	-11.80	4.64	1.00	-11.74	1.16

Results of Grid Search for GSGP's Mutation Step

CONC

Run	MS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	0.05	166.31	18.17	0.03	0.01	-4.59	0.65	1.00	-1.50	0.72
1	0.5	167.89	13.80	0.04	0.02	-4.59	0.65	1.00	-1.50	0.72
1	0.2	165.77	14.03	0.03	0.01	-4.59	0.65	1.00	-1.50	0.72
1	1	162.38	14.13	0.02	0.01	-4.59	0.65	1.00	-1.50	0.72
2	0.05	164.93	80.42	0.01	0.01	-4.61	0.92	1.00	-2.26	1.48
2	0.5	163.41	77.96	0.01	0.01	-4.64	0.96	2.00	-2.27	1.51
2	0.2	166.21	81.46	0.01	0.00	-4.75	1.12	3.00	-2.25	1.47
2	1	148.20	65.28	0.01	0.01	-4.75	1.12	3.00	-2.25	1.47
3	0.05	88.26	32.31	0.01	0.01	-3.96	0.56	1.00	-2.18	1.13
3	0.5	89.84	30.19	0.01	0.01	-3.96	0.56	1.00	-2.18	1.13
3	0.2	97.35	47.26	0.01	0.01	-3.96	0.56	1.00	-2.18	1.13
3	1	90.77	30.95	0.01	0.01	-3.96	0.56	1.00	-2.18	1.13
4	0.05	111.56	30.38	0.01	0.01	-4.13	1.07	1.00	-1.92	1.10
4	0.5	115.88	32.00	0.01	0.01	-4.13	1.07	1.00	-1.92	1.10
4	0.2	116.82	34.10	0.01	0.01	-4.13	1.07	1.00	-1.92	1.10
4	1	115.57	31.96	0.01	0.01	-4.13	1.07	1.00	-1.92	1.10
5	0.05	114.88	7.95	0.02	0.01	-5.53	0.93	1.00	-2.44	0.78
5	0.5	121.04	12.08	0.02	0.01	-5.53	0.93	1.00	-2.44	0.78
5	0.2	119.20	10.75	0.01	0.01	-5.53	0.93	1.00	-2.44	0.78
5	1	118.99	11.38	0.02	0.01	-5.53	0.93	1.00	-2.44	0.78
6	0.05	178.76	29.37	0.03	0.01	-5.65	1.13	1.00	-3.25	0.28
6	0.5	174.87	29.85	0.03	0.01	-5.65	1.13	1.00	-3.25	0.28
6	0.2	171.10	28.99	0.03	0.01	-5.65	1.13	1.00	-3.25	0.28
6	1	165.77	24.87	0.02	0.01	-5.65	1.13	1.00	-3.25	0.28
7	0.05	121.87	39.68	0.03	0.02	-5.26	1.27	1.00	-1.55	0.73
7	0.5	131.90	33.92	0.03	0.02	-5.26	1.27	1.00	-1.55	0.73
7	0.2	129.58	36.39	0.02	0.01	-5.26	1.27	1.00	-1.55	0.73
7	1	114.99	23.83	0.03	0.02	-5.26	1.27	1.00	-1.55	0.73
8	0.05	212.71	31.66	0.04	0.02	-5.53	1.36	1.00	-2.05	0.76
8	0.5	218.61	30.23	0.05	0.02	-5.53	1.36	1.00	-2.05	0.76
8	0.2	225.96	29.16	0.04	0.01	-5.53	1.36	1.00	-2.05	0.76
8	1	210.84	11.94	0.04	0.02	-5.53	1.36	1.00	-2.05	0.76
9	0.05	148.25	26.26	0.02	0.01	-4.36	0.94	1.00	-1.93	0.43
9	0.5	152.12	27.04	0.02	0.02	-4.36	0.94	1.00	-1.93	0.43
9	0.2	154.00	26.91	0.02	0.01	-4.36	0.94	1.00	-1.93	0.43
9	1	149.55	28.97	0.02	0.01	-4.36	0.94	1.00	-1.93	0.43
10	0.05	181.67	26.98	0.03	0.02	-5.13	1.06	1.00	-2.43	0.80
10	0.5	190.84	24.85	0.03	0.02	-5.13	1.06	1.00	-2.43	0.80
10	0.2	189.28	26.59	0.02	0.01	-5.13	1.06	1.00	-2.43	0.80
10	1	174.95	19.41	0.02	0.01	-5.13	1.06	1.00	-2.43	0.80

Results of Grid Search for GSGP's Mutation Step

BIO

Run	MS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	0.05	329.02	189.74	0.02	0.01	-22.62	1.61	1.00	-10.01	3.69
1	0.5	297.30	183.81	0.02	0.01	-22.80	1.32	3.00	-9.32	2.94
1	0.2	289.95	184.12	0.02	0.01	-22.82	1.29	4.00	-10.27	4.05
1	1	299.62	141.56	0.02	0.01	-22.67	1.54	2.00	-9.30	2.92
2	0.05	240.26	19.97	0.03	0.02	-23.37	1.92	2.00	-17.18	4.26
2	0.5	239.31	20.67	0.03	0.02	-23.37	1.92	2.00	-17.18	4.26
2	0.2	253.22	33.23	0.05	0.06	-23.44	1.90	4.00	-17.31	4.33
2	1	218.32	31.54	0.03	0.02	-23.32	1.93	1.00	-17.02	4.19
3	0.05	273.62	288.79	0.04	0.02	-23.29	1.70	4.00	-15.01	3.78
3	0.5	325.73	310.48	0.03	0.03	-23.15	2.21	3.00	-13.83	3.81
3	0.2	146.77	75.42	0.01	0.00	-22.74	2.03	1.00	-14.74	2.81
3	1	292.06	326.27	0.03	0.02	-23.11	1.07	2.00	-16.19	2.53
4	0.05	203.91	130.61	0.02	0.02	-24.14	2.86	4.00	-16.48	3.10
4	0.5	164.41	88.33	0.01	0.00	-22.92	1.16	2.00	-17.07	0.63
4	0.2	148.96	21.26	0.02	0.01	-22.83	1.13	1.00	-17.55	1.24
4	1	137.33	26.70	0.02	0.00	-22.92	1.16	2.00	-17.07	0.63
5	0.05	242.70	100.62	0.03	0.02	-22.37	2.35	4.00	-15.16	3.40
5	0.5	249.46	96.55	0.02	0.01	-22.29	2.18	2.00	-15.17	3.38
5	0.2	247.50	94.65	0.03	0.01	-22.01	2.51	1.00	-13.97	3.42
5	1	213.37	49.35	0.02	0.01	-22.35	2.36	3.00	-15.09	3.35
6	0.05	500.21	146.24	0.02	0.01	-23.87	0.88	2.00	-16.83	4.81
6	0.5	564.55	346.72	0.02	0.00	-24.24	0.86	4.00	-12.56	7.15
6	0.2	390.02	273.14	0.01	0.01	-23.78	1.02	1.00	-15.50	4.24
6	1	369.19	158.64	0.02	0.00	-24.07	0.91	3.00	-12.51	7.39
7	0.05	142.11	74.43	0.01	0.01	-24.48	2.35	1.00	-14.32	5.57
7	0.5	134.28	70.15	0.02	0.01	-24.48	2.35	1.00	-14.32	5.57
7	0.2	145.74	76.15	0.02	0.02	-24.48	2.35	1.00	-14.32	5.57
7	1	132.08	70.40	0.02	0.01	-24.48	2.35	1.00	-14.32	5.57
8	0.05	267.06	93.07	0.02	0.01	-23.81	2.03	1.00	-16.07	3.11
8	0.5	257.55	98.78	0.02	0.02	-23.81	2.03	1.00	-16.07	3.11
8	0.2	250.95	99.74	0.03	0.03	-23.81	2.03	1.00	-16.07	3.11
8	1	211.07	72.73	0.02	0.01	-23.81	2.03	1.00	-16.07	3.11
9	0.05	317.37	235.99	0.03	0.01	-22.09	2.30	1.00	-15.20	1.60
9	0.5	194.31	19.53	0.04	0.02	-22.11	2.31	2.00	-15.20	1.60
9	0.2	199.23	24.38	0.04	0.02	-22.11	2.31	4.00	-15.21	1.59
9	1	199.87	31.66	0.03	0.01	-22.11	2.31	2.00	-15.20	1.60
10	0.05	252.38	145.78	0.03	0.02	-21.80	1.69	1.00	-15.14	2.81
10	0.5	236.89	123.82	0.02	0.01	-21.80	1.69	1.00	-15.14	2.81
10	0.2	221.99	118.09	0.02	0.01	-21.97	1.46	4.00	-15.32	2.58
10	1	198.67	48.17	0.02	0.01	-21.80	1.69	1.00	-15.14	2.81

Results of Grid Search for GSGP's Mutation Step

PPB

Run	MS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	0.05	379.83	129.43	0.04	0.02	-21.38	2.05	3.00	-7.05	4.78
1	0.5	386.49	138.85	0.03	0.01	-21.04	2.13	2.00	-6.65	5.10
1	0.2	377.55	139.76	0.05	0.01	-21.38	2.05	3.00	-7.05	4.78
1	1	342.94	109.17	0.03	0.03	-21.03	2.13	1.00	-7.06	4.77
2	0.05	270.88	87.95	0.01	0.01	-24.12	4.93	2.00	-13.48	4.95
2	0.5	284.00	81.04	0.01	0.01	-24.12	4.93	2.00	-13.48	4.95
2	0.2	307.67	121.33	0.01	0.01	-24.76	4.30	4.00	-14.26	5.42
2	1	370.77	183.38	0.01	0.01	-23.28	4.42	1.00	-13.05	7.00
3	0.05	578.21	346.10	0.02	0.02	-22.28	7.86	3.00	-4.73	2.91
3	0.5	396.38	168.58	0.02	0.01	-19.06	5.07	1.00	-6.38	3.40
3	0.2	304.12	57.79	0.02	0.01	-23.14	6.75	4.00	-2.89	4.40
3	1	611.06	158.64	0.02	0.01	-21.80	6.63	2.00	-2.87	1.65
4	0.05	188.63	31.42	0.03	0.01	-21.67	2.29	1.00	-11.97	5.80
4	0.5	195.24	29.49	0.03	0.02	-21.67	2.29	1.00	-11.97	5.80
4	0.2	198.51	35.23	0.02	0.00	-21.67	2.29	1.00	-11.97	5.80
4	1	192.68	26.75	0.02	0.01	-21.67	2.29	1.00	-11.97	5.80
5	0.05	543.90	190.45	0.01	0.01	-25.40	2.90	1.00	-16.41	6.02
5	0.5	961.49	246.38	0.01	0.01	-27.33	3.02	3.00	-16.12	6.37
5	0.2	712.57	477.34	0.03	0.02	-27.43	2.38	4.00	-15.89	7.24
5	1	679.76	318.17	0.02	0.01	-25.75	3.55	2.00	-11.72	7.71
6	0.05	575.60	305.54	0.02	0.01	-22.34	3.27	3.00	-10.32	5.59
6	0.5	309.13	147.91	0.02	0.02	-22.11	3.77	2.00	-6.37	3.28
6	0.2	222.52	70.95	0.03	0.02	-24.42	1.42	4.00	-7.50	3.01
6	1	368.85	165.07	0.02	0.01	-22.03	3.75	1.00	-7.77	0.99
7	0.05	196.11	122.16	0.01	0.01	-28.14	2.53	1.00	-9.13	6.59
7	0.5	185.44	124.32	0.01	0.01	-28.14	2.53	1.00	-9.13	6.59
7	0.2	194.33	121.40	0.01	0.01	-28.14	2.53	1.00	-9.13	6.59
7	1	173.68	122.30	0.01	0.01	-28.14	2.53	1.00	-9.13	6.59
8	0.05	594.87	336.24	0.02	0.02	-20.52	2.50	1.00	-5.34	5.08
8	0.5	595.37	361.81	0.03	0.01	-21.29	5.36	3.00	-8.93	4.32
8	0.2	671.30	361.63	0.02	0.01	-21.41	2.10	4.00	-8.93	4.91
8	1	639.84	330.88	0.02	0.02	-20.90	3.26	2.00	-6.17	4.44
9	0.05	1412.89	800.67	0.03	0.02	-25.03	4.39	2.00	-12.48	5.70
9	0.5	1478.24	1140.99	0.04	0.04	-25.35	4.75	4.00	-14.24	6.77
9	0.2	1182.58	541.74	0.04	0.04	-24.87	4.53	1.00	-15.25	7.17
9	1	1160.68	223.95	0.05	0.07	-25.22	4.67	3.00	-17.90	6.16
10	0.05	372.66	120.23	0.02	0.02	-24.49	4.91	4.00	-11.74	7.11
10	0.5	353.21	79.91	0.03	0.02	-24.01	5.56	1.00	-12.07	6.90
10	0.2	345.38	71.42	0.04	0.02	-24.48	4.90	2.00	-11.69	7.15
10	1	347.36	78.31	0.02	0.01	-24.48	4.90	2.00	-11.69	7.15

Results of Grid Search for GSGP's Mutation Step

TOX

Run	MS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	0.05	510.43	100.97	0.01	0.01	-1561.60	263.99	1.00	-1329.14	240.17
1	0.5	534.09	118.26	0.01	0.01	-1561.60	263.99	1.00	-1329.14	240.17
1	0.2	502.79	117.72	0.02	0.02	-1561.60	263.99	1.00	-1329.14	240.17
1	1	513.07	114.45	0.01	0.01	-1561.60	263.99	1.00	-1329.14	240.17
2	0.05	649.67	298.74	0.01	0.01	-1393.21	119.33	4.00	-1283.11	157.72
2	0.5	763.37	362.92	0.11	0.19	-1376.48	139.56	2.00	-1187.87	231.65
2	0.2	865.42	397.38	0.02	0.01	-1379.05	129.27	3.00	-1299.49	165.20
2	1	697.66	365.35	0.01	0.01	-1369.11	131.79	1.00	-1265.02	154.97
3	0.05	450.97	92.30	0.01	0.01	-1229.11	150.72	2.00	-1135.86	155.48
3	0.5	553.52	125.50	0.11	0.20	-1235.96	136.62	3.00	-1194.22	104.39
3	0.2	747.21	485.09	0.13	0.22	-1227.38	143.94	1.00	-1042.56	171.38
3	1	598.72	198.22	0.28	0.33	-1241.88	135.13	4.00	-1126.07	99.48
4	0.05	554.46	229.38	0.02	0.01	-1434.21	274.64	1.00	-943.45	214.80
4	0.5	549.19	230.28	0.02	0.01	-1434.21	274.64	1.00	-943.45	214.80
4	0.2	540.28	227.31	0.02	0.01	-1434.21	274.64	1.00	-943.45	214.80
4	1	489.33	188.10	0.02	0.02	-1434.21	274.64	1.00	-943.45	214.80
5	0.05	594.07	322.35	0.01	0.01	-1580.49	398.18	3.00	-1337.75	404.17
5	0.5	591.38	248.92	0.02	0.02	-1593.55	386.88	4.00	-1327.91	354.29
5	0.2	574.72	169.38	0.01	0.01	-1559.58	422.94	2.00	-1492.91	217.68
5	1	707.69	238.56	0.02	0.03	-1552.84	349.86	1.00	-1467.02	243.16
6	0.05	591.43	312.92	0.11	0.17	-1348.58	439.04	2.00	-1331.70	132.81
6	0.5	574.66	344.11	0.02	0.01	-1347.66	439.56	1.00	-1338.90	131.30
6	0.2	501.49	318.40	0.01	0.01	-1364.25	449.33	3.00	-1258.01	217.65
6	1	401.48	165.46	0.01	0.01	-1364.25	449.33	3.00	-1258.01	217.65
7	0.05	286.04	47.05	0.01	0.01	-1633.67	306.06	1.00	-1450.01	235.03
7	0.5	291.37	56.95	0.01	0.01	-1633.67	306.06	1.00	-1450.01	235.03
7	0.2	304.50	51.37	0.01	0.01	-1633.67	306.06	1.00	-1450.01	235.03
7	1	251.28	42.59	0.01	0.01	-1633.67	306.06	1.00	-1450.01	235.03
8	0.05	387.37	172.52	0.01	0.01	-1440.02	279.59	1.00	-1177.75	265.31
8	0.5	334.20	118.16	0.02	0.01	-1440.02	279.59	1.00	-1177.75	265.31
8	0.2	344.93	119.56	0.02	0.01	-1440.02	279.59	1.00	-1177.75	265.31
8	1	361.74	164.10	0.01	0.01	-1440.02	279.59	1.00	-1177.75	265.31
9	0.05	423.71	63.30	0.01	0.01	-1544.73	193.77	4.00	-1147.33	243.87
9	0.5	807.36	179.98	0.02	0.01	-1475.65	228.89	1.00	-1086.39	159.99
9	0.2	573.26	127.59	0.01	0.01	-1514.87	198.74	2.00	-1074.72	279.29
9	1	514.35	145.27	0.02	0.01	-1536.38	196.47	3.00	-1127.63	244.09
10	0.05	396.18	83.70	0.01	0.01	-1381.73	302.41	1.00	-1158.47	196.90
10	0.5	398.40	84.39	0.01	0.01	-1381.73	302.41	1.00	-1158.47	196.90
10	0.2	397.22	81.91	0.01	0.01	-1381.73	302.41	1.00	-1158.47	196.90
10	1	359.60	57.73	0.00	0.01	-1381.73	302.41	1.00	-1158.47	196.90

Results of Grid Search for GSGP's Tournament Size

FF

Run	TS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	2	343.26	44.13	0.00	0.01	-12.81	5.58	3.00	-12.75	1.41
1	5	271.82	33.29	0.00	0.01	-12.75	5.55	1.00	-12.74	1.43
1	10	246.36	72.52	0.00	0.00	-12.76	5.60	2.00	-12.75	1.40
2	2	261.72	94.23	0.01	0.01	-13.56	7.22	1.00	-13.54	1.85
2	5	272.92	116.47	0.00	0.01	-13.60	7.20	3.00	-13.58	1.80
2	10	100.62	39.05	0.01	0.01	-13.58	7.22	2.00	-13.57	1.79
3	2	292.78	125.43	0.01	0.01	-12.41	6.52	3.00	-12.22	1.62
3	5	123.88	7.88	0.00	0.00	-12.33	6.53	2.00	-12.21	1.63
3	10	210.03	74.25	0.01	0.01	-12.30	6.57	1.00	-12.20	1.61
4	2	323.27	175.36	0.01	0.01	-12.96	7.68	1.00	-12.95	1.93
4	5	241.66	108.65	0.01	0.01	-13.05	7.62	3.00	-12.87	1.89
4	10	179.66	21.07	0.00	0.01	-13.04	7.72	2.00	-12.96	1.92
5	2	169.54	90.17	0.00	0.00	-12.68	5.70	2.00	-12.47	1.43
5	5	349.32	52.54	0.00	0.00	-12.74	5.78	3.00	-12.47	1.41
5	10	296.71	60.34	0.00	0.00	-12.56	5.74	1.00	-12.48	1.44
6	2	231.33	76.91	0.00	0.01	-14.08	11.65	2.00	-14.01	2.92
6	5	299.93	88.26	0.00	0.00	-14.05	11.69	1.00	-14.00	2.90
6	10	226.58	86.25	0.00	0.00	-25.63	21.63	3.00	-25.52	24.27
7	2	443.95	193.14	0.00	0.00	-13.14	7.02	2.00	-13.11	1.76
7	5	446.45	159.34	0.00	0.00	-13.22	6.94	3.00	-12.96	1.66
7	10	355.26	70.85	0.00	0.00	-13.14	7.00	1.00	-13.10	1.75
8	2	341.50	69.56	0.00	0.00	-12.43	5.96	2.00	-12.40	1.50
8	5	254.30	142.57	0.00	0.00	-12.42	5.94	1.00	-12.39	1.50
8	10	93.86	40.53	0.00	0.00	-12.45	5.92	3.00	-12.41	1.50
9	2	232.29	90.88	0.00	0.00	-11.78	7.16	2.00	-11.66	1.87
9	5	209.82	76.19	0.01	0.01	-11.74	7.17	1.00	-11.65	1.89
9	10	234.21	79.40	0.00	0.01	-11.79	7.17	3.00	-11.66	1.88
10	2	229.71	44.03	0.01	0.01	-11.81	4.66	3.00	-11.72	1.19
10	5	277.83	35.16	0.00	0.00	-11.79	4.65	2.00	-11.72	1.15
10	10	426.47	141.12	0.00	0.00	-11.78	4.64	1.00	-11.74	1.16

Results of Grid Search for GSGP's Tournament Size

CONC

Run	TS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	2	234.88	26.26	0.02	0.01	-4.94	0.84	3.00	-2.63	0.30
1	5	229.53	47.81	0.03	0.01	-4.51	0.75	2.00	-2.13	0.17
1	10	217.46	89.77	0.02	0.00	-4.40	0.82	1.00	-1.92	0.40
2	2	309.86	151.83	0.01	0.02	-5.52	1.93	3.00	-2.25	0.86
2	5	175.85	32.47	0.05	0.03	-4.76	1.14	2.00	-2.27	1.07
2	10	145.78	23.60	0.02	0.00	-4.62	0.81	1.00	-2.24	1.28
3	2	147.32	18.04	0.03	0.01	-3.94	0.60	3.00	-2.37	0.34
3	5	112.63	28.78	0.01	0.01	-3.92	0.57	2.00	-1.96	1.13
3	10	91.42	55.39	0.01	0.01	-3.87	0.44	1.00	-1.87	0.78
4	2	109.97	27.69	0.01	0.01	-4.66	0.74	2.00	-1.65	0.97
4	5	126.27	12.95	0.02	0.01	-4.72	1.10	3.00	-1.75	0.83
4	10	116.20	31.85	0.01	0.01	-4.13	1.07	1.00	-1.92	1.10
5	2	170.98	36.34	0.03	0.02	-5.23	1.06	2.00	-1.22	0.77
5	5	146.59	20.89	0.02	0.01	-5.19	1.64	1.00	-2.33	0.81
5	10	123.24	12.76	0.01	0.01	-5.33	1.05	3.00	-2.00	0.42
6	2	131.03	22.11	0.01	0.01	-5.80	1.03	3.00	-3.13	0.85
6	5	152.94	25.56	0.02	0.01	-5.52	1.59	1.00	-1.59	0.92
6	10	173.78	27.01	0.02	0.01	-5.65	1.13	2.00	-3.25	0.28
7	2	104.99	27.89	0.01	0.01	-5.29	1.23	2.00	-1.56	0.82
7	5	67.63	13.44	0.01	0.01	-5.44	1.23	3.00	-1.82	0.76
7	10	140.48	42.45	0.02	0.01	-5.26	1.27	1.00	-1.55	0.73
8	2	184.33	49.70	0.02	0.01	-4.69	1.12	1.00	-0.92	0.77
8	5	141.42	3.49	0.02	0.01	-5.05	1.36	2.00	-1.61	0.18
8	10	196.06	55.59	0.03	0.02	-5.53	1.35	3.00	-2.14	0.91
9	2	169.32	40.51	0.02	0.01	-5.87	0.72	2.00	-4.45	0.89
9	5	169.08	30.47	0.02	0.01	-5.89	1.88	3.00	-3.16	0.82
9	10	139.89	28.48	0.01	0.01	-5.08	0.65	1.00	-3.32	1.25
10	2	360.89	200.13	0.02	0.01	-4.63	0.89	2.00	-2.31	1.15
10	5	213.93	34.29	0.01	0.01	-4.44	1.06	1.00	-1.90	1.17
10	10	159.30	22.38	0.02	0.02	-5.13	1.06	3.00	-2.43	0.80

Results of Grid Search for GSGP's Tournament Size

BIO

Run	TS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	2	744.29	486.61	0.04	0.04	-22.79	1.50	1.00	-10.00	2.57
1	5	292.39	121.61	0.01	0.01	-22.95	1.32	3.00	-10.65	6.10
1	10	258.59	123.90	0.01	0.01	-22.82	1.29	2.00	-10.27	4.05
2	2	741.99	1224.48	0.10	0.18	-23.86	1.57	2.00	-16.65	3.60
2	5	307.78	153.65	0.03	0.02	-23.90	1.70	3.00	-14.83	4.44
2	10	311.49	169.18	0.02	0.01	-23.55	2.00	1.00	-16.33	3.22
3	2	319.20	348.57	0.02	0.01	-23.03	1.91	1.00	-17.97	3.58
3	5	193.97	88.22	0.01	0.01	-24.67	1.88	3.00	-18.14	3.16
3	10	79.36	26.56	0.01	0.01	-23.32	1.71	2.00	-19.55	2.09
4	2	189.05	17.04	0.02	0.01	-23.28	1.08	1.00	-16.99	1.36
4	5	264.07	76.09	0.02	0.00	-23.33	1.32	2.00	-15.12	4.01
4	10	136.68	21.38	0.01	0.01	-23.40	1.62	3.00	-16.42	3.21
5	2	138.01	28.22	0.02	0.01	-21.83	1.57	1.00	-14.65	1.90
5	5	352.17	201.29	0.01	0.01	-22.77	0.86	3.00	-17.14	4.43
5	10	271.97	181.18	0.01	0.01	-22.56	2.10	2.00	-13.15	4.13
6	2	490.95	344.83	0.19	0.34	-24.13	1.39	3.00	-18.21	3.00
6	5	429.34	419.66	0.02	0.01	-23.24	1.76	1.00	-14.81	1.86
6	10	473.44	277.70	0.01	0.01	-23.77	1.06	2.00	-10.58	6.94
7	2	94.40	23.61	0.02	0.01	-23.58	1.82	1.00	-16.32	2.94
7	5	770.72	648.26	0.02	0.01	-23.96	1.43	2.00	-15.23	3.38
7	10	125.28	60.72	0.02	0.01	-24.48	2.35	3.00	-14.32	5.57
8	2	178.42	66.85	0.02	0.00	-24.85	2.51	3.00	-17.07	2.60
8	5	249.34	90.85	0.02	0.01	-24.09	2.05	2.00	-8.87	4.27
8	10	231.14	58.32	0.01	0.01	-23.83	2.04	1.00	-16.00	3.05
9	2	798.53	1205.97	0.08	0.11	-21.14	1.68	1.00	-12.63	3.49
9	5	2010.44	1145.52	0.18	0.31	-21.67	2.12	2.00	-13.15	2.72
9	10	489.26	475.72	0.02	0.01	-21.98	1.25	3.00	-11.84	6.11
10	2	421.29	487.76	0.19	0.35	-22.03	1.67	2.00	-14.81	3.90
10	5	728.29	423.18	0.26	0.36	-22.37	1.65	3.00	-14.80	3.24
10	10	237.69	73.62	0.01	0.01	-21.80	1.69	1.00	-15.14	2.81

Results of Grid Search for GSGP's Tournament Size

PPB

Run	TS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	2	251.26	109.54	0.02	0.01	-21.76	2.58	2.00	-6.98	4.09
1	5	455.10	180.83	0.02	0.01	-23.25	2.73	3.00	-5.89	3.95
1	10	370.89	112.42	0.03	0.02	-21.37	2.04	1.00	-6.49	4.88
2	2	172.81	162.37	0.00	0.00	-24.21	4.74	1.00	-8.90	2.44
2	5	413.00	194.15	0.00	0.00	-27.71	5.21	3.00	-7.74	7.16
2	10	273.36	159.54	0.01	0.01	-25.16	2.67	2.00	-10.24	5.80
3	2	230.29	62.64	0.02	0.01	-20.12	5.38	1.00	-5.95	1.56
3	5	495.76	412.22	0.01	0.02	-21.51	3.47	2.00	-7.19	4.37
3	10	394.47	145.48	0.01	0.01	-21.55	6.88	3.00	-4.31	2.22
4	2	244.59	43.47	0.03	0.00	-21.28	3.95	1.00	-11.55	4.04
4	5	399.40	117.72	0.02	0.01	-22.25	3.34	3.00	-11.48	4.40
4	10	198.69	44.19	0.02	0.00	-21.67	2.29	2.00	-11.97	5.80
5	2	399.05	106.32	0.01	0.01	-27.56	3.58	2.00	-12.36	9.80
5	5	673.38	253.19	0.01	0.01	-24.21	3.18	1.00	-15.26	6.84
5	10	720.28	179.91	0.00	0.01	-27.60	3.25	3.00	-9.26	7.83
6	2	539.70	193.46	0.02	0.01	-23.85	4.54	3.00	-8.27	1.72
6	5	481.80	123.58	0.02	0.00	-21.44	3.31	1.00	-8.43	0.90
6	10	273.53	107.33	0.02	0.02	-23.38	1.81	2.00	-10.39	4.67
7	2	97.28	52.16	0.00	0.00	-24.69	4.64	1.00	-12.72	5.57
7	5	145.83	69.77	0.01	0.01	-49.57	49.55	3.00	-14.99	2.54
7	10	176.96	129.99	0.00	0.01	-28.14	2.53	2.00	-9.13	6.59
8	2	551.10	350.68	0.02	0.01	-21.07	2.82	2.00	-10.06	5.92
8	5	495.92	179.58	0.02	0.01	-20.64	2.89	1.00	-7.44	4.05
8	10	617.77	413.69	0.02	0.01	-21.65	5.30	3.00	-6.27	4.93
9	2	631.94	363.46	0.01	0.01	-23.42	3.68	1.00	-17.62	3.21
9	5	1444.66	1921.87	0.04	0.05	-26.51	4.71	3.00	-14.71	3.60
9	10	400.91	85.59	0.01	0.00	-24.84	4.95	2.00	-13.39	6.00
10	2	557.88	223.75	0.02	0.01	-22.86	5.55	1.00	-8.62	2.79
10	5	613.39	382.75	0.01	0.01	-24.07	6.04	2.00	-15.32	7.28
10	10	338.06	173.97	0.01	0.01	-25.15	5.62	3.00	-11.81	7.06

Results of Grid Search for GSGP's Tournament Size

TOX

Run	TS	Mean fit time	Std fit time	Mean score time	Std score time	Mean validation score	Std validation score	Rank validation score	Mean train score	Std train score
1	2	575.58	142.73	0.01	0.01	-1631.56	227.92	3.00	-1139.96	260.50
1	5	459.49	147.13	0.01	0.01	-1564.49	209.63	1.00	-917.28	229.59
1	10	439.19	174.94	0.01	0.01	-1610.02	225.91	2.00	-1195.24	303.09
2	2	339.81	149.73	0.01	0.02	-1374.71	173.73	2.00	-1386.23	33.98
2	5	567.84	105.18	0.11	0.17	-1365.08	125.28	1.00	-1344.96	47.72
2	10	402.57	193.42	0.01	0.01	-1399.74	108.00	3.00	-1257.28	197.53
3	2	193.46	106.16	0.01	0.02	-1265.33	149.90	2.00	-1081.97	394.50
3	5	517.06	340.04	0.00	0.00	-1279.69	121.01	3.00	-1254.27	34.94
3	10	540.83	175.93	0.15	0.16	-1227.38	141.50	1.00	-1040.92	153.45
4	2	507.39	154.09	0.02	0.01	-1578.55	193.63	3.00	-1317.55	309.80
4	5	423.03	112.43	0.01	0.01	-1551.88	217.98	2.00	-993.32	286.88
4	10	563.65	211.18	0.02	0.01	-1448.72	269.06	1.00	-992.21	89.94
5	2	681.71	369.05	0.02	0.02	-1573.69	303.44	2.00	-1411.42	290.37
5	5	710.86	540.50	0.01	0.01	-1636.07	389.45	3.00	-1495.03	105.98
5	10	664.48	256.23	0.02	0.03	-1553.11	362.97	1.00	-1497.68	133.51
6	2	553.39	128.47	0.01	0.01	-1393.67	442.53	2.00	-1273.48	240.22
6	5	675.29	352.61	0.02	0.01	-1495.81	278.27	3.00	-1126.04	132.94
6	10	432.72	143.87	0.02	0.01	-1345.91	448.08	1.00	-1365.10	134.38
7	2	280.46	67.09	0.01	0.01	-1616.40	313.48	3.00	-1373.05	179.88
7	5	447.06	131.53	0.06	0.07	-1523.95	321.43	1.00	-1314.51	205.04
7	10	380.70	74.87	0.01	0.01	-1566.31	261.04	2.00	-1315.65	213.20
8	2	454.53	206.42	0.10	0.19	-1447.62	259.57	1.00	-1204.43	259.75
8	5	582.42	197.35	0.01	0.01	-1470.74	327.80	3.00	-1224.91	368.02
8	10	803.29	828.31	0.07	0.13	-1453.71	317.25	2.00	-1349.43	316.57
9	2	715.92	323.36	0.01	0.01	-1514.95	210.90	3.00	-1138.39	235.51
9	5	705.76	180.45	0.02	0.01	-1465.45	246.18	1.00	-1189.93	242.62
9	10	872.25	239.94	0.02	0.01	-1498.57	178.76	2.00	-1108.71	187.82
10	2	757.36	273.04	0.01	0.01	-1398.01	247.59	3.00	-1130.79	173.26
10	5	675.96	371.24	0.00	0.00	-1342.77	217.06	2.00	-1291.39	49.87
10	10	418.93	39.54	0.01	0.01	-1314.59	220.92	1.00	-941.63	54.87

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa