NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

EVOLVING ENSEMBLES WITH TPOT

Camila Andrea Sarmiento Betancourt

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

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

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

EVOLVING ENSEMBLES WITH TPOT

by

Camila Andrea Sarmiento Betancourt

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

Advisor: Leonardo Vanneschi

ABSTRACT

Machine learning has become popular in recent years as a solution to various problems such as fraud detection, weather prediction, improve diagnosis accuracy, and more. One of its goals is to find the model that best explains the problem. Among the several alternatives on how to accomplish that, significant attention has been laid on the matter of accuracy using stacking ensembles: the objective is to produce a more accurate prediction by combining the predictions of various estimators. This model has often been exhibiting a superior performance in contrast to its single counterparts. Because the process of choosing the best model for a given problem can be time-consuming, a necessity to automatize the machine learning process has emerged. Different tools allow this, including TPOT, a Python library that uses genetic programming to optimize the machine learning process, evolving pipelines randomly created until the best one is found, or a previously fixed maximum number of generations for the given problem is reached. Genetic programming is a field of machine learning that uses evolutionary algorithms to generate new computer programs, and it has been shown successful in quite a few applications. TPOT uses several machine learning algorithms from the Sklearn Python library. It also features some ensembles, such as Random Forest or AdaBoost. Currently, stacking ensembles are not implemented yet on TPOT, and, considering its current accuracy rates, the objective of this thesis is to implement stacking ensembles in TPOT. After we implemented stacking ensembles successfully in TPOT, we performed some experiments with different datasets and noticed that for almost all of them, TPOT has comparable performance to TPOT with stacking ensembles. Also, we observed that, when using the light dictionary version of TPOT, the results of the Stacking configuration improved for two datasets since it used weaker learners.

Keywords: Machine learning, Ensembles, Genetic programming, TPOT

INDEX

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Alan Turing is known as the father of computer science. In 1950, Turing published an essay titled "*Computing Machinery and Intelligence*" [1]. He considered the question "*Can machines think?*" and proposed a game called *imitation game*, where the idea is to find if a machine can be labelled as human according to a series of questions asked by an interrogator. In this essay, Turing also analyzed how machines can learn or how they can be taught concluding that the learning process can appropriately be guided by evolution. Turing identified the power of how evolution could be used to solve problems and create intelligent behaviors. Although Turing never developed a machine that could genuinely learn, he was one of the first to believe that it was possible.

The origin of machine learning is attributed to psychologist Frank Rosenblatt, who created a machine, called Perceptron, to recognize the letters of the alphabet inspired by the human nervous system. This machine became the first prototype for Artificial Neural Networks [2]. Later, in 1980 Kunihiko Fukushima proposed a multilayer convolutional Neural Network [3]. In the early XXI century, the idea of machine learning became popular due to big data and the necessity to reduce computational time [4].

As Machine learning progressed, so did the interest to research and improve on different approaches, such as scaling up supervised learning, reinforcement learning, the learning of complex scholastic models and, enhancing the classification accuracy using ensemble methods [5]. Stacking ensemble was introduced by Wolpert in 1992 [6], which aim is to minimize the error rate and maximize accuracy. It uses meta-classifiers to learn how to combine the prediction from one or more base models. In 1996, Breiman presented a paper called bagging predictor [7], a technique to generate multiple versions of a predictor to get a more accurate aggregated predictor built on bootstrap replicates of the learning set. The outputs are combined by plurality vote. In the same year, Freund and Schapire published a new boosting algorithm [8]. This method combines multiple weak learners to create a strong one by learning the mistakes made by the previous learner. In this sense, boosting can considerably reduce the error of any weak learning algorithm. The interest in ensembles increased because this method showed a better performance compared to single classifiers.

In the same way, in the 1950s and 1960s, the interest in evolution as an optimization tool increased [9]. Holland [10] was the first to introduce the concept of genetic algorithms in 1975, and in 1992 Koza introduced the concept of genetic programming [11]. There is a substantial difference between genetic programming and genetic algorithms. Whereas genetic algorithm typically uses a fixed-length string representation of solutions, genetic programming uses a hierarchical representation of computer programs. In nature, the individuals more able to adapt to the different conditions of the environment will have better odds to survive and reproduce over the weaker ones. This is the concept of natural selection introduced by Charles Darwin [12]. Genetic Programming is a machine-learning approach that is inspired by this principle. The algorithm starts creating an initial population with random individuals or programs that tries to solve a given problem. Then, the quality, or fitness, of each individual in the population is assessed. The fittest individuals are more likely to survive and reproduce. Their offspring will pass on to the next generation. After that, the individuals in the new generation

are also evaluated and the selection process is iterated. This process continues until either a previously fixed maximum number of generations is reached, or the algorithm finds a satisfactory solution for the given problem.

Machine learning has achieved excellent results in different areas, such as computational vision, natural language processing, finance, medicine, etc. However, the machine learning process can be labored and time-consuming because it can be difficult to design effective features, choose the appropriate machine learning algorithm and optimize it. On that account, it is necessary to create a tool that allows us to automatize throughout this process. This process is called Auto Machine learning (AutoML). It can help to save time, improve accuracy, make the process more reproducible, and make the process scalable. There are several tools to automate Machine Learning processes, such as Auto-Sklearn [13], PyCaret [14], TPOT [15] and so on.

TPOT (Tree-based pipeline optimization) is a tool that allows us to automatize the machine-learning processes using genetic programming [15]. In TPOT, the individuals are machine learning pipelines composed of selectors, preprocessors, and classifiers. TPOT uses Sklearn [16], a Python package for machine learning, and for the genetic programming process, it uses the Python package DEAP [17].

TPOT has several machine learning algorithms, among which we can find some ensembles, such as Random Forest, AdaBoost, XGBoost, and others. However, stacking ensembles have not been implemented yet in TPOT. Therefore, this project aims to incorporate stacking ensembles into TPOT since they have often shown better performance than other algorithms.

Train stacking ensembles can be an arduous process since it requires to make combinations of single algorithms, and the hyperparameters of all these algorithms should be tuned before combining them. Additionally, it is necessary to find the best algorithm to combine the prediction made for the previous one. AutoML can be handy with staking ensembles since it can automatically select the models to use and determine the best way to combine the predictions.

The expected result of this project is to successfully incorporate stacking ensembles into TPOT, improve the accuracy of the models that TPOT produce, and give the TPOT users an accessible way to implement this estimator. If the initial objectives will be met, TPOT will have more variety in terms of algorithms, have a model that typically outperforms single models and traditional ensembles, and provide a simple way to tune and train stacking ensembles.

In this thesis, we present the results of the study, and this document is organized as follows. The second chapter of this document shall feature a brief introduction to what machine learning is, and we will introduce the concept of ensembles, including their different types. The focus of this chapter will be Stacking ensembles. Also, we will discuss different research made in Auto Machine Learning. The third chapter will focus on genetic programming, where we will explain how the genetic programming process is carried out and we will present how TPOT works. The fourth chapter contains information about the datasets and the parameters used for the experiments as well as the implementation of Stacking ensembles in TPOT. The fifth chapter contains the results and the analysis of the experiments. Finally, the sixth chapter contains the conclusion and discusses the future work to be considered for the project.

2. MACHINE LEARNING

Machine learning is the process to extract patterns from data. This technique learns automatically from a model the relationships between the feature variables and the target variable and then obtains predictions for new instances. In other words, it is possible to build data-driven models that are more accurate and reliable than traditional models.

Machine learning can be defined as:

"Field of study that gives computers the ability to learn without being explicitly programmed."

Arthur Samuel, 1959

Machine learning can be categorized into three groups: supervised learning, where the machine is provided with training data that includes both the input data and the desired output. The machine then can learn and generalize from this data to produce the desired output for new data; unsupervised learning, where the machine is only provided with input data and is not given any desired output. The machine must then learn to recognize patterns and structure in the data to produce useful results; reinforcement learning is where the machine is given a set of rules or objectives and must learn how to achieve them by trial and error. This type of learning is often used in artificial intelligence applications.

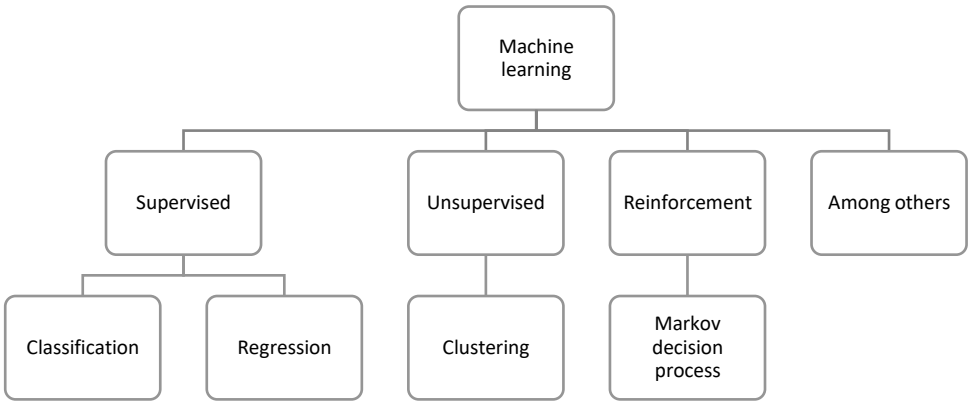

Figure 2.1 Types of Machine Learning

There most common problems in supervised learning are classification and regression. Classification is when the target is a class or label. For instance, classifying an email as spam or not, classifying a document according to a class, image classification, or fraud detection. Regression is when the target is a continuous variable, for example, weather forecasts, predictions of house prices or sales in a company, among others.

The process to build a supervised learning problem is divided into four steps (Figure 2.2). The first one is data preparation. In this step, the cleaning of the data, treatment of nulls, feature transformation, and feature selection all take place. The second step is modelling according to the problem distinct machine learning algorithms can be selected and trained in the dataset, after which the performance of each model is recorded. There are different metrics to measure the model's performance, such as accuracy, F1 score, precision, recall for classification and RMSE, R^2, and MAE, for regression. In the

third step, the best model is evaluated in the test dataset to assess its performance and consistency. Finally, predictions are made in unseen data.

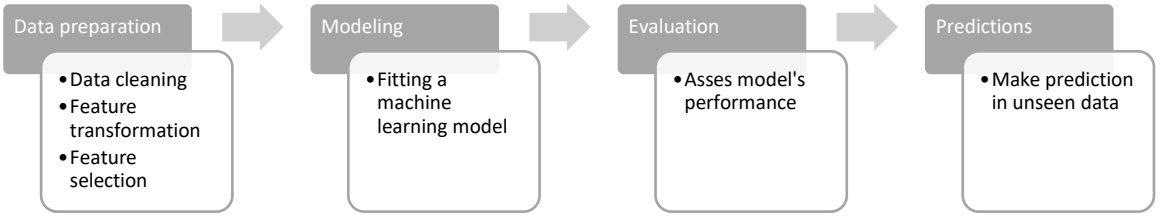

Figure 2.2 Machine Learning process

2.1. ENSEMBLE METHODS

One of the most crucial topics in Machine Learning research has been the improvement of accuracy by learning ensemble methods. The main interest in this area stems from how stacking ensembles outperform individual classifiers [5].

Ensemble models are meta-algorithms that combine other machine learning models into a single predictive one. The goal of ensemble methods is to construct a collection of classifiers that are both diverse and accurate. Two classifiers are diverse if the errors made by them are uncorrelated [18].

There are different ensemble methods, such as bagging, boosting, and stacking.

In bagging, ensembles are made of estimators built on a bootstrap (a resampling method that uses random sampling with replacement) replicates of the training. Outputs are combined by majority vote. The training process is in parallel [7]. One example of bagging is random forest: this is an ensemble of decision trees, and the vote of each decision tree has the same weight.

In boosting, a weak algorithm can be boosted into a strong one. The training cannot be done in parallel; instead, it is made as an iterative process, where the new model is influenced by the performance of the previous one. In this sense, the new model focuses on the mistakes of the previous one. The predictions are combined using a weighted majority vote [8]. For instance, AdaBoost creates a stump (a tree with one node and two leaves), and each stump considers the previous stump's mistakes.

2.1.1. Stacking Ensemble

In 1992 Wolpert [6] introduced the concept of stacking generalization. The main objective was to maximize accuracy and minimize error. Wolpert said that stacked generalization can be seen as a more sophisticated version of cross-validation since it involves training a second machine-learning algorithm on the guesses of the original algorithm. Moreover, stacked generalization can be used with a single generalizer, in this case, it corrects the errors made by that generalizer.

This technique consists of a combination of multiple machine learning algorithms, known as level-0 (base) models, and the data used for training these models is the level-0 data. Then, the output of each model in level-0 is combined into a new dataset (level-1 data), and a learning algorithm is used in the

new dataset. This learning algorithm was called by Wolpert level-1 generalizer but is also called a combiner or meta-classifier [19].

Ting and Witten [19] exposed that as a meta-classifier is preferable to use a multi-response linear regression algorithm over a decision tree, Naïve Bayes classifies, K-nearest neighbors, and majority vote. Furthermore, linear regression can be easily interpreted which gives it more advantage over the other learning algorithms. Also, they found that it is better to use output class probabilities than class predictions.

Stacked generalization is prone to overfitting, particularly when the level-0 models are highly correlated unless the meta-classifier is smooth. As Ting and Witten showed to avoid this issue is appropriate to use multi-response linear regression. However, Reid and Grudic [20] demonstrated that overfitting can be avoided through regularization using Ridge regression, lasso regression or elastic net regression. These techniques result in a sparse linear model, which means that only a small number of classifier posterior predictions are used in stacking generalization. This can lead to improving accuracy since it reduces the chance of overfitting.

In summary, the combiner should be a simple model, interpretable and use regularization. Nevertheless, there is no one best combiner for stacked generalization; the best combiner will depend on the data and the specific classification task.

Contrary to bagging, in stacking the estimators are different and they are trained on the same data set. Opposed to boosting, stacking uses a single model to combine the predictions from the models in level-0, it is not a sequence of estimators that learn from the previous mistakes.

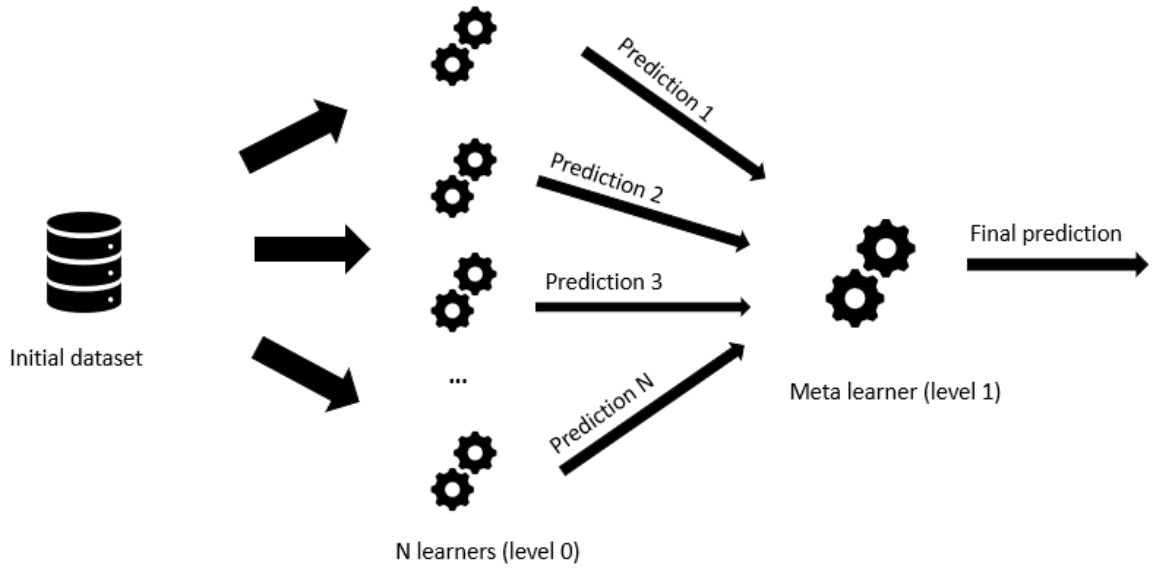

Figure 2.3 Stacking Ensemble

Stacking ensembles have been applied to different fields showing excellent performance. For instance, earthquake-casualty prediction. The stacking ensemble used for this prediction was composed using different combinations of base learners such as CART, Naïve Bayes, k-nearest neighbor, and random rorest. The results showed an outstanding performance of stacking ensembles over popular machine

learning methods, and it was possible to reduce the MSE by 8.10% compared to a gradient boosting decision tree [21].

Moreover, stacking ensembles were used for early diagnosis of Parkinson's disease. For this purpose, a two-layer stacking ensemble was created. The first layer was composed of 4 classifiers: support vector machine, random forest, K-nearest neighbor, and artificial neural network. The second layer was a logistic regression. The results showed that the stacking ensemble model had a better performance than single traditional models [22].

Another application is to predict short-term electric energy demands. In this study, regression trees based on Evolutionary Algorithms, Artificial Neural Networks and Random Forests were selected as base learners, and Generalized Boosted Regression was the combiner. The results showed that compared to linear regression and decision tree, stacking ensembles always had better results [23].

2.2. Auto Machine Learning (AutoML)

The machine learning process can be difficult and time-consuming because of the tunning of the algorithms, the appropriate selection and transformation of the features, and the election of the most suitable model for a given problem [24]. In Figure 2.4. we can see that choosing an appropriate machine learning algorithm involves several decisions.

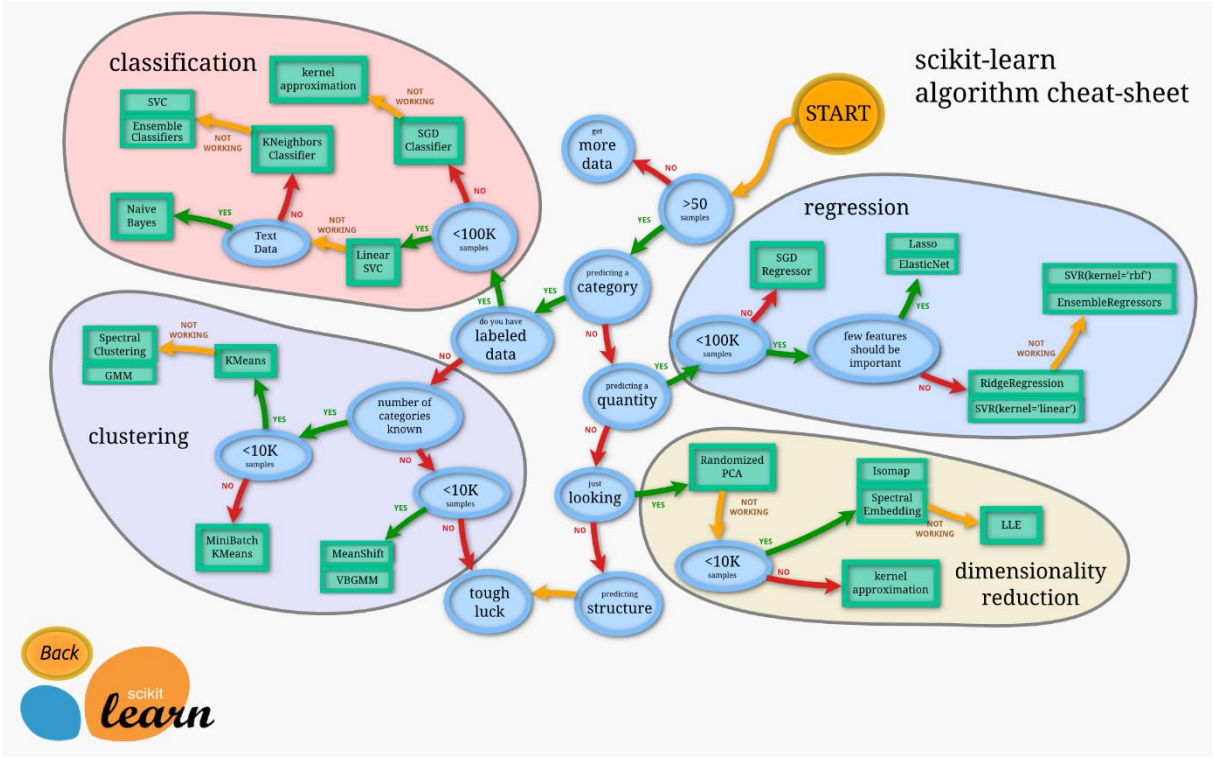

Figure 2.4 Algorithms in scikit-learn [25]

AutoML aims to facilitate the machine-learning process. Then, AutoML automated features transformation, model selection and hyperparameters tuning to find a high-performing machine learning model [26]. AutoML can provide a solution for individuals who are not machine learning experts to find the best algorithm to make predictions.

AutoML has been used to solve different problems. Such as making predictions about diseases, including COVID-19, where it is crucial to track the number of cases, and it can help researchers to develop vaccines. A study was made using the H2O AutoML tool to analyse cases of COVID-19 in India. It was possible to find a suitable algorithm to predict diseases in a reasonable time [27]. AutoML can be essential to accelerate research in machine learning since making predictions about the spread of the disease can help decision-makers plan for and respond to an outbreak.

Another research made in AutoML focuses on automated neural architecture search. Negrinho and Gordon [28] propose a framework to automatically design and train deep neural network models. This framework has three main components: the model search space specification language, the model search algorithm, and the model evaluation algorithm. They use search algorithms such as random search, Monte Carlo tree search (MCTS), and sequential model-based optimization (SMBO). The results showed that MCTS and SMBO outperformance random search.

There are different methods to automate machine learning processes: Bayesian processes allow finding an appropriate model for a given dataset [29]. Gaussian processes can be used to automate statistical modelling and exploratory data analysis. This method has a flaw which is its running time. Thus, this method is not scalable [30]. Similarly, it is possible to use genetic programming to automate machine learning processes.

Moreover, different tools exist for AutoML, such as PyCaret, used to produce machine-learning models with the least possible amount of coding involved. It is practical for classification, regression, and clustering problems [14]. H2O AutoML is a tool made for non-machine learning experts, offering model expandability, and allowing for machine learning models building on big data [31]. TPOT is a Python library design to automate machine learning processes using genetic programming [15], which shall be discussed in the next section.

3. GENETIC PROGRAMMING

Genetic programming (GP) is a field of evolutionary computation that uses evolutionary algorithms inspired by the Theory of Evolution of Charles Darwin [12]. The scope is to create high-quality programs that can solve problems without human intervention. GP is widely used in several fields, such as optimizing financial portfolios, routing telephone calls and scheduling airline flights.

The interest in evolution as an optimization tool started in the 1950s and 1960s to solve engineering problems [9]. In 1962, Holland [32] proposed looking for an adaptive system as a *population of programs*. He highlighted the advantage of population programs to gain generalization instead of taking a single individual. After, in 1975 Holland [10] published a theoretical framework, in which he included the concepts of selection, crossover and mutation inspired by genetics. In 1985, Cramer [33], introduced an adaptive system for generating short sequential computer functions, one of which had a tree-like structure.

Afterwards, the concept of Genetic Programming was published in Koza's book in 1992 [11]. Koza rigorously aboard the notion of genetic programming and how to represent GP problems. Furthermore, Koza provides examples in his book of how problems that had been difficult or even impossible to solve using traditional methods were suddenly easy to solve using genetic algorithms.

3.1. GENETIC PROGRAMMING PROCESS

The genetic programming process is as follows: The first step consists of randomly generating an initial population of computer programs (individuals). Then, each individual gets assessed according to their fitness. After that, the best individuals are selected according to their fitness. Therefore, these individuals are modified by genetic operators, such as mutation, crossover, or reproduction. Each of these operators is selected with a probability specified as a parameter. Then, the modified individuals are included in the new population. This process is iterated until a termination criterion is met, it can be an optimal solution, or the number of generations is exceeded. The steps of GP algorithms are shown in Algorithm 1 [34].

Genetic programming algorithm

1. Randomly generate an initial population of computer programs (Individuals) from the primitives set.
2. While a termination criterion is met (found and acceptable solution or other stop condition is met).
  a. Evaluate the fitness for each individual.
  b. Probabilistic selection of a set of individuals based on their fitness.
  c. Create new individuals by applying genetic operations (crossover, mutation, reproduction) to the selected individuals and insert them into a new population.

Return: The best computer program

Algorithm 1. Genetic programming

3.1.1. GP Representation

The most common representation is the syntaxis tree (tree-based GP). In the tree, the nodes are called primitives and the leaves are terminals. Exists two kinds of terminals: constants, which remain the

same during the process of evolution, and arguments which are the program inputs. For instance, in the program *min(x - 2*y, x+ y),* the constant is *2* and the arguments are the variables *x* and *y*. In Figure 3.1. we can see the tree representation of this program where the nodes are in yellow and the leaves in green.

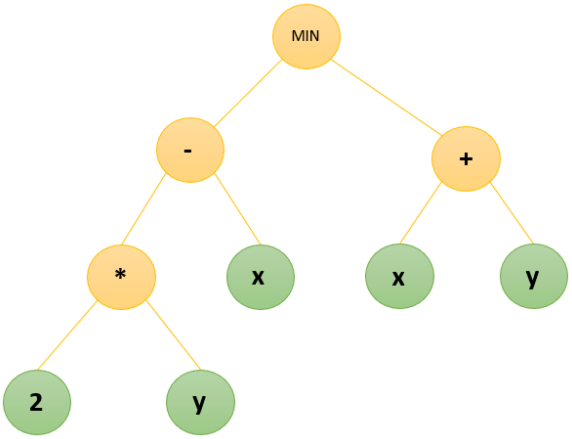

Figure 3.1 Tree based representation

3.1.2. Initialization

At the outset, population individuals are chosen randomly. There are different methods to generate the initial random population, the grow method, the full method, and ramped half-and-half method. For the full and the grow methods, the user needs to specify the maximum depth.

Full method

In this method, all the leaves need to have the same depth. The nodes are randomly chosen from the primitive set. In the last level the terminals are selected. As a result, every branch of the tree reaches the maximum depth. Figure 3.2 shows an example of this process. In this method, not all initial trees will have the same size, which means not all trees will have the same number of nodes.

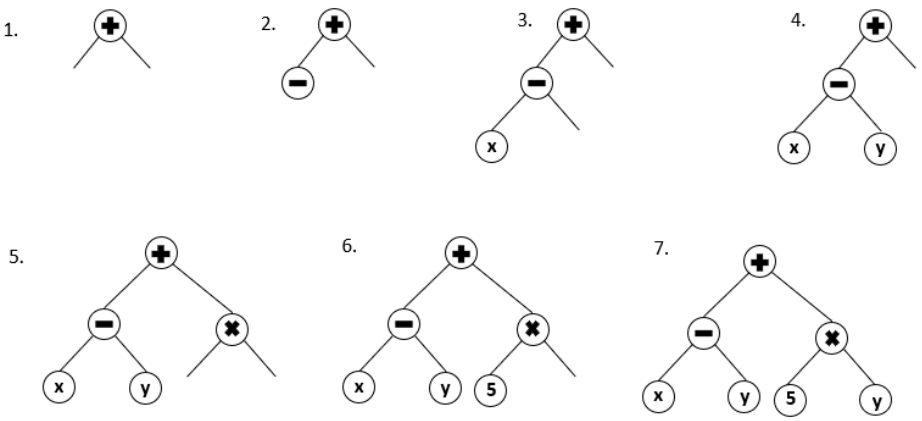

Figure 3.2 Full method with maximum depth 2

Grow method

Opposite to the full method, in the grow method the initial trees can have different shapes and sizes. In this approach, nodes can be selected from the primitive set and from the terminals until the maximum depth is reached; in the last level, only terminals can be added to the tree.

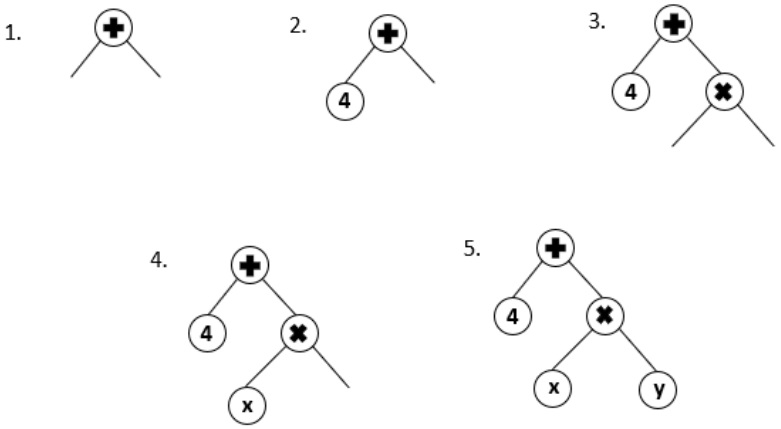

Figure 3.3 Grow method with maximum depth 2

Ramped half-and-half method

According to Koza, the techniques mentioned above may generate trees that are similar to each other [11]. So, the ramped half-and-half method was developed with the purpose to add diversity to the population. In this method, half of the initial population is generated with the full method and the other half uses the grow method.

3.1.3. Selection

A selection algorithm has the objective to choose one individual from the population. A selection algorithm has to be a process with repetition, and it has to be probabilistic. Also, all the individuals in the population must have the possibility to be selected. Individuals are chosen according to their fitness value, which is the ability each individual has to solve a problem. Thus, better individuals are more likely to have offspring than inferior ones. The most common selection algorithms are tournament selection, fitness proportion selection, and ranking selection.

Tournament selection

In tournament selection, there is a number *n* of randomly selected individuals, with uniform distribution from the population. Then, the selected individuals are compared with each other and the one with the best fitness is selected. *n* refers to the tournament size; this parameter establishes the selection pressure. As the selection pressure increases, superior individuals have a higher probability to survive or being selected. This algorithm can be more efficient than others since it is not necessary to evaluate the fitness function of all individuals in the population.

Fitness proportional selection

This technique is also known as Roulette Wheel. In this method, every individual has a probability of being selected proportional to their fitness. Then, superior individuals have a higher probability to be chosen. Despite this method being easy to implement, posing the possibility of superior individuals being selected every time as parents, leading to a loss of diversity within the system.

Ranking selection

In this algorithm, individuals are sorted according to their fitness value, usually from worst to best. The probability to select an individual is directly proportional to the position in the ranking. One of the advantages of this algorithm is that it is not sensitive to differences in fitness in contrast to the fitness proportional selection algorithm.

Multi-objective optimization

In some cases, it is necessary to optimize more than one criterion at the same time. Known as multi-objective optimization, and the most common one is Pareto optimality. For instance, in a classification problem, we want to select the model with the highest accuracy and the smallest number of parameters; in other words, we want to maximize the accuracy and minimize the number of parameters in the model.

In a multi-objective optimization problem, the performance of a solution is determined by dominance. A solution dominates another one if the former one is not worse than the latter in all objectives and there is at least one strictly better objective. The goal of this algorithm is to find a set of non-dominated solutions. This set is called Pareto optimal set, and the boundary that separates the non-dominated solution from the dominated one is the Pareto front.

NSGA-II (Non-dominated Sorting Genetic Algorithm) is an elitist multi-objective genetic scheme. This algorithm has three main characteristics: it uses elitism; it also uses a diversity-preserving mechanism, that is the crowding distance (the average distance of its two neighbourhood solutions), and it emphasizes non-dominated solutions.

3.1.4. Genetic operators

After the best individuals are selected as the parents for the next generations, genetic operators are applied to them to create offspring solutions. There are two approaches for creating a new population: reproduction, which consists of copying the individual unmodified into the new population; and modification, which consists of applying a genetic operator (crossover or mutation) to the individual with a specific probability. Then the modified individuals are introduced to the new population. Moreover, there are cases when it is desired to preserve the best individual to insert into the new population, this is called Elitism.

Crossover

To add diversity to the population, the crossover operator produces a new child taking parts of each parent and adding said offspring into the new population. Some crossover operators tend to preserve the position of the genetic material, known as homologous crossover. The oldest homologous crossover in tree-based GP is one point crossover. In this operator, a common crossover point of the

parents is selected and then the corresponding subtrees are switched to obtain two children [35]. In Figure 3.4 we can see an example of a one-point crossover when the parents have different shapes. In these cases, one-point crossover selected the intersection from the parts of the trees in the common region, where both parents have the same shape.

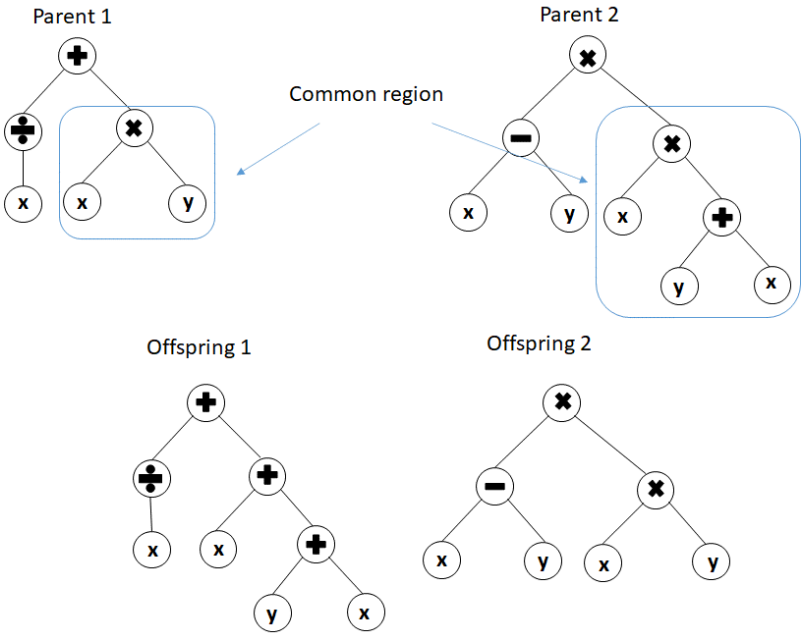

Figure 3.4 One-point crossover

Mutation

Mutation operators modify just one parent solution. The most common mutation operators are point mutation, a node in the tree is randomly selected and swapped with another random node with the same arity. Shrink Mutation selects a random subtree and replaces it with a randomly created terminal; insertion mutation, on its part, adds a new subtree into the individual in a random position.

3.2. TPOT

Genetic programming can be applied to automate the machine learning process, this is the case of TPOT [15]. In this section, we are going to explain how the TPOT process works.

TPOT is a tool to automate machine learning process. In the official document [15] is define as:

> *"TPOT stands for Tree-based Pipeline Optimization Tool. Consider TPOT your Data Science Assistant. TPOT is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming"*

TPOT uses the Python library scikit-learn to feature selection, transformation, and machine learning, as well as for genetic programming uses the Python package DEAP, to find the pipeline with the best performance for a given dataset.

In TPOT, the set of primitives (internal tree nodes) consists of machine learning operators, three to be exact: Pre-processor, Feature Selectors, and Models. The pre-processor operator modifies the data somehow and returns the modified data. Feature selection operators reduce the number of variables in the dataset by considering some criteria and then returns the modified data. Model operators store the predictions as a new feature and the classification of the pipeline. Additionally, TPOT has another operator that combines multiple copies of the dataset into a single dataset.

To combine the primitives into a valid machine learning pipeline, TPOT uses tree-based pipelines and the GP algorithms to evolve them.

3.2.1. TPOT process

3.2.1.1. Initialization

To initialize the process, TPOT chooses the operators in the configuration dictionary and starts to add them to the set of primitives. Then, it generates randomly grown trees with different depths, between one and three, where the root of each tree is a model operator (classifier or regressor). Then, each expression is processed and evaluated to ensure that they are valid ML pipelines (individuals) and to guarantee that the individuals are unique. This prevents TPOT from getting stuck on a bad pipeline and allows it to evaluate the score of each individual, which subsequently is added to the initial population until it has *n* tree-based pipelines, where *n* is the number of individuals within the population.

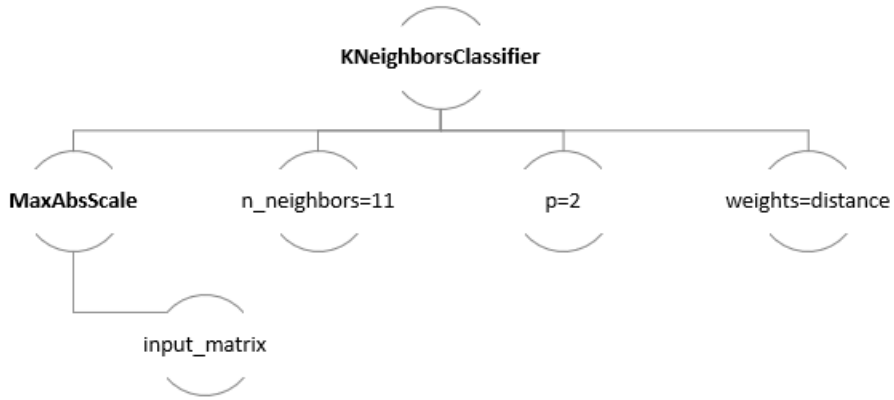

Figure 3.5 TPOT individual for a classification problem

Figure 3.5 shows an individual created by TPOT. The root is a KNeighborsClassifier (a model operator); the node is MaxAbsScaler (a pre-processor operator), and the terminals are the arguments taken by these two operators. The pipeline for this individual is:

Pipeline(steps = [MaxAbsScaler(), KNeighborsClassifier(n_neighbors=11, p=2, weights=distance)])

3.2.1.2. Selection

Each pipeline is evaluated according to its performance on the dataset. TPOT uses score metrics such as accuracy or negative MSE by default for classification or regression, respectively. TPOT uses multi-objective fitness to minimize the number of operators and maximize the score.

Then, for every generation, TPOT selects *n* individuals according to the NSGA-II algorithm, where the pipelines are selected to minimize the number of operators in the pipeline and maximize the score. In every generation, the algorithm updates a Pareto Front of the non-dominated solutions.

3.2.1.3. Variation

For each generation, *n* offspring are produced. 10% of whom are crossovers with another individual using the one-point crossover operator. Two individuals are eligible for mating if they, at least, share one primitive. Only one of the offspring is added to the new population. If the population do not have individuals eligible for crossover, the mutation operator is applied to maintain diversity in the population. For instance, in Figure 3.6 we can see two pipelines that have as a common primitive a MultinomialNB (Naive Bayes classifier for multinomial models) applying one-point crossover, two offspring individuals are created, via exchanging the value of the alpha parameter.

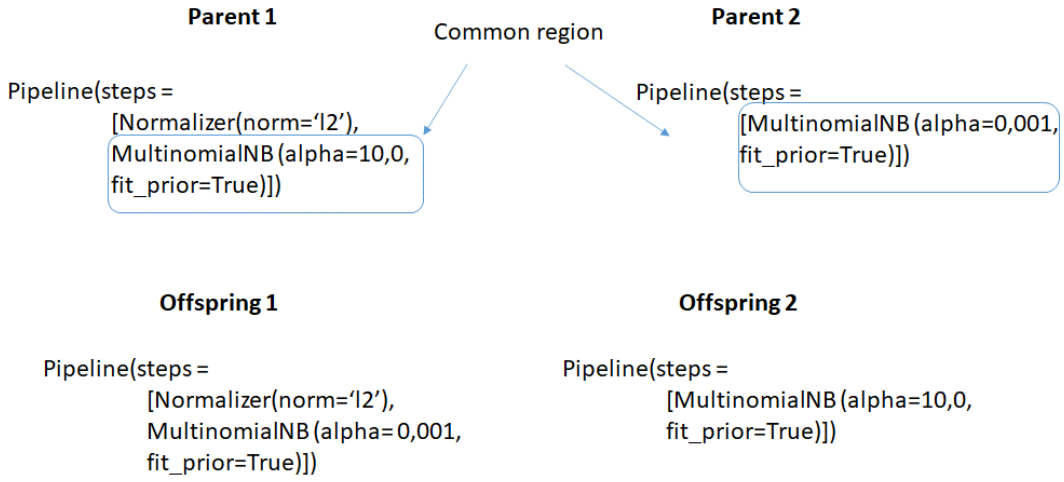

Figure 3.6 One-point crossover in TPOT

The remaining 90% of individuals are modified by applying one of the three types of mutation operators in TPOT: Point mutation, which randomly replaces a chosen primitive from the individual with another random primitive from the set of primitives; insertion mutation consists of inserts a new operator at a random position in the individual; or shrink mutation, which shrinks an individual by randomly choosing an operator and replacing it with one of the operator's arguments. It is possible to apply shrink mutation if the individual has more than one primitive. Each of these techniques has the same probability to be chosen (1/3). In Figure 3.7 we can see how TPOT applies these three types of mutation to different pipelines.

Finally, if the individual is chosen for neither mutation nor crossover, TPOT applies reproduction. The individual is cloned and added to the offspring population. Every time a modified individual is created, TPOT checks if it is still a valid pipeline.

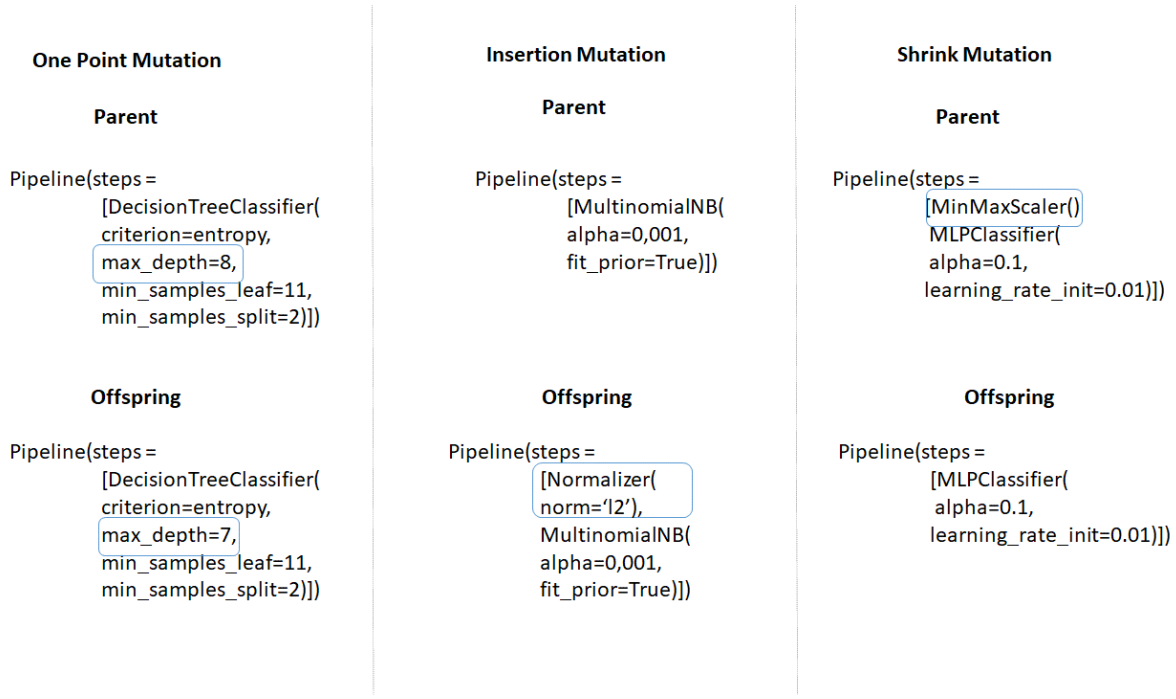

Figure 3.7 Mutation in TPOT

The TPOT process continues until a prefixed number of generations is reached. The result is the best pipeline for a given dataset.

TPOT can be used in different problems, such as classifying liver cancer on multiphasic MRI. In this research, two experiments were accomplished. One used manual analysis to select the optimal machine learning model and the other one used TPOT. The results showed that the performance of the two methods (manual and TPOT optimizations) was similar [36].

4. METHODOLOGY

4.1. DATASETS

The first step in this project was the implementation of stacking ensembles into TPOT. To validate the results, we used several datasets to test the performance of TPOT with Stacking Ensembles for classification and regression problems and compare them against TPOT.

The datasets used are the Fish market dataset, in which the objective is to predict the weight of a fish [37]; the Titanic dataset [38], in which the goal is to predict whether a passenger survived or not. Two datasets that have experimented overfitting (PPB and Bioavailability); House prices [39] to predict the real estate costs in Iowa; Newland [40], where the goal is to predict the income of the passengers (that is a binary rate); and the Weather dataset [41] to predict climate conditions (such as drizzle, rain, sun, snow, and fog).

The cleaning process of the datasets consists of deleting null variables when applicable or imputing null observations with the mode, transforming categorical variables into numerical variables using One Hot Encoder and removing features such as ID, ticket number, and birthday, according to the dataset. The train size for each dataset was 0.75.

The number of observations and features are presented in table 4.1.

Dataset	Type of the task	No. Of Features	No. Of Observations
Titanic	Classification	9	891
Newland	Classification	14	22,400
Weather	Classification	5	1,461
Fish	Regression	6	159
House prices	Regression	80	1,460
Bioavailability	Regression	241	359
PPB	Regression	626	130

Table 4.1 Datasets used and their number of Observations and features

4.2. PARAMETERS

We decided to use the TPOT default parameters since they have shown excellent results [15] except for the population size and the number of generations. We used a population size of 20 since the running time was longer with a larger population size, and we selected 80 generations because the results remained steady after the 80th generation. Table 4.2 lists the parameters used.

Parameter	Value
Population size	20
Generations	80
Offspring size	Population size
Mutation rate	0.9
Crossover rate	0.1
Scoring	Accuracy for classification and negative MSE for
CV	5
Config_dict	Default and Light version
Verbosity	2

Table 4.2 Parameters and their values

TPOT has predefined some dictionaries with different configurations to customize operators and parameters that TPOT selects during optimization. In this project, we work with the default and light settings. In table 4.3, we can see the different dictionaries with their operators and values.

Dictionary	Operators	Values
Default configuration for classification (classifier_config_dict)	Classifiers	GaussianNB, BernoulliNB, MultinomialNB, DecisionTreeClassifier, ExtraTreesClassifier, RandomForestClassifier, GradientBoostingClassifier, KNeighborsClassifier, LinearSVC, LogisticRegression, XGBClassifier, SGDClassifier, MLPClassifier
Default configuration for regression (regressor_config_dict)	Regressors	ElasticNetCV, ExtraTreesRegressor, GradientBoostingRegressor, AdaBoostRegressor, DecisionTreeRegressor, KNeighborsRegressor, LassoLarsCV, LinearSVR, RandomForestRegressor, RidgeCV, SGDRegressor
Default configuration for classification and regression	Preprocessors	Binarizer, FastICA, FeatureAgglomeration, MaxAbsScaler, MinMaxScaler, Normalizer, Nystroem, PCA, PolynomialFeatures, RBFSampler, RobustScaler StandardScaler, ZeroCount, OneHotEncoder
Default configuration for classification and regression	Selectors	SelectFwe, SelectPercentile, VarianceThreshold, RFE (just for classifiers), SelectFromModel
Light configuration for classification (classifier_config_dict_light)	Classifiers	GaussianNB, BernoulliNB, MultinomialNB, DecisionTreeClassifier, KNeighborsClassifier, LogisticRegression

Light configuration for regression (regressor_config_dict_light)	Regressors	ElasticNetCV, DecisionTreeRegressor, KNeighborsRegressor, LassoLarsCV, LinearSVR, RidgeCV
Light configuration for classification and regression	Preprocessors	Binarizer, FeatureAgglomeration, MaxAbsScaler, MinMaxScaler, Normalizer, Nystroem, PCA, RBFSampler, RobustScaler, StandardScaler, ZeroCount
Light configuration for classification and regression	Selectors	SelectFwe, SelectPercentile, VarianceThreshold

Table 4.3 Different configurations dictionaries with their operators and values

4.3. IMPLEMENTATION

The objective of this project is to implement heterogeneous ensembles into TPOT using stacking operators from the Sklearn library. To do that, we created a new configuration dictionary using either the default settings or the light version (as it can also be chosen).

The models in each Stacking ensemble are chosen randomly from the given dictionary (default or light), and the number of models in each stacking randomly varies between one and three. In the first experiment, logistic and linear regression were chosen as combiners. Eventually, we decided to add more combiners such as decision trees and linear classifiers with stochastic gradient descent (SGD) learning for classification, as well as dummy regressors (using the median), linear models with SGD, Linear Support Vector regressor, and Ridge CV for regression. Combiners are chosen randomly for each stacking ensemble. We decided to implement linear models with SGD (SGDClassifiers) since it applies regularized linear model with SGD to build an estimator, SGD works well in large-scale datasets and is easy to implement [42].

For instance, an individual in TPOT with ensembles is as follows:

```
Pipeline(steps =
        [FeatureAgglomeration(affinity="euclidean", linkage="ward"),
        StackingClassifier(
                estimators=[
                ("GaussianNB", GaussianNB()),
                 ("LogisticRegression", LogisticRegression(
                        C=0.5, dual=False,
                        penalty="l2")),
                ("KNeighborsClassifier", KNeighborsClassifier(
                        n_neighbors=26,
                        p=2,
                        weights="uniform"))],
                final_estimator= LogisticRegression()
        ])
```

In this case, the root is a stacking classifier composed of three estimators and as a final estimator, it uses Logistic Regression. Also, it has another primitive that is a pre-processor called Feature Agglomeration.

This process continues until we have the same number of stacking models in the new dictionary as the number of models in the initial configuration dictionary. For instance, if there are thirteen classifiers in the default configuration, then it will be thirteen stacking ensembles constructed with those classifiers, in the new dictionary. The preprocessors and selectors are the same as in the initial configuration dictionary. Once we have the new dictionary, the TPOT optimization process starts as usual.

In the variation process, the same algorithms for crossover and mutation remain with the same crossover and mutation probabilities. For instance, the process of crossover is shown in Figure 4.1.

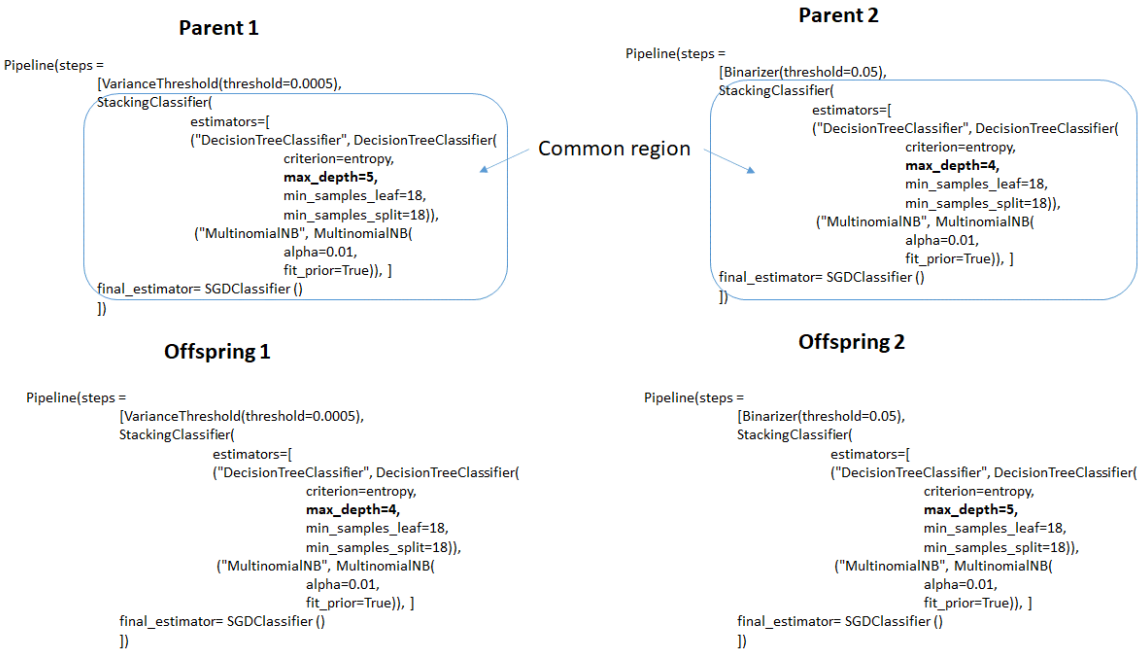

Figure 4.1 One-point crossover using stacking ensembles

Similarly, we can see in Figure 4.2 how mutation is applied in TPOT with stacking ensembles.

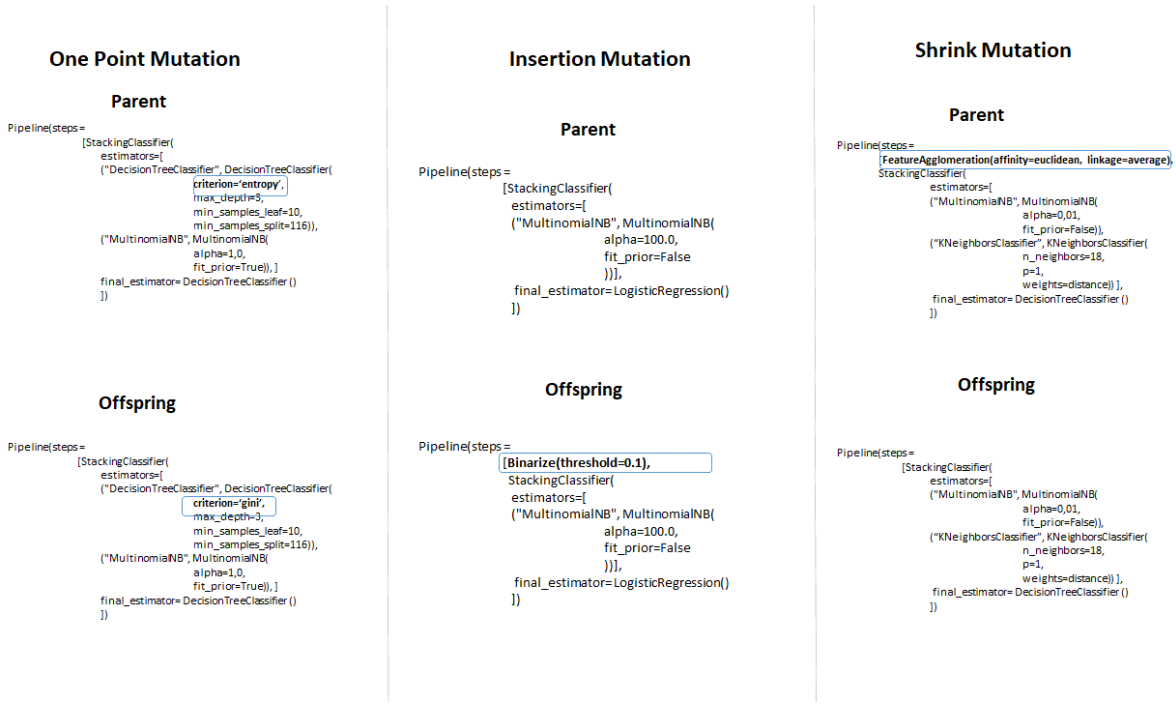

Figure 4.2 Mutation using stacking ensembles

5. RESULTS AND DISCUSSION

5.1. IMPLEMENTATION OF ENSEMBLES

To validate the implementation of stacking ensembles in TPOT, different runs were tested over the same datasets, using the same parameters, and comparing the results obtained with the standard version of TPOT and TPOT with stacking ensembles. For each generation, we obtained the CV score for the best individual. Then, the experiment is repeated for 10 iterations. To compare the results for each configuration, we calculated the Median Best Fitness (MBF), the median of the best score obtained at a given generation. To evaluate the statistical difference of the results obtained, we used the Wilcoxon rank-sum test at 5% on the last performed generation. The results obtained will be discussed in this section. The graphs were made using the Plotly library for Python 3. In the graphics, the label False indicates that the default TPOT configuration was used, and the label True signifies that TPOT with stacking ensembles was used.

5.1.1. Comparison of results

First, we used the default configuration dictionary in TPOT to create stacking ensembles using the pre-defined models in this dictionary. In the first approach, Linear and Logistic Regressions were used as combiners, for regression and classification, respectively.

Figure 5.1 shows the MBF for each dataset in the train and test sets. In the House Pricing and Fish datasets, we can see that the default configuration of TPOT is slightly better than using stacking ensembles. In the other datasets, however, there does not seem to be a clear difference between the two configurations.

We can also see in Figure 5.1 that, for PPB and Bioavailability datasets, the Stacking configuration presents more overfitting, as we mentioned before these two datasets have experimented overfitting and when we add more learners to the base or level 0 in Stacking models, we obtain a more complex model and then it is more likely to be overfitted.

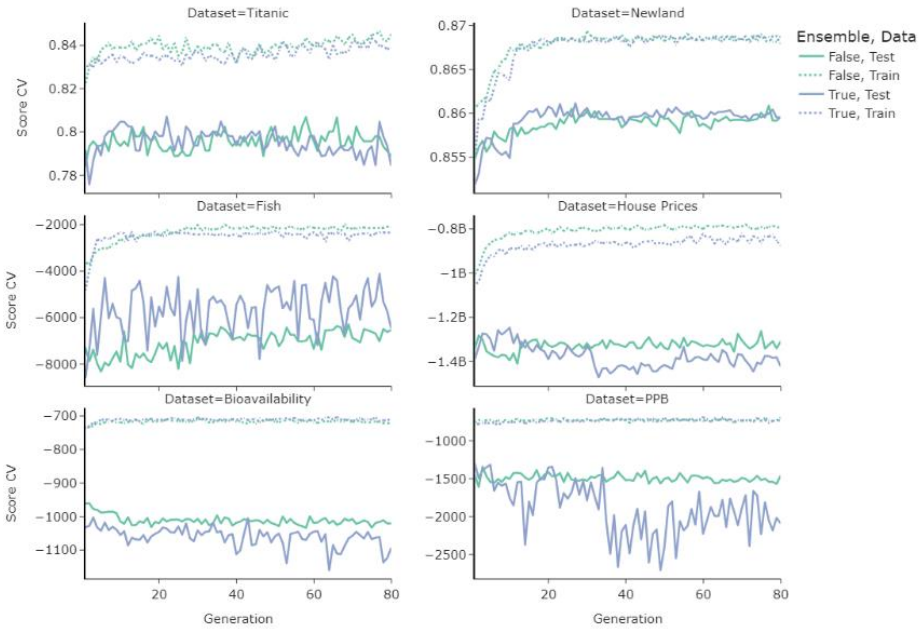

Figure 5.1 MBF for each configuration in the first approach

Dataset	Wilcoxon rank-sum test in train	Wilcoxon rank-sum test in test
Titanic	0.3447	0.9698
Newland	0.1736	0.3846
Fish	0.0393	0.8282
House prices	0.0494	0.0343
Bioavailability	0.3284	0.0756
PPB	0.8065	0.2885

Table 5.1 P-value for the Wilcoxon test for the train and test set in the first experiment

Table 5.1 depicts the results of the Wilcoxon test for the train and test sets for the two configurations, “True” and “False”, of the previous figure. As we can see, the p-value for almost the entirety of the dataset is not statistically significant at 5%, except for the House Prices dataset. That is to say, the performance of TPOT with stacking is comparable to standard TPOT.

Moreover, we measure the time each configuration takes to run (TPOT standard, TPOT with stacking ensembles) in each generation and for each iteration. We calculated the median of the running time obtained in each generation. We found out that running the code using Stacking takes twice as long or more than using the Standard configuration of TPOT. We can see the results in Figure 5.2. These results are expected since training stacking ensembles require, also training the base models of which it is composed.

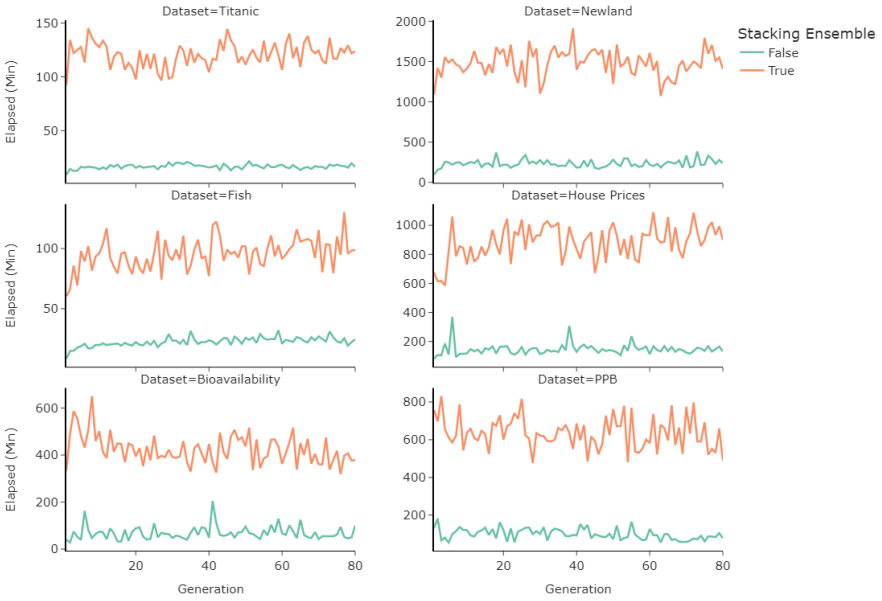

Figure 5.2 Running time in the first experiment

Consequently, we decided to add more combiners (level-1) as decision trees and linear classifiers with stochastic gradient descent (SGD) learning for classification, as well as dummy regressors (using the median), linear models with SGD, Linear Support Vector regressors, and Ridge CV for regression. This implementation was run over the Titanic and PPB datasets. We observe for the Titanic dataset the best combiner was Logistic Regression, and for the PPB dataset, the best combiners were Linear Regression and Ridge CV.

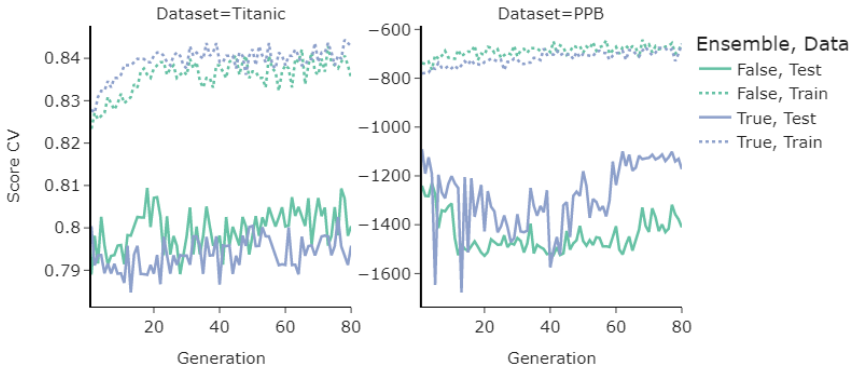

Figure 5.3 MBF of each configuration using different combiners for the stacking model

Dataset	Wilcoxon rank-sum test in train	Wilcoxon rank-sum test in test
Titanic	0.2123	0.2122
PPB	0.7054	0.6501

Table 5.2 P-values of the Wilcoxon test in the test and train set for the configuration using different combiners for the stacking model

In Figure 5.3, we can see that the results obtained do not show a clear difference between the two configurations. Similarly, in Table 5.2 the Wilcoxon test does not present a statistical difference between the two configurations presented in Figure 5.3.

In the same vein, we can see via Figure 5.4. that the running time using TPOT with Stacking is slower than the standard TPOT configuration. As we mentioned before, train stacking ensembles is time-consuming, then these results are expected.

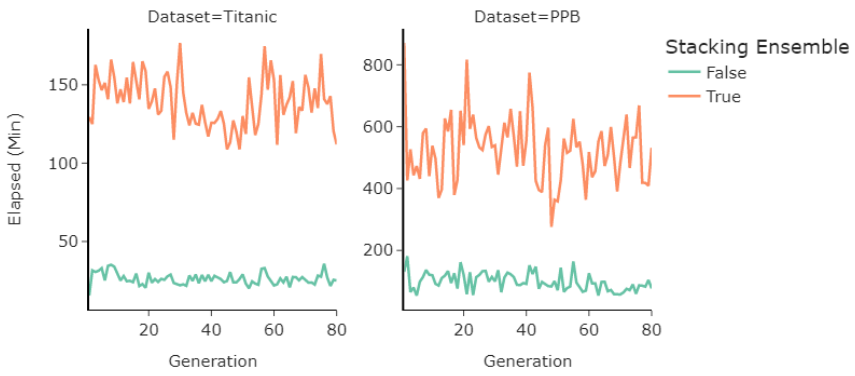

Figure 5.4 Running time using different combiners for the stacking model

As the results were not improving, we decided to delete the ensemble models (XGB, extra trees, random forest, and gradient boosting) from the initial default configuration dictionary to obtain stacking ensembles with simpler models, and we opted to keep different combiners. We ran the code using Titanic, Newland, Fish and House Pricing datasets. Furthermore, we included the Weather dataset since stacking ensembles have performed excellently for forecasting problems. We decided not to use PPB and Bioavailability since the running time was excessively high for these two datasets.

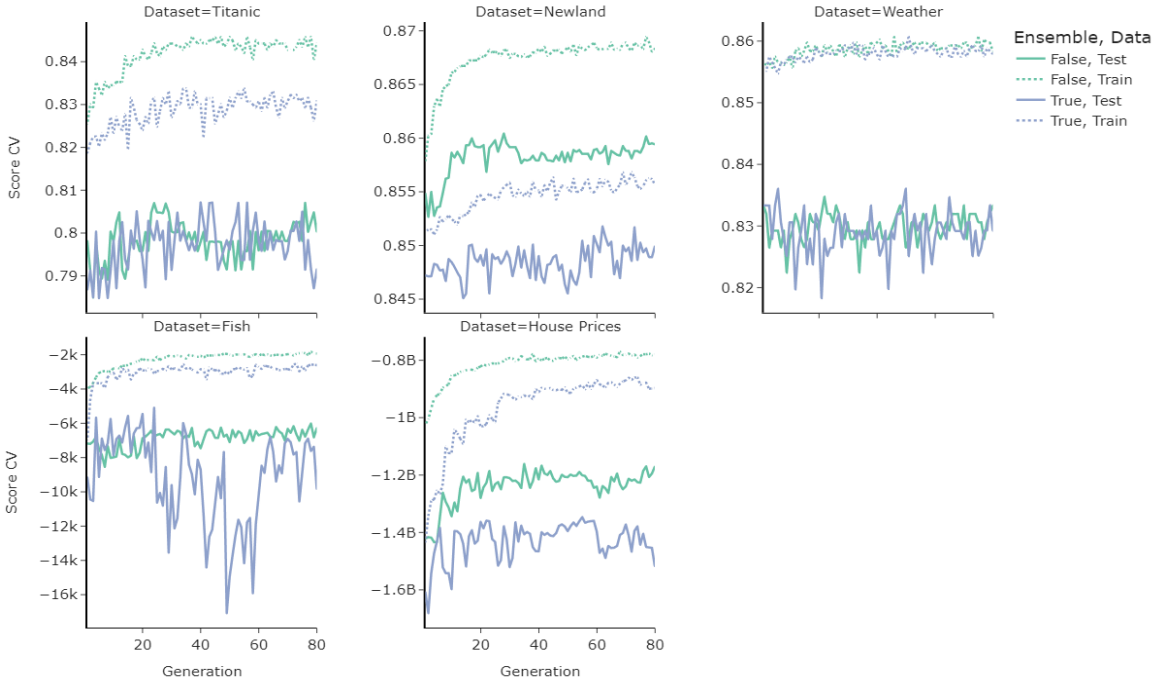

Figure 5.5 MBF of each configuration removing the ensembles models from the default configuration dictionary in TPOT

Dataset	Wilcoxon rank-sum test in train	Wilcoxon rank-sum test in test
Titanic	0.0025	0.6776
Newland	0.0002	0.0004
Weather	0.7913	0.0539
Fish	0.0082	0.1509
House prices	0.0025	0.0009

Table 5.3 P-values of the Wilcoxon test in the test and train set for the configuration removing the ensembles models from the default configuration dictionary in TPOT

Figure 5.5 shows that the best configuration is the default TPOT for all datasets except for the Weather dataset, for which it does not reflect a clear difference between the two configurations. We can confirm these results in Table 5.3, which shows that exists a statistical difference between the two configurations except for the test in the Titanic dataset, the Fish dataset, and the test and train in the Weather dataset. The best combiners for the Classification problems were logistic regression for the Titanic and Newland datasets, and SGDClassifier for the Weather dataset. Linear regression was the best combiner for regression problems.

In Figure 5.6, we can see that the running time is similar for the Newland dataset, and for the other databases it took with the stacking configuration.

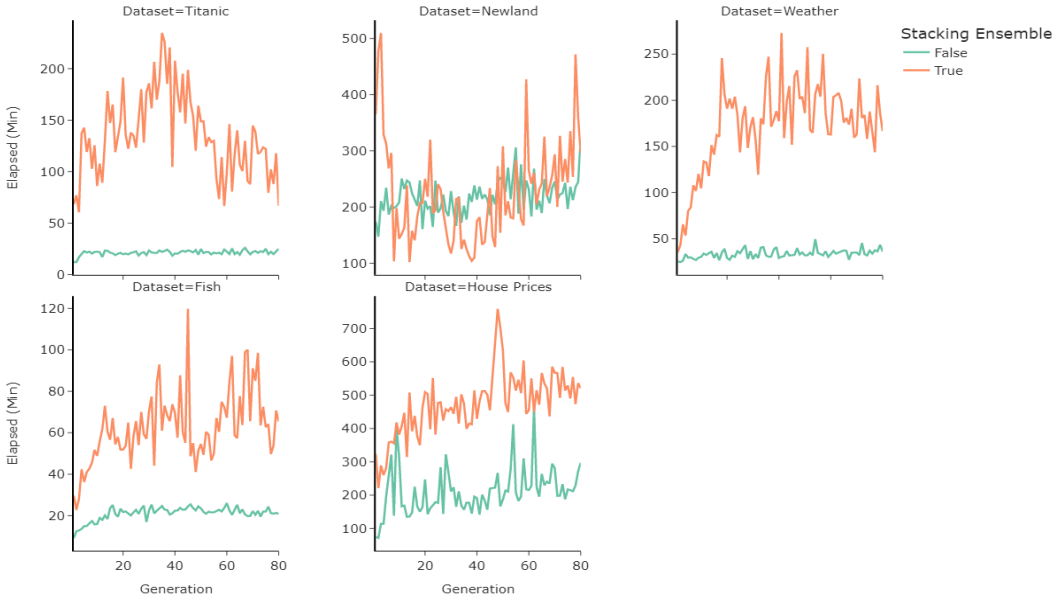

Figure 5.6 Running time of each configuration removing the ensembles models from the default configuration dictionary in TPOT

As we found a difference between the two configurations, we decided to repeat the experiment using the light dictionary configuration in TPOT, considering it has weaker learners. The databases used for this experiment were Titanic, Fish, House Pricing, and Weather. We decided to include Bioavailability again to see if the results improve using weaker learners.

In Figure 5.7, we can see that the configuration using stacking ensembles presents higher performance than the default configuration of TPOT for Newland and Weather datasets. In contrast, for the House Price dataset, it looks like the best configuration is the default one. On the other hand, for Titanic, Fish and Bioavailability datasets there does not seem to be a clear difference between the two configurations. We can confirm these results with the Wilcoxon test (Table 5.4), as it shows that there is a statistical difference between the two configurations for the Newland dataset, but for the House Price dataset, there is only a difference in the training dataset. In the same way, there is a statistical difference between the two configurations for the train set in the Weather database. However, for the Fish, Titanic, and Bioavailability datasets, there is no statistical difference between the two configurations in the test and train datasets. For this experiment, we observe that the best combiner for the classification problem was Logistic Regression, except for the Weather dataset for which the best combiner was SGDClassifier. For the regression problems, the best combiner was Linear Regression.

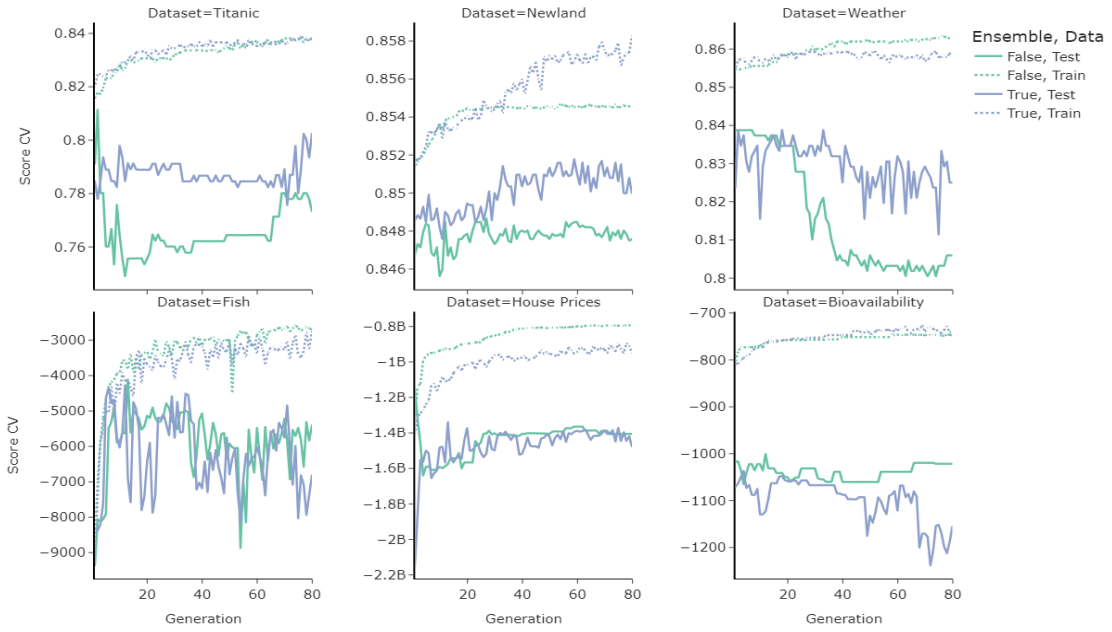

Figure 5.7 MBF of each configuration using the light dictionary

Dataset	Wilcoxon rank-sum test in train	Wilcoxon rank-sum test in test
Titanic	0.8206	0.2123
Newland	0.0233	0.0140
Weather	0.0051	0.1858
Fish	0.8205	0.1988
House prices	0.0065	0.1306
Bioavailability	0.7054	0.1124

Table 5.4 P-values of the Wilcoxon test in the test and train set for the configuration using the light dictionary

Additionally, in Figure 5.8, we can observe that the running time of the stacking configuration using the light dictionary is smaller than in the others experiments. However, the running time of the stacking configurations is longer than the running time of the default TPOT configuration.

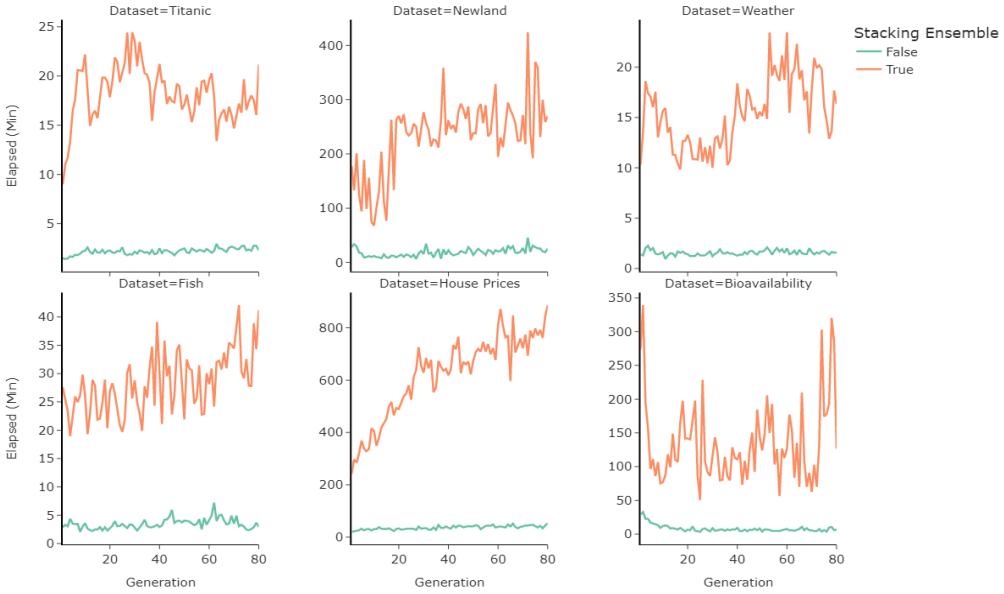

Figure 5.8 MBF of the time with the configuration using the light dictionary

To summarize, the experimental study was a success since stacking ensembles were satisfactorily implemented into TPOT. Moreover, we could obtain results for different datasets. In the beginning, the Stacking configuration was not statistically different from the default TPOT configuration; but by removing ensembles from the TPOT dictionaries or using weaker learners, we achieved a difference between the two configurations. Furthermore, in most cases, the TPOT configuration outperformed the Stacking Ensemble configuration.

6. CONCLUSIONS AND FUTURE WORKS

The objective of this chapter is to present the conclusions about the project, discuss our approaches and the problems faced during its completion, and some possible improvements for the future.

The main purpose of this project was to implement heterogeneous ensembles in TPOT, specifically stacking ensembles since they can improve predictive accuracy, reduce the error, and perform better than single algorithms. To implement stacking ensembles in TPOT, we created a new dictionary using the pre-existing TPOT algorithms in the default and light version dictionaries. The stacking ensembles are shaped by 1 or 3 algorithms that are chosen randomly from the TPOT's dictionaries. Analogously, the meta-classifier is chosen randomly from a list of simple learners that are well known for giving good results as combiners, such as linear regression, Ridge regression, decision trees, and others. Once, the stacking ensembles are made the TPOT process starts as usual.

During the experimental study, we compared the results obtained by TPOT using stacking ensembles and standard TPOT on eight different datasets. In the first stage, we used the default configuration dictionary and linear or logistic regression as combiners. Our results demonstrated that there is no statistical difference between the two configurations. In other words, TPOT with stacking has a comparable performance with standard TPOT. In the second examination, we added more algorithms to the combiner list to assess if the results improve. The results conclude that there is no improvement just adding more combiners. In view of the result not improving, we decided to remove the traditional ensembles (random forest, XGBoost, AdaBoost, and others) from the default configuration dictionary, to obtain stacking ensembles with weaker learners. From these results, we could conclude that exists a statical difference between the two configurations in some of the datasets. However, better results were obtained using standard TPOT for almost all the datasets. In the final stage, we used the light configuration dictionary to obtain stacking ensembles with simpler algorithms. The results showed a statistical difference between the two configurations in a few datasets. Slightly superior results were achieved with our configurations for two datasets.

The present study confirmed the findings about the most suitable algorithms to use as combiners, demonstrating that linear regression and logistic regression were usually selected for the evolution process as the best combiners. A further finding is that linear models with stochastic gradient descent are also adequate combiners. There is no research showing the advantages of these algorithms as combiners in stacking ensembles.

The results demonstrate that better results were achieved using weaker algorithms as base models. Also, the running time using TPOT with Stacking ensembles is longer than standard TPOT, as we expected. In this sense, it is recommendable to use the light configuration dictionary as the first approach to TPOT with stacking ensembles.

Contrary to the findings of previous studies showing that stacking ensembles outperform another machine learning algorithm we did not find a general improvement using this technique. An explanation can be that we used simple databases which do not have complex patterns to find, such as Titanic and Fish. However, the performance of stacking ensembles depends on the database, and

we cannot conclude that they are not better than single machine learning algorithms. Ideally, these findings should be replicated in a study where we can use databases with more complex patterns.

The major limitation of this study was the computational resource since stacking ensembles can be computationally intensive, and when TPOT evolves them, the process becomes time-consuming, taking twice as long as standard TPOT, which can make them impractical for some applications.

Despite the limitations, these are valuable in light of TPOT now includes another machine learning algorithm that has been showing robustness in several applications and can provide TPOT users with another option to find the best solution for their machine learning problem.

Future research should consider the potential effects of using TPOT with stacking ensembles with more complex databases. We believe that besides performing, future research should look for combining machine learning pipelines using stacking ensembles. Moreover, we would like to implement the optimization of the hyperparameters of machine learning models using simulated annealing with TPOT using stacking ensembles to optimize the hyperparameters used by them.

7. BIBLIOGRAPHY

[1] A. Turing, "Computing machinery and intelligence," *Mind,* pp. 433-460, 1950.

[2] F. Rosenblatt, *The Perceptron, A Perceiving and Recognizing Automaton,* Project Para Report No. 85-460-1, Cornell Aeronautical Laboratory (CAL), 1957.

[3] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics,* vol. 36, no. 4, pp. 193-202, 1980.

[4] A. L. Fradkov, "Early History of Machine Learning," *IFAC-PapersOnLine,* vol. 53, no. 2, pp. 1385-1390, 2020.

[5] T. G. Dietterich, "Machine-Learning Research Four Current Directions," *AI Magazine,* vol. 18, no. 4, pp. 97-136, 1997.

[6] D. H. Wolper, "Stacked Generalization," *Neural Networks,* vol. 5, pp. 241-259, 1992.

[7] L. Breiman, "Bagging Predictors," *Machine Learning,* vol. 24, pp. 123-140, 1996.

[8] Y. Freund and R. Schapire, "Experiments with a New Boosting Algorithm," *Machine Learning: Proceedings of the Thirteenth International Conference,* pp. 148-146, 1996.

[9] M. Melanie, An Introduction to Genetic Algorithms, London, England: MIT Press, 199.

[10] J. Holland, Adaptation in Natural and Artificial Systems, Ann Arbor, MI: MIT, 1975.

[11] J. Koza, Genetic Programming, Cambridge: MIT Press, 1992.

[12] C. Darwin, On The Origin of Species by Means of Natural Selection, or Preservation of Favoured Races in the Struggle for Life, London: John Murray, 1859.

[13] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum and F. Hutter, "Efficient and Robust Automated Machine Learning," in *Advances in Neural Information Processing Systems 28 (2015)*, 2015, pp. 2962--2970.

[14] M. Ali, "PyCaret: An open source, low-code machine learning library in Python," April 2020. [Online]. Available: {https://www.pycaret.org.

[15] R. Olson and J. Moore, "TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning," in *Automated Machine Learning*, Springer, 2019, pp. 151-160.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[17] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research,* vol. 13, pp. 2171-2175, 2012.

[18] T. Dietterich, "n Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization," *Machine Learning,* vol. 40, pp. 139-157, 2000.

[19] K. Ting and I. Witten, Stacked generalization: when does it work?, Hamilton, New Zealand: University of Waikato, Department of Computer Science, 1997.

[20] G. Sam and R. Greg, "Regularized Linear Models in Stacked Generalization," in *Multiple Classifier Systems*, Berlin, Heidelberg, Springer, 2009, pp. 112-121.

[21] S. Cui, Y. Yin, D. Wang, Z. Li and Y. Wang, "A stacking-based ensemble learning method for earthquake casualty prediction," *Applied Soft Computing,* vol. 101, 2021.

[22] Y. Yang, L. Wei, Y. Hu, Y. Wu, L. Hu and S. Nie, "Classification of Parkinson's disease based on multi-modal features and stacking ensemble learning,," *Journal of Neuroscience Methods,* vol. 350, 2021.

[23] F. Divina, A. Gilson, F. Goméz, M. García and J. Torres, "Stacking Ensemble Learning for Short-Term Electricity Consumption Forecasting," *Energies,* vol. 11, 2018.

[24] D. Radečić, Machine Learning Automation with TPOT, Birmingham: Packt, 2021.

[25] "Scikit learn," [Online]. Available: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.

[26] S. Das, Cakmak and U, Hands-On Automated Machine Learning: A beginner's guide to building automated machine learning systems using AutoML and Python, Packt, 2018.

[27] S. Gomathi, R. Kohli, M. Soni, G. Dhiman and R. Nair, "Pattern analysis: predicting COVID-19 pandemic in India using AutoML," *World Journal of Engineering,* vol. 19, pp. 21-28, 2020.

[28] R. Negrinho and G. Gordon, "DeepArchitect: Automatically Designing and Training Deep Architectures," arXiv:1704.08792, 2017.

[29] G. Malkomes, C. Schaff and R. Garnett, "Bayesian optimization for automated model selection," in *ICML 2016 AutoML Workshop*, New York, 2016.

[30] H. Kim and Y. Teh, "Scalable Structure Discovery in Regression using Gaussian Processes," in *ICML 2016 AutoML Workshop*, New York, 2016.

[31] E. LeDell and S. Poirier, "H2O AutoML: Scalable Automatic Machine Learning," July 2020. [Online]. Available: https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf.

[32] J. Holland, "Outline for a Logical Theory of Adaptive Systems," *Journal of the ACM,* vol. 9, pp. 297-314, 1962.

[33] N. Cramer, "A representation for the adaptive generation of simple sequential programs," *First International Conference on Genetic Algorithms,* pp. 183-187, 1985.

[34] R. Poli, W. Langdon and N. McPhee, A field guide to genetic programming, http://lulu.com and freely available at http://www.gp-field-guide.org.uk., 2008.

[35] R. Poli and W. Langdon, "Schema Theory for Genetic Programming with One-Point Crossover and Point Mutation," *Evolutionary Computation,* vol. 6, no. 3, pp. 231-252, 1998.

[36] R. Hu, H. Li, H. Horng, N. Thomasian, Z. Jiao, C. Zhu, B. Zou and H. Ba, "Automated machine learning for differentiation of hepatocellular carcinoma from intrahepatic cholangiocarcinoma on multiphasic MRI," *Scientific Reports,* vol. 12, 2022.

[37] A. Pyae, "Fish market," 2019. [Online]. Available: https://www.kaggle.com/datasets/aungpyaeap/fish-market.

[38] "Titanic," [Online]. Available: https://github.com/EpistasisLab/tpot/blob/master/tutorials/data/titanic_train.csv.

[39] D. D. Cock, "House Prices - Advanced Regression Techniques," Ames Housing dataset, [Online]. Available: https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques.

[40] "Newland," 2020. [Online]. Available: https://www.kaggle.com/competitions/newland.

[41] R. ANANTH, "WEATHER PREDICTION," 2022. [Online]. Available: https://www.kaggle.com/datasets/ananthr1/weather-prediction?resource=download.

[42] "sklearn SGDClassifier," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.