NOVA IMS Information Management School

MDSAA

Master Degree Program in
Data Science and Advanced Analytics

CO-EVOLVING HYPER-PARAMETERS OF TPOT WITH SIMULATED ANNEALING

Laura Marcela Ramos Salamanca

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

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

CO-EVOLVING HYPER-PARAMETERS OF TPOT WITH SIMULATED ANNEALING

by

Laura Marcela Ramos Salamanca

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

Co Supervisor*:* Leonardo Vanneschi

November 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.

Laura Marcela Ramos Salamanca

Lisbon, November 2022

ABSTRACT

Machine Learning (ML) is a field of artificial intelligence that allows learning from past data to predict the future. Numerous ML algorithms can be used to solve the same type of problem, and each algorithm has different possible configurations of hyperparameters (parameters that control the learning process of an ML algorithm). This creates the need for Auto Machine Learning (AutoML). AutoML, in addition to producing powerful results, also eliminates the manual and tedious tasks of fine tuning the hyperparameters and system architectural of individual ML models. One of the most popular AutoML libraries that exist for the Python programming language is Tree-Based Pipeline Optimization Tool (TPOT). This is based on genetic programming (GP), built on the idea that solutions to problems can be found by evolving them from a group of potential solutions. Other optimization techniques for finding the best hyperparameters include Simulated Annealing (SA). SA is a method used to find the global maximum of a function by starting with an initial guess and then continually making minor changes to the guess until the global maximum is found. There are proven cases with satisfactory results where GP and SA are combined to select the best programs. The purpose of this study is to demonstrate that by combining GP and SA you can optimize the execution time of TPOT.

TPOT has some pre-established ML models with multiple hyperparameters that will execute during its processing. At the same time, each model has a list of hyper-parameters that, depending on the complexity of the model, can have more than 7 thousand combinations. This makes it challenging to select the best one since every possible combination would have to be executed. When TPOT uses GP, it randomly selects the model hyper-parameters. This study will pre-select a global optimal hyperparameter using the SA.

Finally, after selecting optimal ML models and hyperparameters with SA, the results were integrated with TPOT in different experiments, reducing the execution time by 66% in the best scenario.

KEYWORDS

Machine Learning (ML); Auto Machine Learning (AutoML); Genetic Programing (GP); Simulated Annealing (SA); Tree-based Pipeline Optimization Tool (TPOT), Hyper-parameters.

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ML	Machine Learning
AutoML	Auto Machine Learning
GP	Genetic Programing
SA	Simulated Annealing
TPOT	Tree-Based Pipeline Optimization Tool
GA	Genetic Algorithm
KNN	K-Nearest Neighbors
QAP	Quadratic Assignment Problem
CHD	Congenital Heart Defects

1. Introduction

Information is one of the most valuable resources in the world since it can be used to solve problems, understand the world, and connect people. Exploring how to obtain, analyze and use data can support a wide variety of different fields of study, including medicine (Ravi et al., 2017), finance (Dixon et al., 2020), economics (Mosavi et al., 2020), etc. As an example, when a bank wants to analyze the satisfaction of its customers based on its services, it can develop a model based on costs, benefits, and rates, making it possible to identify which factors influence customer's purchasing decisions made in relation to banking services (Guerra & Castelli, 2021). Analysis like this have led data scientists, statisticians, systems engineers, and others to develop tools, programs, and algorithms to improve the way data is organized, filtered, and selected. How information is evaluated and used can be employed to optimize decision making in many fields by making it more strategic and assertive.

AutoML is a technological program developed to automatically learn from data and to optimize data processing and analysis. (Fradkov, 2020). However, one of the ML's challenges is the difficulty to choose the right modelling parameters. (Hutter et al., 2019). Thanks to AutoML's progress, libraries such as TPOT have been developed, which is based on the Python programming language. TPOT, with the help of GP, can build tree-based pipelines from different ML models and select the appropriate hyper-parameters. This library allows optimizing both regression and classification problems. However, there are disadvantages, such as long execution time and the stability of the results caused by having random factors. This research aims to analyze TPOT for classification models adding SA when defining the optimal parameters. SA is an optimization technique to find the result of a problem, comparing different individuals and choosing a global optimal solution.

This research will integrate both methodologies as follows. TPOT has some pre-established list containing all the ML models it will execute during its process. This includes models such as Decision Tree Classifier, K-Neighbors Classifier, Logistic Regression, etc. At the same time, each model has lists of hyper-parameters that, depending on the complexity of the models, can have more than 7 thousand combinations. This makes it challenging to select the optimal set of hyper-parameters since every possible combination would have to be executed. TPOT using GP, randomly selects the hyper-parameters of these models. That will allow finding a model and its respective parameters that have acceptable results compared to all the executions that were calculated. This study preselected optimal hyper-parameters with the SA technique to further improve this process. This technique compares a certain number of hyper-parameters for each model and selects the best one compared to the previous one or worse than the previous one, with a certain probability (Lin et al., 2008). Adoption of these techniques allow for quickly finding the optimal solution of hyperparameters for each ML model. The best individuals are selected after executing a certain number of iterations, which will in turn yield better results.

Finally, in the conclusions of this document, the advantages and disadvantages of this research will be outlined based on the methodology briefly explained in the previous paragraph. These are related to time and performance because of the experiments that were applied with the combination of TPOT with SA.

2. Literature review

In today's world, the study and analysis of data are significant because information is a resource that has allowed social and human development. That is why the best programs, tools, and algorithms are currently being developed and implemented; to exploit the data, to solve incalculable problems in daily life. (Cuzzocrea et al., 2011)

For this reason, ML has managed to be a valuable tool in data analysis because its classification algorithms allow us to predict scenarios whose results may be applicable in different sectors, such as economics and medicine. Since "In a nutshell, classification algorithms help us in various scenarios, such as predicting customer attrition, whether a tumor is malignant or not, whether someone has a given disease, and so on. You get the point." (Olson & Moore, 2019)

Due to its functionality, application, and use, how could ML be defined: "Machine learning is a field of computer science that aims to teach computers how to learn and act without being explicitly programmed" (Olson & Moore, 2019). Therefore, investigating how to improve and optimize it is essential for its use and possible benefits. Since ML has proven to be very useful in everyday life. For example, it has managed to suggest to people which books to read, which movies to watch (Furtado & Singh, 2020), and which places to visit (Petrozziello & Jordanov, 2017), and even what kind of people to spend time with (Sun et al., 2015). However, no algorithm is perfect in all possible scenarios. That is why human beings work and research ways to develop ML further.

ML evolves along with the algorithms used in it, as is the specific case of GP. "Evolutionary algorithms are used to find solutions to problems that we humans don't know how to solve directly" (Olson & Moore, 2019). In addition to solving impossible problems for human beings, GP improves the performance of ML: "In machine learning, GP can be used to discover the relationship between features in a dataset (regression), and to group data into categories (classification)" (Olson & Moore, 2019). The GP is inspired by Darwin's evolutionary theory, referring to the algorithmic use of random mutations, crosses, aptitude functions, etc., to solve regression and classification problems (Olson & Moore, 2019).

Predicting data requires a process based on programs and algorithms in order to transform and obtain the best results. Using a library like TPOT, which in this case works with ML and is based on GP procedures, to improve its performance. "It uses the well-known scikit-learn machine learning library to perform data preparation, transformation, and machine learning. It also uses GP procedures to discover the best-performing pipeline for a given dataset." (Olson & Moore, 2019). According to the previous information, the TPOT library adapts to ML. Its design based on GP allows specific processes to be automatic, facilitating more time to collect and clean data.

By improving hyper-parameter optimization, it results in improving the forecast results. SA is a metaheuristic search algorithm aiming to find the most optimal value in a search space. (Koza, 1992) Why is all the information mentioned above so important? To show if AutoML with SA can optimize parameters. Analyzing how SA works with PG allows us to observe the benefits or results that they have when working together Since we intend to use them in this investigation. According to Wong (2011), "Many combinatorial optimization problems are too difficult to be solved optimally, and hence heuristics are used to obtain "good" solutions in "reasonable" time. A heuristic that has been successfully applied to various problems is simulated annealing." In this case, he claims that SA works to get good results. However, he realizes that researchers use and apply much manual searching in their algorithms.

For this reason, he proposes utilizing GP to replace this step. (Wong et al., 2011). He applied GP to robots so they would not have to start the process from scratch. He decided to apply GP and SA because: "we propose to do GP with SA, which is abbreviated as GP/SA. The primary goal is to generate a

program closer to the global optimum in the solution space of the problem." (Wong et al., 2011). Using GP in conjunction with SA will allow the development of better models.

Additionally, in the case of Thonemann (1994) in which he applied an SA algorithm and a GP algorithm in his investigation shows how his program was optimized: “Our algorithm uses a simulated annealing algorithm that solves the quadratic assignment problem (QAP) and a genetic programming algorithm that optimizes the annealing schedule. Each generation consists of 500 annealing schedules that are evaluated based on the objective function of the simulated annealing algorithm”. Their progress is positive because they produce new generations which will allow, based on the GP theory, to generate better models.

What are the advantages of using ML? Remembering that it is based on GP processes and uses the Python TPOT library. "Automated machine learning (AutoML) frameworks have become important tools in the data scientist's arsenal, as they dramatically reduce the manual work devoted to the construction of ML pipelines" (Lazebnik et al., 2022). ML is a great tool that has reduced the manual work of data scientists, saving time. Another advantage is its design, which was built with engineering for the use of models and the adjustment of hyperparameters. "Containing feature engineering, model selection and hyperparameters tuning steps - and finally output an optimal pipeline in terms of predictive accuracy." (Lazebnik et al., 2022). Improving its predictive precision capacity allows it to generate more efficient models. To give an example of how TPOT reduces its execution time, "Our experimental results, performed on two popular AutoML frameworks, Auto-Sklearn and TPOT, show that SubStrat reduces their running times by 79% (on average), with less than 2% average loss in the accuracy of the resultant ML pipeline." (Lazebnik et al., 2022)

3. Methodology

The importance of ML has increased greatly over the recent years. There is a wide variety of tools with which you can solve a classification problem (Classification is a statistical technique to classify the data into categories) and regression (Regression is a statistical technique used to model relationships between variables) (Kelleher et al., 2015) (Bishop, 2013). Two of the biggest challenges when solving a use case are which model should be selected and which hyperparameters these models should have. (Li et al., 2016) Other tools mixed with ML models are the selectors, which are based on selecting the best variables from a data set, and the transformers, which allow variables to be transformed to enhance the results (Le et al., 2018; Olson & Moore, 2019). If all these practices are added and combined, a considerable number of tasks are programmed to have the best possible result for a problem.

The graphic below shows which path a data scientist might take when solving an ML problem and then how solutions start to be found with some examples.

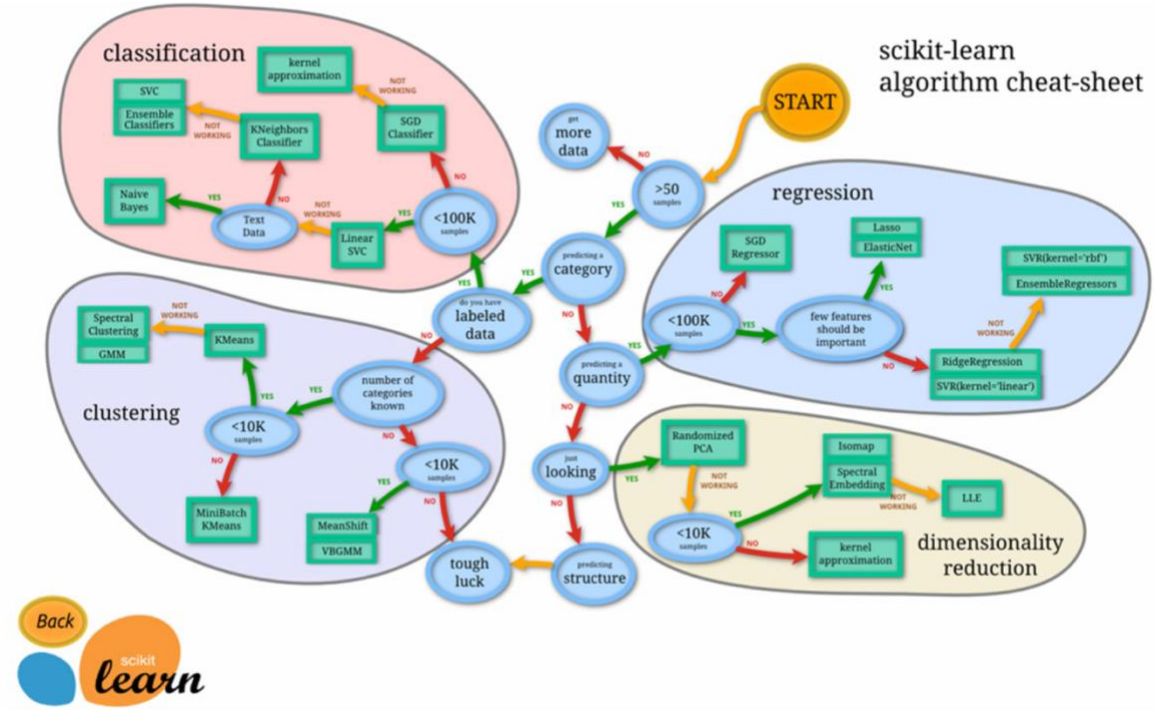


Figure 3-1: Algorithms in scikit-learn (Olson & Moore, 2019; Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. et al., 2011)

AutoML answers these two questions by allowing the user to compare the results of different models without it being a manual and tedious task. (Vaccaro et al., 2021). However, AutoML still has drawbacks, such as, in some cases, advising distinct (pipelines) with the same sample of data. Furthermore, the execution time of the models can take whole days. (Olson & Moore, 2019).
This thesis proposes how to help solve these problems if the hyper-parameters are pre-selected to the ML models using SA. SA has other advantages, like finding optimal solutions and fast execution (Wong et al., 2011). Before describing how SA was included in AutoML, it is crucial to give some definitions.

3.1. Genetic programing (GP)

It is an evolutionary programming method that is inspired by natural selection. Moreover, works with general hierarchical computer programs. (Vanneschi , Wong et al., 2011)
GP works with trees that will be considered individuals, transforming throughout the iterations since "GP is inspired by biological evolution and its mechanisms. It uses algorithms based on random mutation, crossover, fitness functions, and generations." (Olson & Moore, 2019). In the following test, some of these terms will be defined by applying them to the examples of a tree.

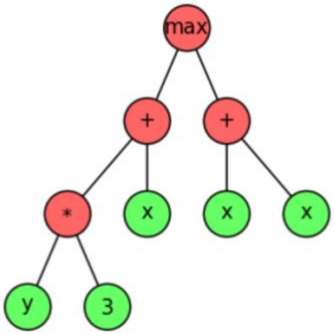


Figure 3-2: Illustrative tree was taken from the documentation of the deap library (Fortin and Rainville and Gardner and Parizeau and Gagné 2012)

3.1.1. Mutation

Mutation is based on creating a variation to one of the branches of a tree, as shown in the diagram below. Deap (Fortin and Rainville and Gardner and Parizeau and Gagné 2012) is the most used library of GP in python. It has six types of mutations, of which TPOT uses three (mutshrink, mutnodeReplacement , and mutinsert). These configurations will not be discussed in depth in this work because only the values that TPOT recommends by default will be used.

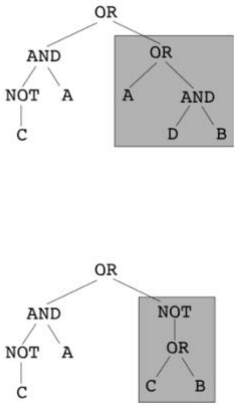


Figure 3-3: Example of a type of mutation (Vanneschi)

3.1.2. Crossover

It is based on randomly exchanging one of the parts of a tree with that of another tree. (Vanneschi). The original parent trees and the resulting children will be called in this case. Deap (Fortin and Rainville and Gardner and Parizeau and Gagné 2012) has three different crossovers, of which TPOT uses one which is the cxonepoint. (Félix Antoine Fortin, François Michel De Rainville, Marc André Gardner, Marc Parizeau, and Christian Gagné, 2012)

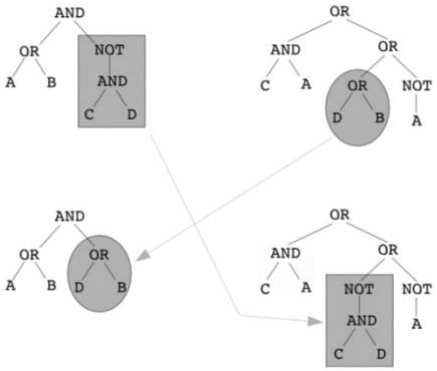


Figure 3-4: Example of a type of crossover (Vanneschi)

3.1.3. Fitness function

The function allows us to compare how well individuals have behaved compared to others. (Koza, 1992; Vanneschi). In the case of this work and in TPOT, the same ones are used as in a classic scikit-learn ML model (Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. et al., 2011). To understand a little more thoroughly how GP works, here are the steps to execute the algorithm considering the previous definitions.

1. Generate an initial population of computer programs (or individuals).

2. Iteratively perform the following steps until the termination criterion has been satisfied:

   2.1. Execute each program in the population and assign it a fitness value according to how well it solves the problem.
   2.2. Create a new population by applying the following operations:
      2.2.1. Probabilistically select a set of computer programs to be reproduced, on the basis of their fitness (selection).
      2.2.2. Copy some of the selected individuals, without modifying them, into the new population (reproduction).
      2.2.3. Create new computer programs by genetically recombining randomly chosen parts of two selected individuals (crossover).
      2.2.4. Create new computer programs by substituting randomly chosen parts of some selected individuals with new randomly generated ones (mutation).

3. The best computer program appeared in any generation is designated as the result of the GP process at that generation. This result may be a solution (or an approximate solution) to the problem.

Figure 3-5: GP Process (Vanneschi)

Some advantages of GP are that it is not necessary to have specific knowledge of the problem when solving it, and it is not affected by local maxima (false solutions) (Sivanandam & Deepa, 2008) (Koza, 1992), which makes it ideal for solving ML problems.

3.2. Tree-based pipeline optimization tool (TPOT)

TPOT is the AutoML framework that will be used in this thesis. It is an open-source library that is developed in the Python programming language. TPOT, through GP. This can be used to build the best tree-based pipeline from different ML models, selectors, and transformers. One of the most used libraries in TPOT is scikit-learn (Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. et al., 2011). Additionally, TPOT uses deap (Fortin and Rainville and Gardner and Parizeau and Gagné

2012) as the main library in GP. TPOT allows the user to solve both regression and classification problems. For this work, the results will be focused on the optimization of TPOT for classification (Olson & Moore, 2019)

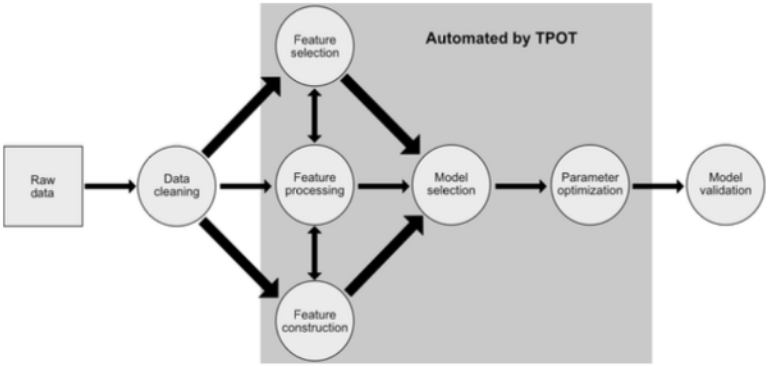


Figure 3-6: Example ML pipeline (Olson & Moore, 2019)

3.3. Simulated annealing (SA)

SA is a technique of optimization with the purpose of finding the optimal result for a problem. Models will be evaluated with performance measures for comparing ML algorithms. The default measures in scikit-learn (Olson & Moore, 2019; Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. et al., 2011) library will be used. The algorithm begins with a random solution and then tries to progress it by building on minor changes. It is accepted if the new result is superior to the last one. If the new result is inferior to the previous one, it is accepted with a certain probability, which is calculated by a parameter (Vanneschi Leonardo, s. f.; Wong et al., 2011)

Algorithm 2: Pseudo-code of the Simulated Annealing for an instance of an optimization problem (S, f) and a neighborhood structure N.

1. Initialize a feasible solution i_{start} from the search space S (typically at random);
2. $i := i_{start}$; // Let the current solution i be equal to i_{start}
3. Initialize L and c;
   // L is the number of iterations of the internal loop, c is the control parameter
4. **repeat until** *termination condition*
   - 4.1. **repeat** L **times**
     - 4.1.1. Generate a solution j from $N(i)$;
     - 4.1.2. **if** ($f(j)$ is better or equal than $f(i)$) **then**
       - $i := j$; // Let j become the new current solution
     - **else if** $\left(\text{Rand}[0,1) < e^{\frac{-|f(j)-f(i)|}{c}}\right)$ **then**
       - $i := j$; // Let j become the new current solution
     - **end**
   - 4.2. Update c;
   - 4.3. Update L;
   - **end**
5. **return** the solution with best fitness encoutered so far;

Figure 3-7: Pseudo-code of the SA (Vanneschi)

For the case of SA with ML, an individual will be one to whom it will randomly select the values of each parameter

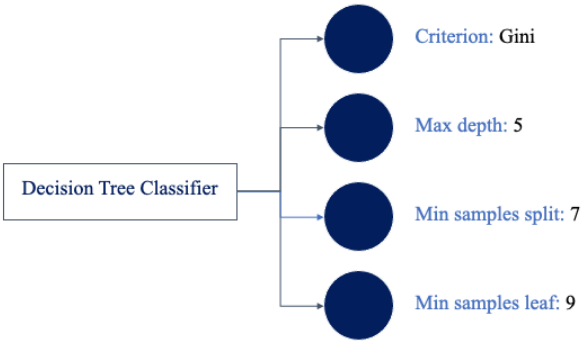


Figure 3-8: Example Individual i SA

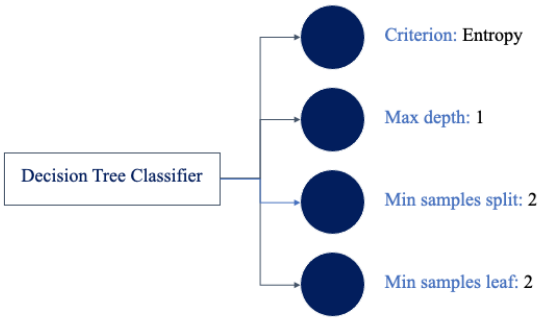


Figure 3-9: Example Individual j SA

The results of the execution of these individuals will be compared, leaving the best model the one that always has the best result. Furthermore, the rules of the algorithm mentioned above will be applied. The advantage of applying this technique is that it is fast compared to GP and allows us to have an optimal point for the different ML models that will be included in TPOT. (Koza, 1992).

3.4. Implementation

3.4.1. GP and SA

The two methodologies GP, and SA, will be integrated as follows. The first step of GP is based on creating an initial population of programs, in this case, randomized ML algorithms. This first step is where the preselected algorithms that resulted from running SA will be included. The rest of the steps will remain the same for GP. The change in the GP algorithm is basically to add to the initial individuals the optimal individuals selected by SA

3.4.2. TPOT and SA

The first thing to consider is how the python library will need to be set up and configured. As TPOT (Le, T. T., Fu, W., & Moore, J. H., 2018) is an open-source resource, user can download the package at the following link https://github.com/EpistasisLab/tpot/ . It cannot be installed traditionally like other packages with the *pip command install TPOT*. In the following link are the instructions on how you can contribute to the library http://epistasislab.github.io/tpot/contributing/ .

After performing the initial set up, it is possible to obtain results running with a sample dataset.
The first thing TPOT generates is the individuals with whom it will work. For this document, all the experiments were executed with 20 individuals since more individuals will require a longer execution time. Primitive trees are built with deap library (Fortin and Rainville and Gardner and Parizeau and Gagné 2012). These primitive trees are modified in such a way that by randomly selecting models, classifiers, and selectors, the structure of a tree will become a Python executable pipeline.

Here's an example of how a model looks like with its different possible Hyper-parameters.

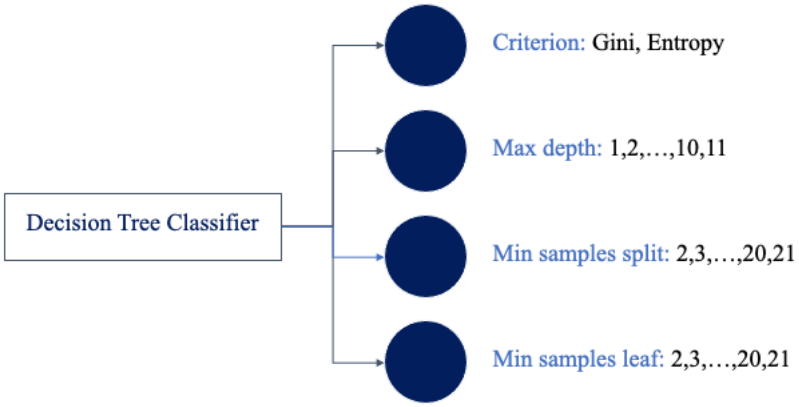


Figure 3-10: List of Hyper-parameters for a decision tree model

In this example, it can be seen how for a Decision Tree Classifier, all the possible combinations of results, if each model were executed, would be 7600. However, due to the advantage that SA has in searching for optimal solutions, it will be guaranteed to have hyper-parameters with its optimal performance. To continue with the description of the algorithm, for the hyper-parameters of a model, it is not easy to talk about neighbors if the continuous values were left stable. Categorical variables, such as criteria with a Gini or entropy option, do not have an ordinal level. Therefore, each model that is run on the SA will be using randomly selected parameters. Allowing different combinations during the various executions that generate a superior result is because the best possible solution will always be obtained compared to those previously executed.

In each pipeline, it will be possible to measure how well the pipeline is solving the classification problem according to the selected performance measure. These TPOT individuals allow users to generate them with the different model dictionaries according to how you want to configure them. In the case of our thesis, three dictionaries will be used: Default, Light, and Sparse. The description of these dictionaries is in Annexes. The first is the one that comes by default in TPOT. The light dictionary is the most effective for running fast models, and the sparse works with hot-encode and supports sparse matrices.

Every possible model in the selected dictionary in TPOT will be executed with SA. In other words, in the case of the light configuration, there are six different models. Gaussian NB, Bernoulli NB, Multinomial NB, Decision Tree Classifier, K-Neighbors Classifier, and Logistic Regression. Each algorithm had a different set of parameters to optimize. In this specific case, some restrictions were applied. For Gaussian NB, there is no parameter. Therefore, SA will not be executed in this specific case.

Additionally, Multinomial NB does not work with dependent variables that contain negative values. Another caveat is that when models such as K-Neighbors Classifiers are executed with many neighbors, the algorithm results in the value 1, which means it can predict each value. Moreover, since we know that overfitting is severe, these values, which do not exist in real life, will not be considered either.

TPOT, as a library, has different parameters that can be configured. For this work, 20 individuals with a certain number of generations are to be determined by running results and seeing where the data stabilizes. We are looking to find a value less than 100, which is the library's value by default. This means that the execution time could be reduced from the default of 100 generations TPOT utilizes. Each experiment can take days to run, so it was preferred to reduce the generations to optimize training time.

The values of the TPOT configurations for mutation and crossover will be kept with the default values in the library. This thesis will use the following values for the SA parameters. For the values of L and m, two scenarios will be compared to find which one takes the shortest execution time with optimal

results. In the case of the probability of acceptance, it will be $Rand[0,1] < e^{(\frac{-|f(j)-f(i)|}{100/(l+1)})}$. This is because the use of other parameters was compared, and the execution time was considerably higher. Later we will show in detail the comparison with other parameters, and how it would be a computational waste to execute a model more times than necessary.

3.4.3. Experiments

There are two ways to integrate TPOT results with SA:

- The first is to integrate the dictionaries with which TPOT will start executing the code, allowing individuals to be randomly selected.
- The second is forcing the individuals to be the ones that have been preselected but introduces more programming complexity since not only the TPOT library in Python will have to be changed but also the deap library (Fortin and Rainville and Gardner and Parizeau and Gagné 2012).

For this study, the first solution proposal was selected.

A dictionary will be generated with the parameters that best result from SA for each ML model. This means that the selectors and Transformers will remain stable. This dictionary will be integrated in two ways within TPOT:

1. The first test (TPOT+SA): The SA results are added to the original TPOT dictionary. As the individuals are randomly selected, they may or may not have those of SA.

1. Second SA test: ML models from the original dictionary will be removed, leaving only the best results selected by SA

1. The third test (TPOT control set): TPOT will be executed without SA to compare results with the two previous results.

Three different experiments are compared due to the random nature of GP. Each result will be executed ten times to be able to compare that the results are not only due to a lucky selection of parameters but that they are results that are maintained despite the randomness.

Example of the selection of individuals according to the experiments

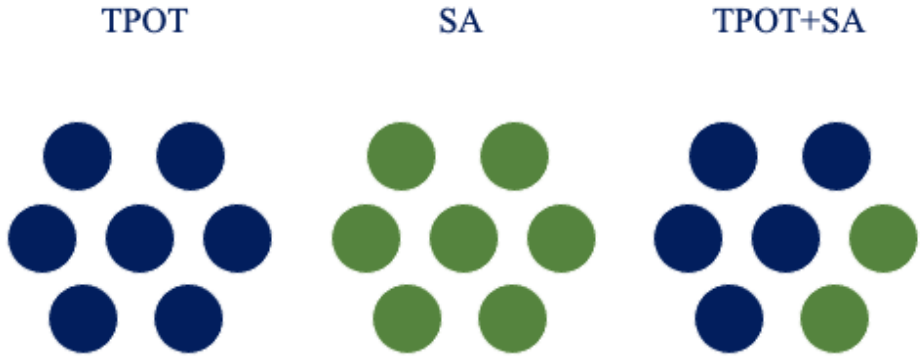


Figure 3-11: Example individuals per experiment

The SA+TPOT experiment will have more complexity since it must execute everything that TPOT already does, plus SA.

3.4.4. Datasets

SA and the experiments will be executed for each model with a stratified sample according to the objective variable of 75% of the data. The same sample with seed=34 will be run for each data set. Moreover, 25% of the data will be left as a test that will never be included in the TPOT or SA training until the final pipeline is ready.

Three datasets will be used to solve classification problems. All the sets were extracted from www.kaggle.com; This is a popular website within the data science community, where users often compete to have the best ML models.

Set 1 - Titanic dataset (Train)

It is a data set that came by default in the TPOT library to give new users an introduction to sample data sets. Only the training data set was used, which was divided into a new train and test data sets. The purpose of this dataset is to predict whether the passenger of the Titanic ship survived the 1912 collision. https://www.kaggle.com/c/titanic/data

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Figure 3-12: Titanic Data Dictionary

Set 2 - Cardiovascular study dataset:

The objective variable of this data set is to predict whether the patient is at risk of having Congenital heart defects (CHD) during the next ten years. The data was taken from a study on residents of the town of Framingham, Massachusetts.https://www.kaggle.com/datasets/christofel04/cardiovascular-study-dataset-predict-heart-disea

Demographic:
• **Sex**: male or female("M" or "F")
• **Age**: Age of the patient;(Continuous - Although the recorded ages have been truncated to whole numbers, the concept of age is continuous)
Behavioral
• **is_smoking**: whether or not the patient is a current smoker ("YES" or "NO")
• **Cigs Per Day**: the number of cigarettes that the person smoked on average in one day.(can be considered continuous as one can have any number of cigarettes, even half a cigarette.)
Medical(history)
• **BP Meds**: whether or not the patient was on blood pressure medication (Nominal)
• **Prevalent Stroke**: whether or not the patient had previously had a stroke (Nominal)
• **Prevalent Hyp**: whether or not the patient was hypertensive (Nominal)
• **Diabetes**: whether or not the patient had diabetes (Nominal)
Medical(current)
• **Tot Chol**: total cholesterol level (Continuous)
• **Sys BP**: systolic blood pressure (Continuous)
• **Dia BP**: diastolic blood pressure (Continuous)
• **BMI**: Body Mass Index (Continuous)
• **Heart Rate**: heart rate (Continuous - In medical research, variables such as heart rate though in fact discrete, yet are considered continuous because of large number of possible values.)
• **Glucose**: glucose level (Continuous)
Predict variable (desired target)
• **10 year risk of coronary heart disease CHD**(binary: "1", means "Yes", "0" means "No")

Figure 3-13: Cardiovascular Data Dictionary

Set 3 - Santander customer satisfaction:

Banco de Santander's data set makes it possible to predict whether a customer is satisfied with the bank's experience. This database is anonymized. That is, the names of the variables are not given, and it not possible to personally identify the individuals involved. https://www.kaggle.com/competitions/santander-customer-satisfaction/data

```
imp_ent_var16_ult1
imp_op_var39_comer_ult1
imp_op_var39_comer_ult3
imp_op_var40_comer_ult1
imp_op_var40_comer_ult3
imp_op_var40_efect_ult1
imp_op_var40_efect_ult3
imp_op_var40_ult1
imp_op_var41_comer_ult1
imp_op_var41_comer_ult3
imp_op_var41_efect_ult1
imp_op_var41_efect_ult3
imp_op_var41_ult1
imp_op_var39_efect_ult1
imp_op_var39_efect_ult3
imp_op_var39_ult1
imp_sal_var16_ult1
```

Figure 3-14: Example of a part of the column names

Set 4 - Pistachio dataset:

The objective of this dataset is to predict between two different species of pistachio, Kirmizi pistachios and Sirt pistachios, which correspond to the most significant export from Turkey. (Singh et al., 2022). The database is made up of 12 characteristic variables of pistachios and 16 variables corresponding to the shape and color of the images.
https://www.kaggle.com/datasets/muratkokludataset/pistachio-image-dataset

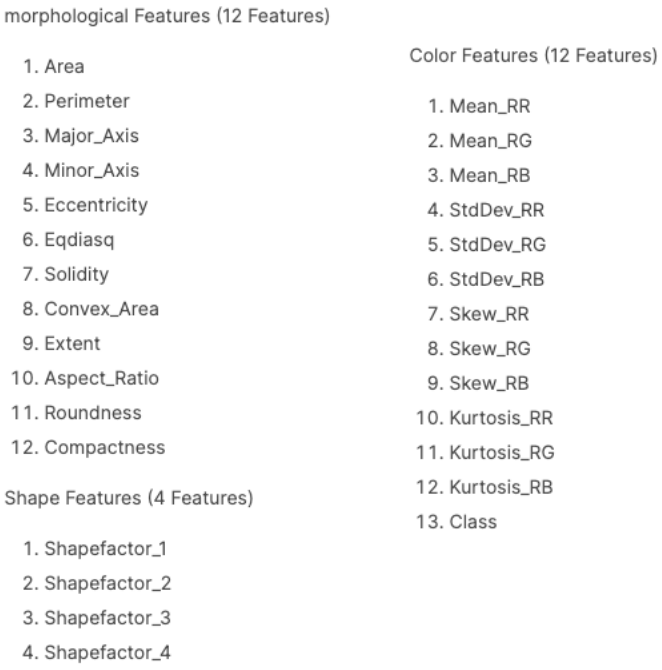


Figure 3-15: Pistachio Data columns

3.4.5. Data Cleaning

TPOT is a robust library speaking in ML terms. However, it is necessary to perform preprocessing to ensure good quality data. For all the data sets mentioned above, the variables with which they cannot be modeled, such as identifiers or proper names, were eliminated. Also, when the categorical variables had values in string format, these were transformed into numbers. This was done with a preprocessor called Sklearn's MultiLabelBinarizer. (Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. et al., 2011) This was done since not all ML models work with string variables, for example, KNN. Therefore, with this transformation, potential sources of errors will be avoided.

3.4.6. Evaluation of results

The measure of selection of the best model will be the f1-score in both SA and TPOT algorithms because we are working with unbalanced databases (Humphrey et al., 2022), and this would be the ideal measure. For now, this study will not focus on the logic of the target variable in solving classification problems, i.e., if we have a case of customer satisfaction as the target variable. We will not care about how important it is to be satisfied or not in addition to the f1-score. Average best fitness (ABF) will be calculated as the average f1-score of the ten runs for each generation. Additionally, the runtime for each TPOT run (10) will be recorded. This will be used in combination with the f1-score to determine the optimal configuration.

To compare, if the experiment's results have significant differences, the Wilcoxon test will be executed. It is a non-parametric test that compares two populations. (Woolson, 2008).

$$Ho: The\ difference\ between\ the\ pairs\ follows\ a\ symmetric\ distribution\ around\ zero.$$
$$Ha: The\ difference\ between\ the\ pairs\ does\ not\ follows\ a\ symmetric\ distribution\ around\ zero.$$

The p-value will be used as a measure of rejection in the test. Taking that values greater than 0.05 will be assumed as not enough statistical evidence to reject the null hypothesis.

Also, for interpretive purposes, for TPOT+SA vs. SA vs. TPOT, the average time will be taken, and the percentage change will be compared to the control group in the last run.

3.4.7. Virtual Machines

Virtual machines were created in Microsoft Azure, which is Microsoft's cloud. The reason for this decision, before deciding to go to the cloud, it was tried to execute TPOT in a traditional computer. Because the memory was insufficient for the execution, the computer automatically closed the session and never got clear results. By creating virtual machines in the cloud, the user can also run different databases simultaneously without being limited by having to run one dataset after another. The machines with the best performance but at a lower cost were selected. The virtual machines were created in the following link. https://portal.azure.com.

3.4.8. Repository

GitHub was used as a repository. This program allows user to save the changes made to the code and copy it to different machines without changing the files one by one. At the same time, it allows users to go back to previous versions if necessary.

3.4.9. Other technologies

The code editor used in this work was Visual Studio Code, which was integrated with the GitHub repository. The environment with the necessary characteristics to use TPOT was created with Anaconda, and Microsoft Excel and pandas profiling (Brugman, 2019) were used to create tables and graphs.

4. Results

This chapter will present the results of the methodology explained above.

4.1. Exploratory Analysis:

After performing the data cleaning mentioned in the previous section and discarding variables not used in the study. The following data sets were used:

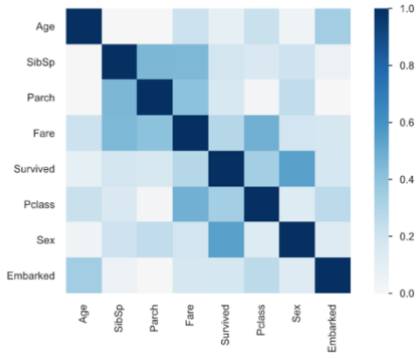
Dataset	Number of observations	Number of variables	Train size	Test size	Total size in memory	Executed dictionaries
Titanic	891	8	668	223	55.8 KiB	Light, Sparse, Default
Cardiovascular	3390	16	2542	848	423.9 KiB	Light, Sparse, Default
Pistachio	2148	28	1611	537	486.8 KiB	Light, Sparse, Default
Santander	76020	369	57015	19005	215.2 MiB	Not posible

Table 4-1: Characteristics of the studied datasets

The Santander dataset is the largest in several observations and variables and, therefore, in memory size, as these experiments can take days to run. Santander dataset was excluded after two days of running the data. It was barely possible to get three executions for the light dictionary, which is the one that processes the fastest. If the three experiments had been executed with the 30 executions it took, it would have taken 20 days to run. And because virtual machines were to run, there was a time limit (1 month) and a budget limit (200 USD). With the other datasets, the 200 USD budget was spent on running the five virtual machines, which in the end, limited the study because there were no funds remaining to run more data sets.

Next, a description of the independent variables of the data set and their correlation matrix is made. For all data sets, the target variable was defined as a class.

Titanic dataset



Cardiovascular dataset

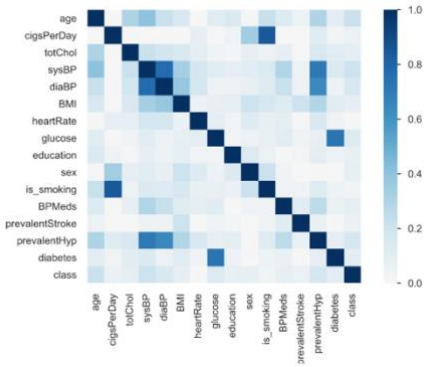


Figure 4-1: Bar chart for the target variable and correlation matrix for Titanic and Cardiovascular

Pistachio dataset

Santander dataset

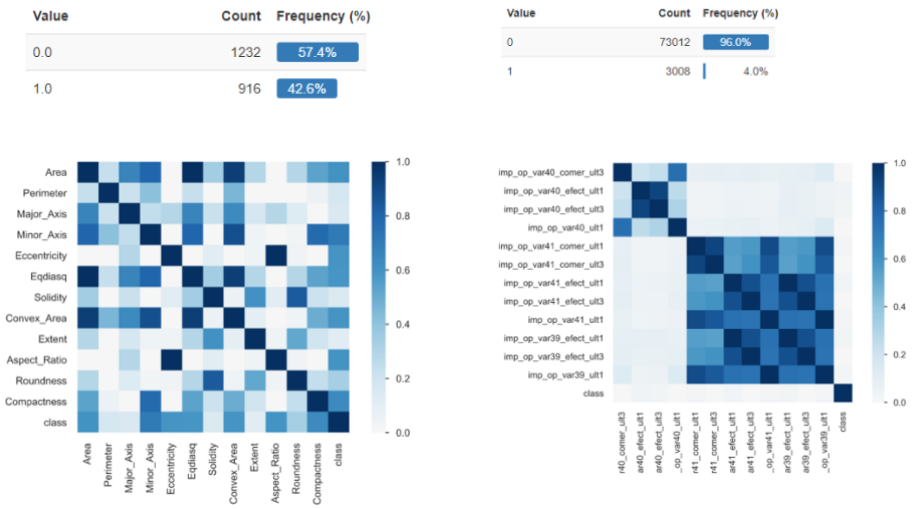


Figure 4-2: Bar chart for the target variable and correlation matrix for Pistachio and Santander

In general, it was found that the four datasets have different proportions in their independent variable and how bar charts are displayed with frequency. To generate the sample, a stratified sample selection was used according to the objective variable to maintain the same proportion.
The correlation matrix shows that, in general, there are not variables that were calculated with the target variable.

The exploratory study was carried out for four datasets; however, from this moment, only results will be presented for the three datasets where the experiments could be executed. This means that 27 different combinations of results between dictionary, dataset, and experiments were performed. In the end, there will be nine groups to which we will compare TPOT, SA, and TPOT+SA

No	Dataset	Dictionary	Experiment
1	Titanic	Light	TPOT
2	Titanic	Light	SA
3	Titanic	Light	TPOT+SA
4	Titanic	Sparce	TPOT
5	Titanic	Sparce	SA
6	Titanic	Sparce	TPOT+SA
7	Titanic	Default	TPOT
8	Titanic	Default	SA
9	Titanic	Default	TPOT+SA
10	Cardiovascular	Light	TPOT
11	Cardiovascular	Light	SA
12	Cardiovascular	Light	TPOT+SA
13	Cardiovascular	Sparce	TPOT
14	Cardiovascular	Sparce	SA
15	Cardiovascular	Sparce	TPOT+SA
16	Cardiovascular	Default	TPOT
17	Cardiovascular	Default	SA
18	Cardiovascular	Default	TPOT+SA
19	Pistachio	Light	TPOT
20	Pistachio	Light	SA
21	Pistachio	Light	TPOT+SA
22	Pistachio	Sparce	TPOT
23	Pistachio	Sparce	SA
24	Pistachio	Sparce	TPOT+SA
25	Pistachio	Default	TPOT
26	Pistachio	Default	SA
27	Pistachio	Default	TPOT+SA

Table 4-2: List of experiment groups

4.2. Parameter selection

4.2.1. Parameters simulated annealing

For the Titanic dataset for light dictionary, the results obtained were compared between($i = 50$, $l = 50$) with ($i = 5, l = 10$).

Light dictionary models	Possible combination of models	$L = 5\ and\ i = 10$ F1- score	$L = 50\ and\ i = 50$ F1- score
sklearn.naive_bayes.BernoulliNB	12	0,252	0,252
sklearn.naive_bayes.MultinomialNB	12	0,719	0,719
sklearn.tree.DecisionTreeClassifier	7600	0,841	0,923
sklearn.neighbors.KNeighborsClassifier	400	0,982	0,982
sklearn.linear_model.LogisticRegression	44	0,727	0,727

Table 4-3: SA results with two parameter scenarios.

The results in terms of f1-score were similar. Except for the Decision Tree Classifier, which has more possible combinations of models, the score result decreased with fewer executions. This was to be expected knowing that it has 7600 possible combinations.

Experiment	Time minutes	% Time minutes	Maximum executions
$L = 50\ and\ i = 50$	7,47	100%	2500
$L = 5\ and\ i = 10$	0,12	2%	50

Table 4-4: SA time results with two parameter scenarios.

The execution time of the two experiments was compared, and the SA execution decreased by 98%. After comparing the two results, it was decided to use the experiment with the shortest execution time since its impact on the f1-score does not seem significant.

4.2.2. Generations

A test was done with the Titanic data set running 100 generations to decide which number of generations would be the most recommended.

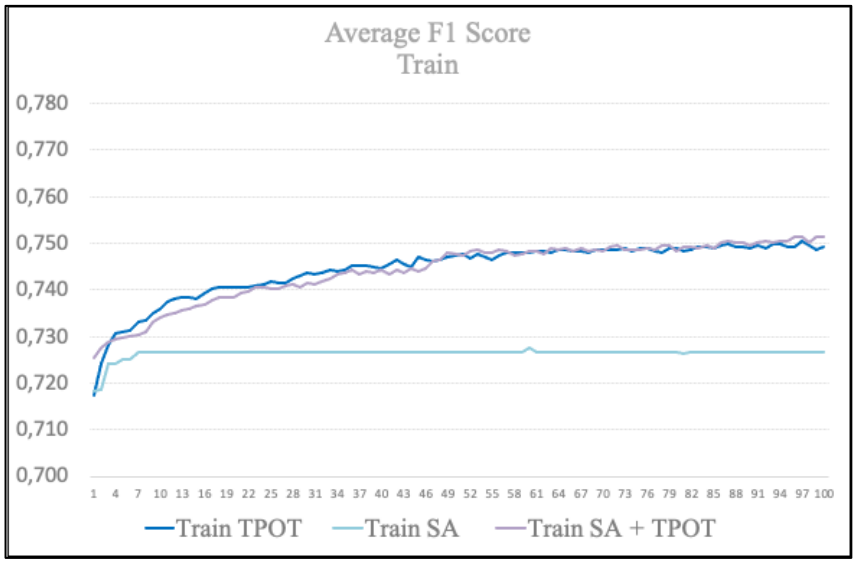


Figure 4-3: Titanic line chart for 100 generations

Moreover, the graph presented in figure 4-3 shows how the score stabilizes over time. The 100 executions that have TPOT by default are not necessary. Therefore 50 generations will be used in each experiment. This will help reduce the waiting time of the algorithm, and at the same time, with 50 observations per execution, it will be enough to perform the Wilcox test (Woolson, 2008).

4.3. Computer equipment selected

Three different virtual machine configurations were used with the characteristics shown in the following table.

Machine	RAM	vCPUs	Size	Operating system	Optimal
Virtual Machine 1	16 GB	4	Standard D4ds v4	Windows (Windows 11 Pro)	yes
Virtual Machine 2	8GB	4	Standard F4s v2	Windows (Windows 11 Pro)	yes
Virtual Machine 3	4GB	2	Standar B2s	Windows (Windows 11 Pro)	no

Table 4-5: Configuration of machines created in Microsoft Azure

Not all machines worked correctly. Virtual Machine 3 crashed when executing the code and did not allow to continue using it, so it was discarded for TPOT use with a machine with those characteristics.

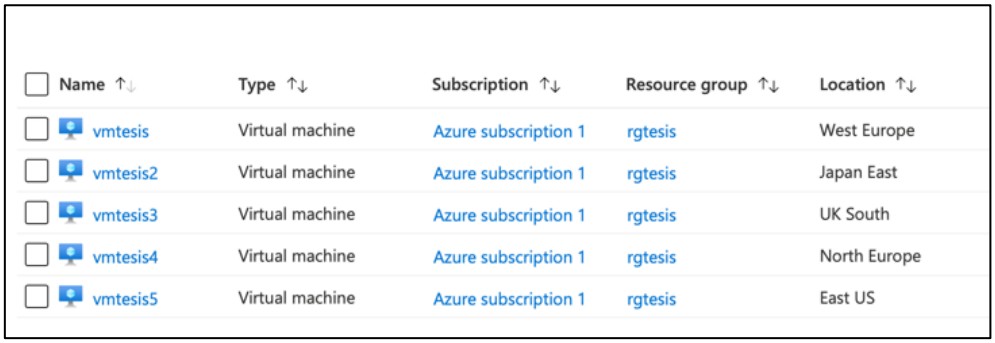


Figure 4-4: Characteristics of virtual machines in Azure, while they are running

Finally, during this work, after mentioning the valuable machines for the execution, five virtual machines with the characteristics mentioned above were created to execute all the experiments.

4.4. Results of the experiments

4.4.1. Titanic dataset

For the Titanic set, three dictionaries were run from TPOT: Light, sparse, and the library's default dictionary. There are three graphs with the average results of the f1-score by generations for the test and train. We applied it to our three experiments (TPOT, SA, TPOT+SA). These experiments are described below.

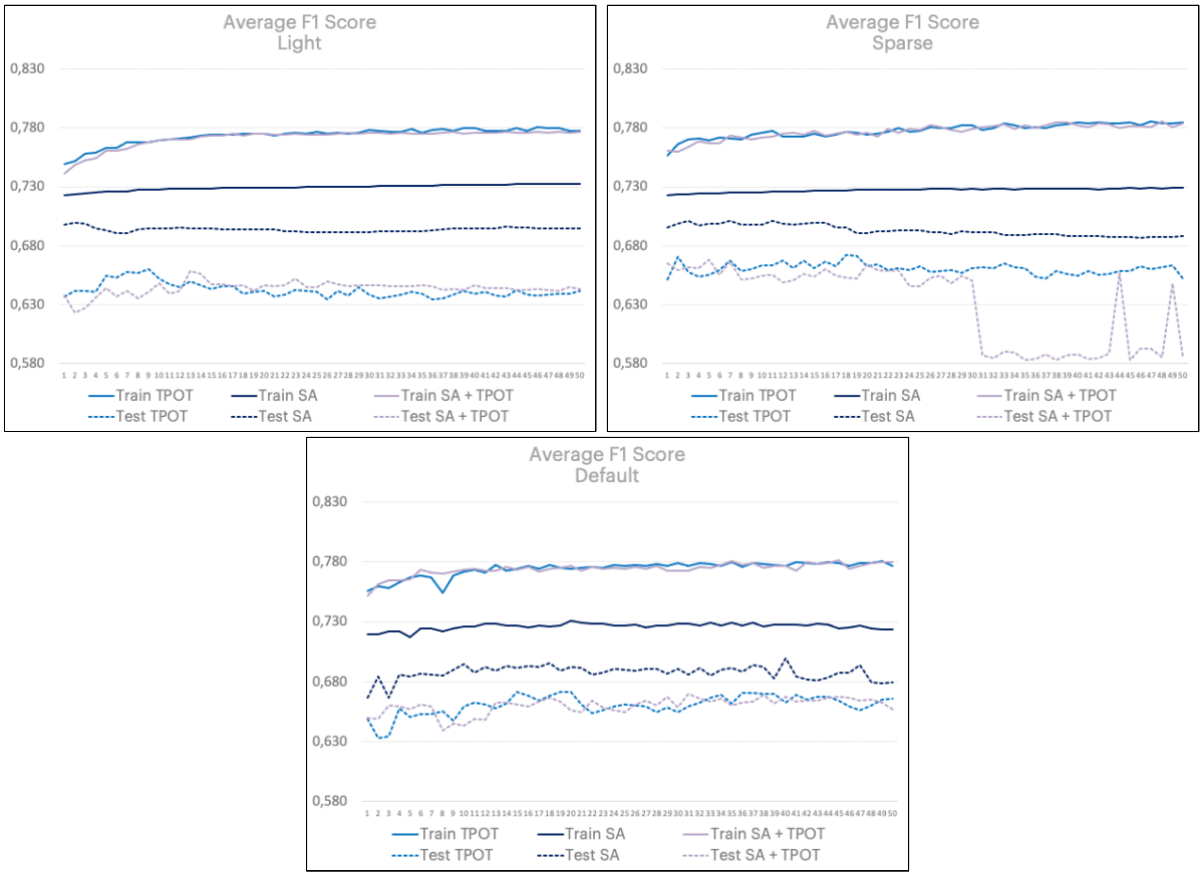


Figure 4-5: Titanic line graphs with the average f1 score per generation for train and test

According to the graphs 4-5 shown above:

- The SA result for the train in the light and sparse dictionaries remains the same over the generations.
- For training in SA+TPOT and TPOT, the data has similar values across generations.
- For the test results, the best scores of the three dictionaries are held by SA
- In tests for SA+TPOT in the last generations, there are some abrupt changes in values. These changes may be due to extreme values of the f1-score found when calculating the average for some generations.

For the Wilcoxon test, where it is studied if the distributions of the averages of the generations are equal, assuming that the threshold point is 0.05, the following results were calculated.

Experiment	Dictionary	P-value train	Result train	P-value test	Result test
TPOT vs. SA	Light	0,000	Reject	0,000	Reject
TPOT vs. TPOT+SA	Light	0,000	Reject	0,025	Reject
TPOT vs. SA	sparse	0,000	Reject	0,000	Reject
TPOT vs. TPOT+SA	sparse	0,058	No reject	0,000	Reject
TPOT vs. SA	default	0,000	Reject	0,000	Reject
TPOT vs. TPOT+SA	default	0,237	No reject	0,606	No reject

Table 4-6: Wilcoxon test for titanic data

Sufficient statistical evidence was found to reject the null hypothesis that the results of TPOT vs. SA are equal. All p-values are less than 0.05. However, in the case of TPOT vs. TPOT+SA, there is not enough evidence to confirm that the null hypothesis is rejected in some of the experiments, which makes sense since, at the time of selecting the individuals, they may be different. To those proposed by SA and therefore, the experiments would have similar results to just running TPOT.

The execution time of the different dictionaries in each experiment was also measured.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	1,6	1,4	2,3
2	2,1	1,4	1,7
3	2,1	1,4	1,9
4	1,7	1,7	2,0
5	2,2	1,8	3,9
6	1,4	1,8	1,9
7	2,2	1,3	2,6
8	1,6	2,1	1,8
9	1,3	1,5	2,5
10	1,9	1,9	2,3

	TPOT	SA	TPOT+SA
Average executions	1,81	1,63	2,28
Variation with respect to (TPOT)	0%	90%	126%

Table 4-7: Times execution and variations in Titanic data (Light)

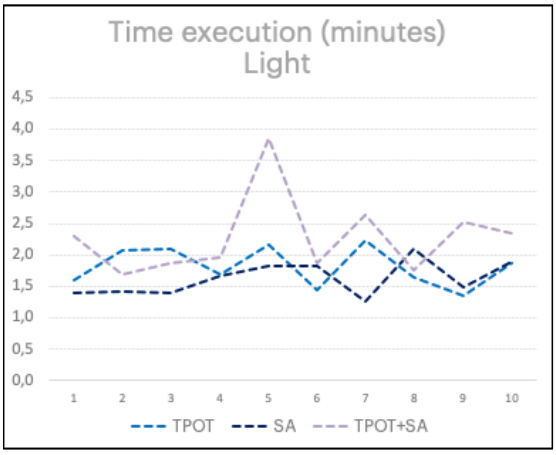


Figure 4-6: Graphical view of times per experiment for Titanic (Light)

The green color shows when the average execution time of the executions was less than the classic TPOT, and the red color shows when the average execution time exceeded TPOT.

For the Light SA vs. TPOT dictionary, it managed to reduce the execution time by 10%. On average, it takes 90% of the initial time. In the case of TPOT+SA, the average execution time increased by 26%. The graph on the right shows how TPOT and SA are the experiments that remain more stable during the executions—having TPOT+SA a considerable peak in run 5.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	18,1	6,5	15,6
2	22,6	5,9	16,0
3	22,9	8,6	15,8
4	22,0	8,0	18,0
5	22,5	7,6	16,4
6	18,3	8,2	15,4
7	17,1	9,5	16,9
8	19,2	5,1	21,4
9	28,7	5,9	13,2
10	18,6	6,2	19,4

	TPOT	SA	TPOT+SA
Average executions	21,00	7,15	16,81
Variation with respect to (TPOT)	0%	34%	80%

Table 4-8: Times execution and variations in Titanic data (Sparse)

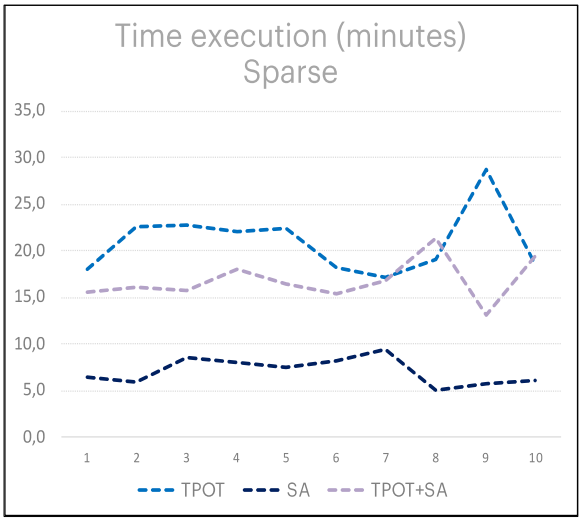


Figure 4-7 : Graphical view of times per experiment for Titanic (Light)

For the Sparse dictionary, SA vs. TPOT decreased execution time by 66%. On average, it takes 34% of the initial time. For TPOT+SA vs. TPOT, it was also possible to reduce the average execution time by 20%. On average, it takes 80% of the initial time. In the graph on the right, it can be seen how SA is the experiment that remains most stable during the executions. TPOT shows a considerable spike in run 9.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	73,6	14,4	72,3
2	31,5	13,0	47,8
3	40,6	27,3	25,5
4	17,6	25,4	83,9
5	59,4	25,8	40,7
6	55,1	17,6	60,4
7	58,2	29,8	70,6
8	75,6	20,2	57,8
9	68,9	34,2	54,2
10	23,2	19,1	67,0

	TPOT	SA	TPOT+SA
Average executions	50,36	22,68	58,02
Variation with respect to (TPOT)	0%	45%	115%

Table 4-9: Times execution and variations in Titanic data (Default)

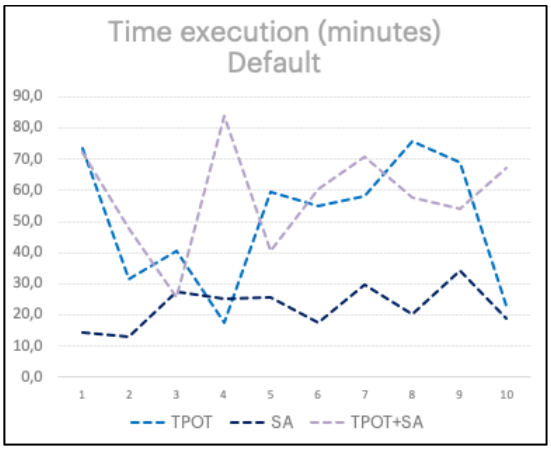


Figure 4-8: Graphical view of times per experiment for Titanic (Light)

SA vs. TPOT decreased execution time by 55% for the default dictionary. On average, it takes 45% of the initial time. In the case of TPOT+SA, the average execution time increased by 15%.
On average, it takes 115% of the initial time. In the graph on the right, it can be seen how SA is the experiment that remains most stable during the executions. TPOT and TPOT+SA wider variations during execution.

4.4.2. Cardiovascular dataset

Alternatively, for the cardiovascular set, three dictionaries of TPOT, Light, sparse, and the default dictionary of the library were executed. When the results are compared for the test and training between generations and experiments, it is observed that:

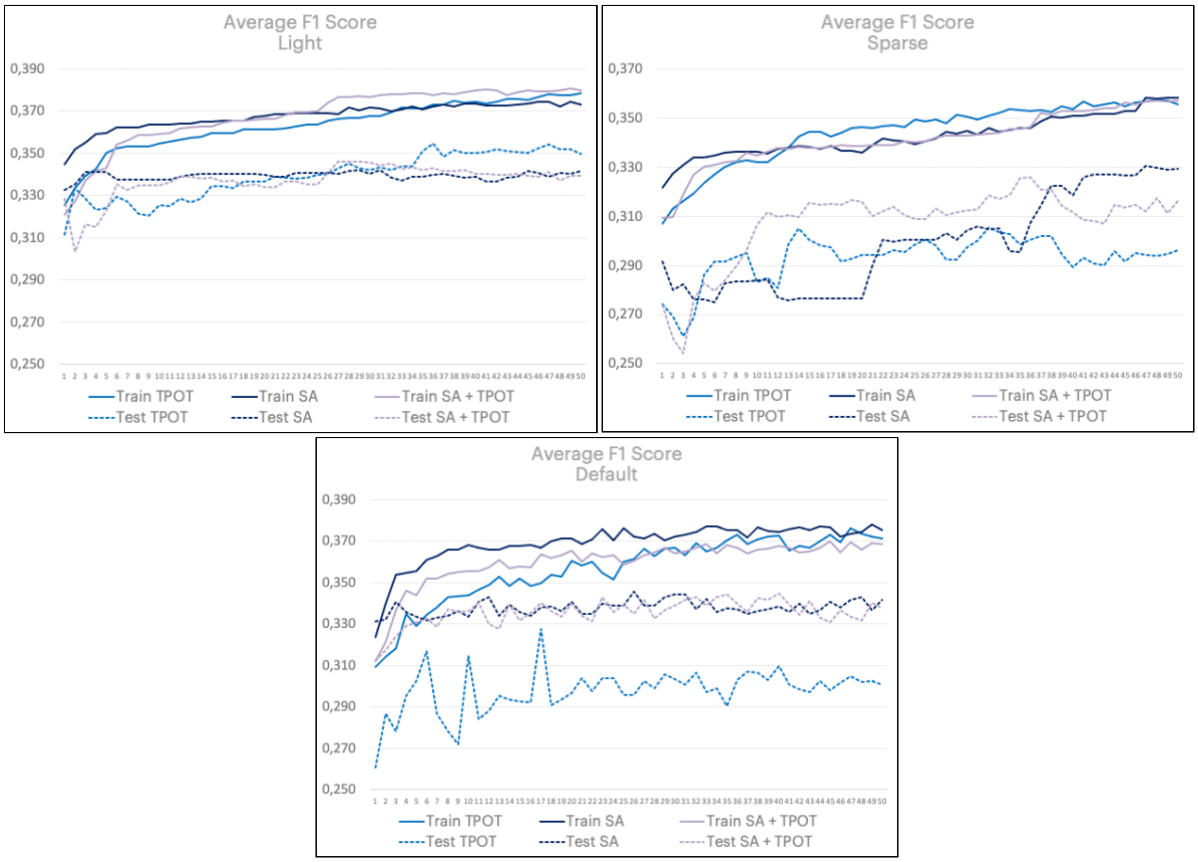


Figure 4-9: Cardiovascular line graphs with the average f1 score per generation for the train and test

- For the train in the two dictionaries in all the experiments, similar results were seen throughout the generations.
- For the test, the results maintain the increasing trend. However, for the light dictionary at the end of the 50 generations, traditional TPOT performs better in the last generations. In sparse, SA is the experiment that ends with the best results.

For the Wilcoxon test, where it is studied if the distributions of the averages of the generations are equal, assuming that the threshold point is 0.05, the following results were calculated.

Experiment	Dictionary	P-value train	Result train	P-value test	Result test
TPOT vs. SA	Light	0,00	Reject	0,98	No rejected
TPOT vs. TPOT+SA	Light	0,00	Reject	0,11	No rejected
TPOT vs. SA	sparse	0,02	Reject	0,05	No rejected
TPOT vs. TPOT+SA	sparse	0,00	Reject	0,00	Reject
TPOT vs. SA	default	0,00	Reject	0,00	Reject
TPOT vs. TPOT+SA	default	0,01	Reject	0,00	Reject

Table 4-10: Wilcoxon test for cardiovascular data

For this data set, both TPOT vs SA and TPOT vs TPOT+SA experiments were rejected. However, most of the experiments were maintained, they showed statistical evidence to reject that the distributions of the data are equal. Emphasizing that the rejected results correspond to the test.

The result of the execution times for each experiment with the different dictionaries was also measured.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	2,0	1,5	2,8
2	2,0	1,3	2,6
3	3,5	1,5	3,1
4	1,7	1,5	2,9
5	1,8	1,7	3,0
6	1,8	1,6	3,3
7	1,8	1,5	2,4
8	2,2	1,6	2,7
9	2,1	1,9	3,4
10	1,7	1,6	3,2

	TPOT	SA	TPOT+SA
Average executions	2,07	1,54	2,94
Variation with respect to (TPOT)	0%	74%	142%

Table 4-11: Times execution and variations in cardiovascular data (Light)

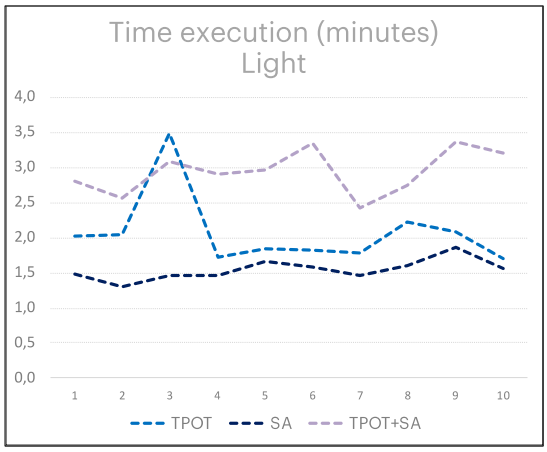


Figure 4-10: Graphical view of times per experiment for Cardiovascular (Light)

For the light dictionary in the table above, the execution time per iteration was calculated by calculating the average. For the SA, the average execution time was 74% of the TPOT value. That is, it decreased by 26 % time on average. However, in the case of the TPOT+SA mixture, the execution process was increased by 42%. In the graph on the right, it can be seen how SA is the experiment that remains most stable during the executions.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	18,0	16,4	17,7
2	22,2	18,1	33,3
3	27,3	19,1	21,4
4	22,9	20,9	24,6
5	28,9	17,8	18,4
6	20,5	16,6	20,6
7	38,0	17,8	19,8
8	29,1	22,7	18,2
9	22,8	17,5	18,5
10	24,4	20,1	20,4

	TPOT	SA	TPOT+SA
Average executions	25,41	18,71	21,29
Variation with respect to (TPOT)	0%	74%	84%

Table 4-12: Times execution and variations in cardiovascular data (Sparse)

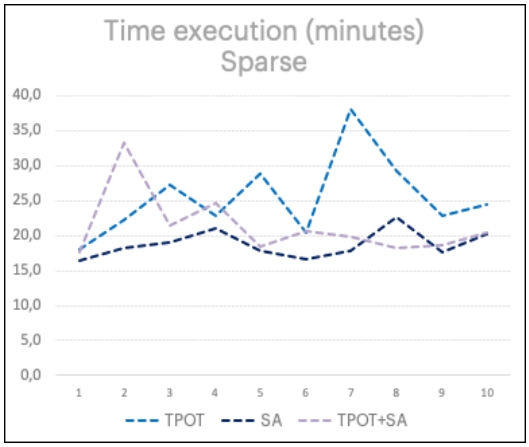


Figure 4-11: Graphical view of times per experiment for Cardiovascular (Sparse)

For the sparse dictionary, the average execution time is higher than that of the light, passing in all cases 18 minutes for each experiment. This dictionary is characterized by having hot encoding for the different variables, generating higher computational wear. The SA experiment took 74% of the original time. In the case of TPOT+SA, the time also decreased, going to 84% of the original time. During the executions, it is seen that the SA time remains stable over time.

However, for TPOT and TPOT+SA, it can be observed that some executions had a higher performance time.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	18,2	16,5	18,1
2	26,1	10,2	18,1
3	16,8	14,4	21,6
4	33,7	15,3	18,3
5	15,2	19,6	19,4
6	16,7	19,6	20,7
7	19,8	16,5	39,2
8	20,7	19,9	16,9
9	14,7	14,1	19,4
10	14,4	13,2	20,4

	TPOT	SA	TPOT+SA
Average executions	19,65	15,93	21,22
Variation with respect to (TPOT)	0%	81%	108%

Table 4-13: Times execution and variations in cardiovascular data (Default)

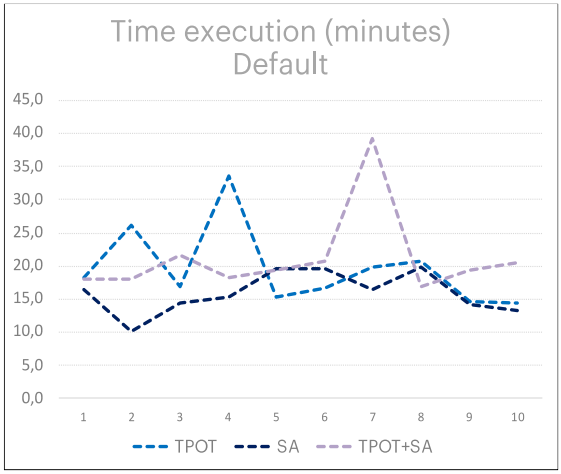
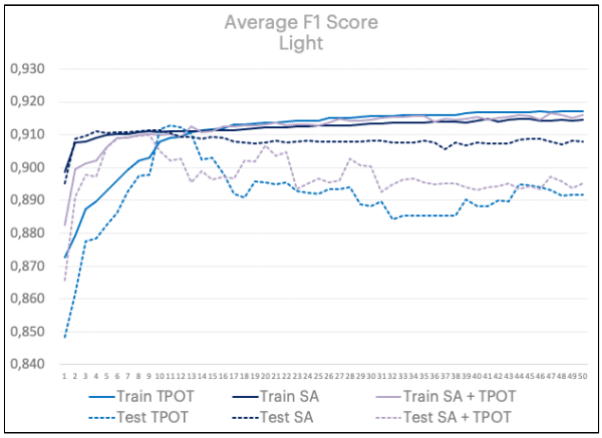


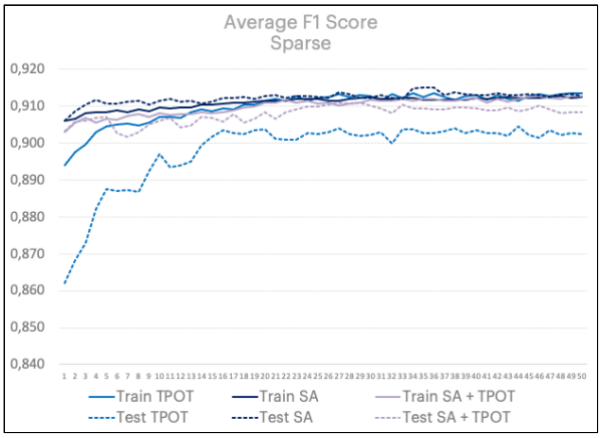
Figure 4-12: Graphical view of times per experiment for Cardiovascular (Default)

The default dictionary has execution times higher than light but lower than Sparse on average. SA maintained an average execution time lower than TPOT in 81% of its executions. For TPOT + SA, execution time increased by 8%. Again, in the plot across runs, it is seen that SA has fewer high peaks compared to the other groups in this data set.

4.4.1. Pistachio dataset

Three dictionaries from TPOT, Light, sparse, and the library's default dictionary were executed for the Pistachio set. When comparing test and train results between generations, it is observed that:





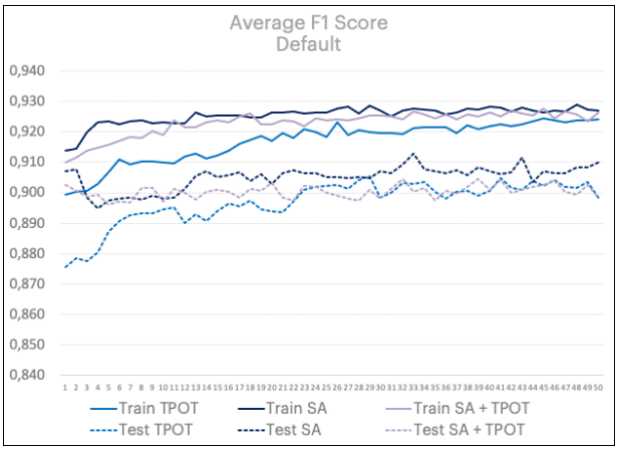


Figure 4-13: Pistachio line graphs with the average f1 score per generation for the train and test

The results of the training for light and sparse dictionaries have similar behavior, on default dictionary TPOT shows underperformance of the others.

- For the tests in SA, there are stable results that have higher values than the results of TPOT and TPOT+SA
- In the light dictionary graph, it is observed that the results during the test for TPOT and TPOT+SA seem to decrease after generation 10.

For the Wilcoxon test, where it is studied if the distributions of the averages of the generations are equal, assuming that the threshold point is 0.05, the following results were calculated.

Experiment	Dictionary	P-value train	Result train	P-value test	Result test
TPOT Vs SA	Light	0,15	No rejected	0,00	Reject
TPOT Vs TPOT+SA	Light	0,26	No rejected	0,00	Reject
TPOT Vs SA	sparse	0,14	No rejected	0,00	Reject
TPOT Vs TPOT+SA	sparse	0,11	No rejected	0,00	Reject
TPOT Vs SA	default	0,00	Reject	0,00	Reject
TPOT Vs TPOT+SA	default	0,00	Reject	0,00	Reject

Figure 4-14: Wilcoxon test for Pistachio data

For training, it cannot be concluded that the results are different for the light and sparse dictionaries. There is enough statistical evidence for the default dictionary to assume that the distributions are different between TPOT and the other two experiments. For test results, sufficient statistical evidence was found to reject the null hypothesis that the results of TPOT vs. SA and TPOT vs. TPOT+SA are equal, all p-values are less than 0.05 in test data.

The execution times of the three experiments were measured, and the following results were obtained.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	4,5	4,6	5,8
2	3,8	5,4	4,6
3	4,5	4,6	5,0
4	3,9	6,3	4,6
5	4,3	5,3	5,7
6	5,0	5,6	6,7
7	4,4	5,3	5,7
8	4,2	5,5	5,2
9	5,4	4,7	6,1
10	3,5	5,2	5,5

	TPOT	SA	TPOT+SA
Average executions	4,36	5,25	5,49
Variation with respect to (TPOT)	0%	120%	126%

Table 4-14: Times execution and variations in pistachio data (Light)

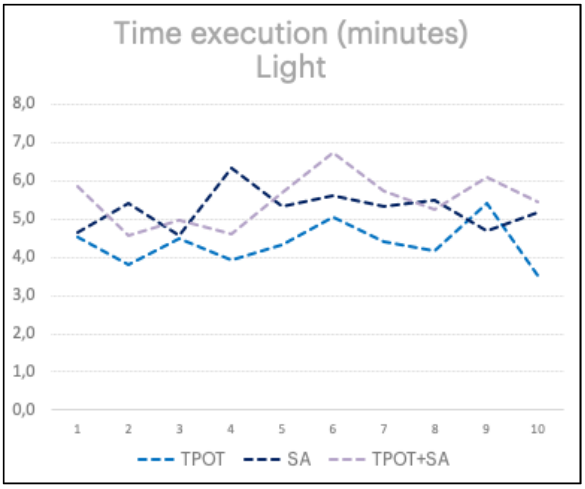


Figure 4-15: Graphical view of times per experiment for Pistachio (Light)

For the case of the SA and TPOT+SA experiments, the execution time concerning TPOT was higher by 20% and 26%, correspondingly. This may be because the light dictionary executes faster than the other dictionaries, and intervening in the process, only added complexity to the process.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	60,2	32,5	44,5
2	41,1	15,7	91,9
3	42,1	16,5	102,1
4	34,6	40,8	114,8
5	99,7	15,7	76,0
6	59,2	32,2	45,4
7	38,9	45,8	41,9
8	59,1	41,3	60,7
9	59,0	51,5	47,6
10	31,7	30,6	34,4

	TPOT	SA	TPOT+SA
Average executions	52,57	32,25	65,92
Variation with respect to (TPOT)	0%	61%	125%

Table 4-15: Times execution and variations in pistachio data (Sparse)

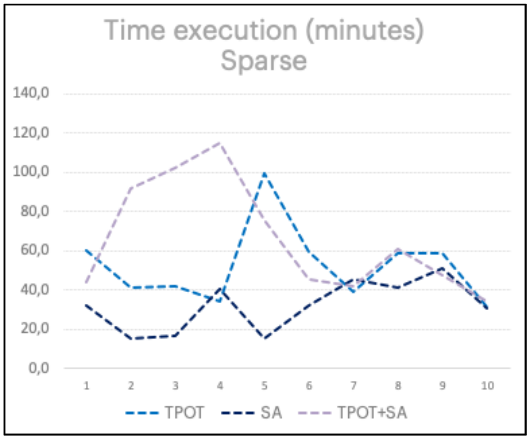


Figure 4-16: Graphical view of times per experiment for Pistachio (Sparse)

For the Sparse SA vs. TPOT dictionary, it managed to reduce the execution time by 39%. On average, it takes 61% of the initial time. TPOT+SA vs. TPOT increased execution time by 25%. In the graph on the right, it can be seen how SA is the experiment that remains more stable during the executions. TPOT had a prominent peak in run five and TPOT+SA in run 4.

Time executions (Minutes)			
Executions	**TPOT**	**SA**	**TPOT+SA**
1	86,7	37,6	78,3
2	63,7	92,9	116,1
3	86,8	59,3	61,0
4	115,3	65,7	105,0
5	181,7	45,8	144,8
6	71,5	74,2	96,1
7	63,6	40,3	126,1
8	102,2	53,2	99,2
9	125,4	86,1	105,0
10	291,0	51,7	83,6

	TPOT	SA	TPOT+SA
Average executions	118,79	60,66	101,50
Variation with respect to (TPOT)	0%	51%	85%

Table 4-16: Times execution and variations in pistachio data (Default)

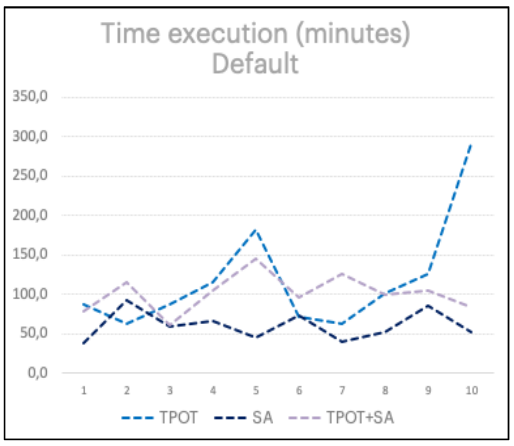


Figure 4-17: Graphical view of times per experiment for Pistachio (Default)

For the default SA vs. TPOT dictionary, it managed to reduce the execution time by 49%. On average, it takes 51% of the initial time. TPOT+SA vs. TPOT managed to decrease execution time by 15%. On average, it takes 85% of the initial time. In the graph on the right, it can be seen how SA is the experiment that remains more stable during the executions. Having TPOT a significant peak in run ten and TPOT+SA in run 5 and 10.

5. Conclusions and recommendations for future works

Finding the best hyperparameters of an ML model and selecting the best model is one of the biggest challenges a data scientist faces. Libraries like TPOT which perform AutoML are designed to help solve such challenges. TPOT has managed to reduce operational tasks, making possible combinations of all the best practices found in the literature today. As this thesis outlines, data scientist's also need to be aware of the disadvantages of relying on the use of these tools and techniques.

The key disadvantages of using AutoML techniques with the TPOT library are the increased execution time, and ensuring results are kept constant over time despite being executed within the same data set. The general objective of this study was to use the SA technique to include top individuals at the beginning of the program to help TPOT have a "smarter" way to start their process and not wait until a random selection has the best pipeline.

To conduct the experiments, it was necessary to discard one of the datasets because running results for a single dictionary took 20 days of execution. The budget was limited because virtual machines were used in Microsoft Azure, so the results were calculated for three datasets.

For the first experiment TPOT vs. SA (The difference is that TPOT is the original GP algorithm, and in SA, all individuals are replaced by those pre-selected initially by SA). These experiments were applied to 3 different dictionaries: Light, Sparse, and Default.

When comparing the f1-score results for training and test in two of the nine selected groups, no statistical evidence was found to show that the results between the experiments were different. For training, the three dictionaries of the titanic data set, the f1-score results for SA showed values below those of TPOT. The values in the last run remained similar for the other two datasets in the three dictionaries (6 experiments) according to the graphs. For the test results, in eight out of the nine results, the f1 score in the last generation was higher than the results of the original TPOT.
In terms of execution time, in eight out of the nine experiments, SA performed better than TPOT. In the best-case scenario, we could reduce the execution time by an average of 66% of the execution time. For the only case where the execution took longer, it took 20% more time than TPOT, on average.

For the TPOT vs. TPOT+SA experiment (The difference is that TPOT is the original GP algorithm, and in TPOT+SA, the optimal SA individuals are added, however, the random factor is still present and may or may not select those included in SA). These experiments were applied to 3 different dictionaries: Light, Sparse, and Default. For training, in four out of the nine dictionaries, no statistical evidence was found to determine that the results were different. There was insufficient evidence for the test in two out of the nine experiments. The test and train results had an average f1-score on the last run similar to the TPOT, according to the data collected.
In terms of run time, in six out of the nine runs, the TPOT+SA results were more time-consuming than TPOT. In the best case, a 20% run time saving was achieved. However, it took 42% more time than average in the worst case.

In conclusion, it is recommended to integrate TPOT with SA by replacing all individuals with those selected by SA, as the results for the test will remain stable or even better. Moreover, it will decrease the running time in most cases. Second, it is recommended to use something other than TPOT+SA, as the run time increases in more than half of the cases even if the f1- score remains similar.

As future work, SA could be extended to regression algorithms, which are also part of the TPOT library. It is also recommended to run the experiments on large databases. Finally, another technique called stacking ensembles (Hu et al., 2020) could be added, consisting of mixing machine learning algorithms.

6. References

Fortin and Rainville and Gardner and Parizeau and Christian Gagné. (2012). *"DEAP: Evolutionary Algorithms Made Easy"*. *13*, 2171--2175.

Fradkov, A. L. (2020). Early History of Machine Learning. *IFAC-PapersOnLine*, *53*(2), 1385-1390. https://doi.org/10.1016/j.ifacol.2020.12.1888

Furtado, F., & Singh, A. (2020). Movie Recommendation System using Machine Learning. *International Journal of Research in Industrial Engineering*, *Online First*. https://doi.org/10.22105/riej.2020.226178.1128

Dixon, M. F., Halperin, I., & Bilokon, P. (2020). *Machine learning in finance: From theory to practice*. Springer. https://doi.org/10.1007/978-3-030-41068-1

Guerra, P., & Castelli, M. (2021). Machine Learning Applied to Banking Supervision a Literature Review. *Risks*, *9*(7), 136. https://doi.org/10.3390/risks9070136

Humphrey, A., Kuberski, W., Bialek, J., Perrakis, N., Cools, W., Nuyttens, N., Elakhrass, H., & Cunha, P. A. C. (2022). Machine-learning classification of astronomical sources: Estimating F1-score in the absence of ground truth. *Monthly Notices of the Royal Astronomical Society: Letters*, *517*(1), L116-L120. https://doi.org/10.1093/mnrasl/slac120

Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.). (2019). *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing. https://doi.org/10.1007/978-3-030-05318-5

Hu, X., Zhang, H., Mei, H., Xiao, D., Li, Y., & Li, M. (2020). Landslide Susceptibility Mapping Using the Stacking Ensemble Machine Learning Method in Lushui, Southwest China. *Applied Sciences*, *10*(11), 4016. https://doi.org/10.3390/app10114016

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). *Fundamentals of machine learning for predictive data analytics: Algorithms, worked examples, and case studies*. The MIT Press.
Koza, J. R. (1992). *Genetic programming. 1: On the programming of computers by means of natural selection*. MIT Press.

Le, T. T., Fu, W., & Moore, J. H. (2018). *Scaling tree-based automated machine learning to biomedical big data with a dataset selector* [Preprint]. Genomics. https://doi.org/10.1101/502484

Le, Trang T and Fu, Weixuan and Moore, Jason H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, *36*(1), 250--256.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2016). *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. https://doi.org/10.48550/ARXIV.1603.06560

Lin, S.-W., Lee, Z.-J., Chen, S.-C., & Tseng, T.-Y. (2008). Parameter determination of support vector machine and feature selection using simulated annealing approach. *Applied Soft Computing*, *8*(4), 1505-1512. https://doi.org/10.1016/j.asoc.2007.10.012

Mosavi, A., Faghan, Y., Ghamisi, P., Duan, P., Ardabili, S. F., Salwana, E., & Band, S. S. (2020). Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics. *Mathematics*, *8*(10), 1640. https://doi.org/10.3390/math8101640

Olson, R. S., & Moore, J. H. (2019). TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. En F. Hutter, L. Kotthoff, & J. Vanschoren (Eds.), *Automated Machine Learning*

(pp. 151-160). Springer International Publishing. https://doi.org/10.1007/978-3-030-05318-5_8

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

Petrozziello, A., & Jordanov, I. (2017). Data Analytics for Online Travelling Recommendation System: A Case Study. *Modelling, Identification and Control*. Modelling, Identification and Control, Innsbruck, Austria. https://doi.org/10.2316/P.2017.848-041

Ravi, D., Wong, C., Deligianni, F., Berthelot, M., Andreu-Perez, J., Lo, B., & Yang, G.-Z. (2017). Deep Learning for Health Informatics. *IEEE Journal of Biomedical and Health Informatics*, *21*(1), 4-21. https://doi.org/10.1109/JBHI.2016.2636665

Singh, D., Taspinar, Y. S., Kursun, R., Cinar, I., Koklu, M., Ozkan, I. A., & Lee, H.-N. (2022). Classification and Analysis of Pistachio Species with Pre-Trained Deep Learning Models. *Electronics*, *11*(7), 981. https://doi.org/10.3390/electronics11070981

Sivanandam, S. N., & Deepa, S. N. (2008). Genetic Programming. En *Introduction to Genetic Algorithms* (pp. 131-163). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-73190-0_6

Sun, Z., Han, L., Huang, W., Wang, X., Zeng, X., Wang, M., & Yan, H. (2015). Recommender systems based on social networks. *Journal of Systems and Software*, *99*, 109-119. https://doi.org/10.1016/j.jss.2014.09.019

Vaccaro, L., Sansonetti, G., & Micarelli, A. (2021). An Empirical Review of Automated Machine Learning. *Computers*, *10*(1), 11. https://doi.org/10.3390/computers10010011

Vanneschi. *Computational Intelligence for Optimization*. Document extracted from the book: "Lectures in Intelligent Systems" Universidade Nova de Lisboa

Wong, W.-K., Chen, H.-Y., Hsu, C.-Y., & Chao, T.-K. (2011). Reinforcement Learning of Robotic Motion with Genetic Programming, Simulated Annealing and Self-Organizing Map. *2011 International Conference on Technologies and Applications of Artificial Intelligence*, 292-298. https://doi.org/10.1109/TAAI.2011.57

Woolson, R. F. (2008). Wilcoxon Signed-Rank Test. En R. B. D'Agostino, L. Sullivan, & J. Massaro (Eds.), *Wiley Encyclopedia of Clinical Trials* (p. eoct979). John Wiley & Sons, Inc. https://doi.org/10.1002/9780471462422.eoct979

Brugman, S. (2019). *pandas-profiling: Exploratory Data Analysis for Python*. https://github.com/pandas-profiling/pandas-profiling. Version: 3.5. License: MIT License (MIT)

Annexes

1. Dictionary Light TPOT

```
# -*- coding: utf-8 -*-

"""This file is part of the TPOT library.

TPOT was primarily developed at the University of Pennsylvania by:
    - Randal S. Olson (rso@randalolson.com)
    - Weixuan Fu (weixuanf@upenn.edu)
    - Daniel Angell (dpa34@drexel.edu)
    - and many more generous open source contributors

TPOT is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.

TPOT is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with TPOT. If not, see <http://www.gnu.org/licenses/>.

"""

import numpy as np
# Check the TPOT documentation for information on the structure of config dicts
classifier_config_dict_light = {
    # Classifiers
    'sklearn.naive_bayes.GaussianNB': {
    },
    'sklearn.naive_bayes.BernoulliNB': {
        'alpha': [1e-3, 1e-2, 1e-1, 1., 10., 100.],
        'fit_prior': [True, False]
    },
```

```
'sklearn.naive_bayes.MultinomialNB': {
    'alpha': [1e-3, 1e-2, 1e-1, 1., 10., 100.],
    'fit_prior': [True, False]
},

'sklearn.tree.DecisionTreeClassifier': {
    'criterion': ["gini", "entropy"],
    'max_depth': range(1, 11),
    'min_samples_split': range(2, 21),
    'min_samples_leaf': range(1, 21)
},


'sklearn.neighbors.KNeighborsClassifier': {
    'n_neighbors': range(1, 101),
    'weights': ["uniform", "distance"],
    'p': [1, 2]
},
'sklearn.linear_model.LogisticRegression': {
    'penalty': ["l1", "l2"],
    'C': [1e-4, 1e-3, 1e-2, 1e-1, 0.5, 1., 5., 10., 15., 20., 25.],
    'dual': [True, False]
},
# Preprocesssors
'sklearn.preprocessing.Binarizer': {
    'threshold': np.arange(0.0, 1.01, 0.05)
},
'sklearn.cluster.FeatureAgglomeration': {
    'linkage': ['ward', 'complete', 'average'],
    'affinity': ['euclidean', 'l1', 'l2', 'manhattan', 'cosine']
},
'sklearn.preprocessing.MaxAbsScaler': {
},
'sklearn.preprocessing.MinMaxScaler': {
},
'sklearn.preprocessing.Normalizer': {
    'norm': ['l1', 'l2', 'max']
},
'sklearn.decomposition.PCA': {
    'svd_solver': ['randomized'],
```

```
        'iterated_power': range(1, 11)
    },
    'sklearn.kernel_approximation.RBFSampler': {
        'gamma': np.arange(0.0, 1.01, 0.05)
    },
    'sklearn.preprocessing.RobustScaler': {
    },
    'sklearn.preprocessing.StandardScaler': {
    },
    'tpot.builtins.ZeroCount': {
    },
    # Selectors
    'sklearn.feature_selection.SelectFwe': {
        'alpha': np.arange(0, 0.05, 0.001),
        'score_func': {
            'sklearn.feature_selection.f_classif': None
        }
    },
    'sklearn.feature_selection.SelectPercentile': {
        'percentile': range(1, 100),
        'score_func': {
            'sklearn.feature_selection.f_classif': None
        }
    },
    'sklearn.feature_selection.VarianceThreshold': {
        'threshold': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2]
    }
}
```

2. Dictionary Sparce TPOT

```
# -*- coding: utf-8 -*-

"""This file is part of the TPOT library.
TPOT was primarily developed at the University of Pennsylvania by:
    - Randal S. Olson (rso@randalolson.com)
    - Weixuan Fu (weixuanf@upenn.edu)
    - Daniel Angell (dpa34@drexel.edu)
    - and many more generous open source contributors
TPOT is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
```

```
published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.
TPOT is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.
You should have received a copy of the GNU Lesser General Public
License along with TPOT. If not, see <http://www.gnu.org/licenses/>.

"""
import numpy as np
classifier_config_sparse = {
    'tpot.builtins.OneHotEncoder': {
        'minimum_fraction': [0.05, 0.1, 0.15, 0.2, 0.25]
    },
    'sklearn.neighbors.KNeighborsClassifier': {
        'n_neighbors': range(1, 101),
        'weights': ["uniform", "distance"],
        'p': [1, 2]
    },
    'sklearn.ensemble.RandomForestClassifier': {
        'n_estimators': [100],
        'criterion': ["gini", "entropy"],
        'max_features': np.arange(0.05, 1.01, 0.05),
        'min_samples_split': range(2, 21),
        'min_samples_leaf':  range(1, 21),
        'bootstrap': [True, False]
    },
    'sklearn.feature_selection.SelectFwe': {
        'alpha': np.arange(0, 0.05, 0.001),
        'score_func': {
            'sklearn.feature_selection.f_classif': None
        }
    },
    'sklearn.feature_selection.SelectPercentile': {
        'percentile': range(1, 100),
        'score_func': {
            'sklearn.feature_selection.f_classif': None
        }
    },
```

```
'sklearn.feature_selection.VarianceThreshold': {
    'threshold': np.arange(0.05, 1.01, 0.05)
},
'sklearn.feature_selection.RFE': {
    'step': np.arange(0.05, 1.01, 0.05),
    'estimator': {
        'sklearn.ensemble.ExtraTreesClassifier': {
            'n_estimators': [100],
            'criterion': ['gini', 'entropy'],
            'max_features': np.arange(0.05, 1.01, 0.05)
        }
    }
},
'sklearn.feature_selection.SelectFromModel': {
    'threshold': np.arange(0, 1.01, 0.05),
    'estimator': {
        'sklearn.ensemble.ExtraTreesClassifier': {
            'n_estimators': [100],
            'criterion': ['gini', 'entropy'],
            'max_features': np.arange(0.05, 1.01, 0.05)
        }
    }
},
'sklearn.linear_model.LogisticRegression': {
    'penalty': ["l1", "l2"],
    'C': [1e-4, 1e-3, 1e-2, 1e-1, 0.5, 1., 5., 10., 15., 20., 25.],
    'dual': [True, False]
},
'sklearn.naive_bayes.BernoulliNB': {
    'alpha': [1e-3, 1e-2, 1e-1, 1., 10., 100.],
    'fit_prior': [True, False]
},
'sklearn.naive_bayes.MultinomialNB': {
    'alpha': [1e-3, 1e-2, 1e-1, 1., 10., 100.],
    'fit_prior': [True, False]
},
'sklearn.svm.LinearSVC': {
    'penalty': ["l1", "l2"],
    'loss': ["hinge", "squared_hinge"],
    'dual': [True, False],
```

```
        'tol': [1e-5, 1e-4, 1e-3, 1e-2, 1e-1],
        'C': [1e-4, 1e-3, 1e-2, 1e-1, 0.5, 1., 5., 10., 15., 20., 25.]
    },
    'xgboost.XGBClassifier': {
        'n_estimators': [100],
        'max_depth': range(1, 11),
        'learning_rate': [1e-3, 1e-2, 1e-1, 0.5, 1.],
        'subsample': np.arange(0.05, 1.01, 0.05),
        'min_child_weight': range(1, 21),
        'n_jobs': [1],
        'verbosity': [0]
    }
}
```

3. Dictionary Default TPOT

```
# -*- coding: utf-8 -*-
"""This file is part of the TPOT library.
TPOT was primarily developed at the University of Pennsylvania by:
    - Randal S. Olson (rso@randalolson.com)
    - Weixuan Fu (weixuanf@upenn.edu)
    - Daniel Angell (dpa34@drexel.edu)
    - and many more generous open source contributors
TPOT is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.
TPOT is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.
You should have received a copy of the GNU Lesser General Public
License along with TPOT. If not, see <http://www.gnu.org/licenses/>.
"""
import numpy as np
# Check the TPOT documentation for information on the structure of config dicts
classifier_config_dict = {
    # Classifiers
    'sklearn.naive_bayes.GaussianNB': {
    },
    'sklearn.naive_bayes.BernoulliNB': {
```

```
    'alpha': [1e-3, 1e-2, 1e-1, 1., 10., 100.],
    'fit_prior': [True, False]
},
'sklearn.naive_bayes.MultinomialNB': {
    'alpha': [1e-3, 1e-2, 1e-1, 1., 10., 100.],
    'fit_prior': [True, False]
},
'sklearn.tree.DecisionTreeClassifier': {
    'criterion': ["gini", "entropy"],
    'max_depth': range(1, 11),
    'min_samples_split': range(2, 21),
    'min_samples_leaf': range(1, 21)
},
'sklearn.ensemble.ExtraTreesClassifier': {
    'n_estimators': [100],
    'criterion': ["gini", "entropy"],
    'max_features': np.arange(0.05, 1.01, 0.05),
    'min_samples_split': range(2, 21),
    'min_samples_leaf': range(1, 21),
    'bootstrap': [True, False]
},
'sklearn.ensemble.RandomForestClassifier': {
    'n_estimators': [100],
    'criterion': ["gini", "entropy"],
    'max_features': np.arange(0.05, 1.01, 0.05),
    'min_samples_split': range(2, 21),
    'min_samples_leaf':  range(1, 21),
    'bootstrap': [True, False]
},
'sklearn.ensemble.GradientBoostingClassifier': {
    'n_estimators': [100],
    'learning_rate': [1e-3, 1e-2, 1e-1, 0.5, 1.],
    'max_depth': range(1, 11),
    'min_samples_split': range(2, 21),
    'min_samples_leaf': range(1, 21),
    'subsample': np.arange(0.05, 1.01, 0.05),
    'max_features': np.arange(0.05, 1.01, 0.05)
},
'sklearn.neighbors.KNeighborsClassifier': {
    'n_neighbors': range(1, 101),
```

```
    'weights': ["uniform", "distance"],
    'p': [1, 2]
},
'sklearn.svm.LinearSVC': {
    'penalty': ["l1", "l2"],
    'loss': ["hinge", "squared_hinge"],
    'dual': [True, False],
    'tol': [1e-5, 1e-4, 1e-3, 1e-2, 1e-1],
    'C': [1e-4, 1e-3, 1e-2, 1e-1, 0.5, 1., 5., 10., 15., 20., 25.]
},
'sklearn.linear_model.LogisticRegression': {
    'penalty': ["l1", "l2"],
    'C': [1e-4, 1e-3, 1e-2, 1e-1, 0.5, 1., 5., 10., 15., 20., 25.],
    'dual': [True, False]
},
'xgboost.XGBClassifier': {
    'n_estimators': [100],
    'max_depth': range(1, 11),
    'learning_rate': [1e-3, 1e-2, 1e-1, 0.5, 1.],
    'subsample': np.arange(0.05, 1.01, 0.05),
    'min_child_weight': range(1, 21),
    'n_jobs': [1],
    'verbosity': [0]
},
'sklearn.linear_model.SGDClassifier': {
    'loss': ['log', 'hinge', 'modified_huber', 'squared_hinge', 'perceptron'],
    'penalty': ['elasticnet'],
    'alpha': [0.0, 0.01, 0.001],
    'learning_rate': ['invscaling', 'constant'],
    'fit_intercept': [True, False],
    'l1_ratio': [0.25, 0.0, 1.0, 0.75, 0.5],
    'eta0': [0.1, 1.0, 0.01],
    'power_t': [0.5, 0.0, 1.0, 0.1, 100.0, 10.0, 50.0]
},
'sklearn.neural_network.MLPClassifier': {
    'alpha': [1e-4, 1e-3, 1e-2, 1e-1],
    'learning_rate_init': [1e-3, 1e-2, 1e-1, 0.5, 1.]
},
# Preprocesssors
'sklearn.preprocessing.Binarizer': {
```

```
    'threshold': np.arange(0.0, 1.01, 0.05)
},
'sklearn.decomposition.FastICA': {
    'tol': np.arange(0.0, 1.01, 0.05)
},
'sklearn.cluster.FeatureAgglomeration': {
    'linkage': ['ward', 'complete', 'average'],
    'affinity': ['euclidean', 'l1', 'l2', 'manhattan', 'cosine']
},
'sklearn.preprocessing.MaxAbsScaler': {
},
'sklearn.preprocessing.MinMaxScaler': {
},
'sklearn.preprocessing.Normalizer': {
    'norm': ['l1', 'l2', 'max']
},
'sklearn.kernel_approximation.Nystroem': {
    'kernel': ['rbf', 'cosine', 'chi2', 'laplacian', 'polynomial', 'poly', 'linear', 'additive_chi2', 'sigmoid'],
    'gamma': np.arange(0.0, 1.01, 0.05),
    'n_components': range(1, 11)
},
'sklearn.decomposition.PCA': {
    'svd_solver': ['randomized'],
    'iterated_power': range(1, 11)
},
'sklearn.preprocessing.PolynomialFeatures': {
    'degree': [2],
    'include_bias': [False],
    'interaction_only': [False]
},
'sklearn.kernel_approximation.RBFSampler': {
    'gamma': np.arange(0.0, 1.01, 0.05)
},
'sklearn.preprocessing.RobustScaler': {
},
'sklearn.preprocessing.StandardScaler': {
},

'tpot.builtins.ZeroCount': {
},
```

```
'tpot.builtins.OneHotEncoder': {
    'minimum_fraction': [0.05, 0.1, 0.15, 0.2, 0.25],
    'sparse': [False],
    'threshold': [10]
},
# Selectors
'sklearn.feature_selection.SelectFwe': {
    'alpha': np.arange(0, 0.05, 0.001),
    'score_func': {
        'sklearn.feature_selection.f_classif': None
    }
},
'sklearn.feature_selection.SelectPercentile': {
    'percentile': range(1, 100),
    'score_func': {
        'sklearn.feature_selection.f_classif': None
    }
},
'sklearn.feature_selection.VarianceThreshold': {
    'threshold': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2]
},
'sklearn.feature_selection.RFE': {
    'step': np.arange(0.05, 1.01, 0.05),
    'estimator': {
        'sklearn.ensemble.ExtraTreesClassifier': {
            'n_estimators': [100],
            'criterion': ['gini', 'entropy'],
            'max_features': np.arange(0.05, 1.01, 0.05)
        }
    }
},
'sklearn.feature_selection.SelectFromModel': {
    'threshold': np.arange(0, 1.01, 0.05),
    'estimator': {
        'sklearn.ensemble.ExtraTreesClassifier': {
            'n_estimators': [100],
            'criterion': ['gini', 'entropy'],
            'max_features': np.arange(0.05, 1.01, 0.05)
        }
    }
```

```
    }

}
```

4. Simulated annealing code in python

```
import numpy as np
import pandas as pd
from matplotlib import pyplot
import math
import random
from sklearn.base import BaseEstimator, is_classifier, is_regressor
from tpot.operator_utils import source_decode
from sklearn.metrics import f1_score
import timeit
import random
def model_extractor(_config_dict):
    '''extrae el nombre de los modelos del diccionario: ex sklearn.naive_bayes.GaussianNB'''
    models=[]
    # for every value in a dictionary
    for key in _config_dict.keys():
        import_str, op_str, op_obj=source_decode(key)
        # select just models
        if is_classifier(op_obj):
            models.append(key)
    return models

def get_parameters_model(_config_dict,term):
    'selecciona random parametros de un modelo de la libreria. term es el modelo'

    keys = []
    values = []
    len_parameters_model=len(_config_dict[term])

    if len_parameters_model==0:
        indv_parameters = {}
    else:
        for parameters_model in _config_dict[term]:
            random_param=np.random.choice(_config_dict[term][parameters_model])
            keys.append(parameters_model)
```

```
            values.append(random_param)
        indv_parameters = dict(zip(keys, values))
    return indv_parameters

def parameters_list(param_indv):
    '''Quita las comillas de las palabras al incio'''
    if len(param_indv) == 0:
        config_list_parameters=""
    else:
        config_list_parameters=[]
        for key in param_indv:
            if isinstance(param_indv[key], float)==True or isinstance(param_indv[key], (int, np.integer))==True or
param_indv[key]==True or param_indv[key]==False:
                format_list=key+" = "+str(param_indv[key])
            else:
                format_list=key+" = "+"\""+str(param_indv[key])+"\""


            config_list_parameters.append(format_list)
        config_list_parameters=str("{0}".format(','.join(map(str,config_list_parameters))))
    return config_list_parameters


def neighbor_score(_config_dict,term,data_execute,target_column):
    """"Calcula el score para el modelo ejecutado e imprime los paramtros con
    los datos que se incluyan para la binomial como tiene error con valores negativos toca construir condicion para
omitir """
    libraries_sklearn, model_name, op_obj=source_decode(term)
    listToStr=-888.0
    while listToStr == -888.0 or listToStr ==1.0 or listToStr ==0.0:
        global score
        score=[]
        parameters=get_parameters_model(_config_dict,term)
        model_parameters=parameters_list(parameters)
        text_01 ="""
try:
    from """+libraries_sklearn+""" import """+model_name+"""
    log_model ="""+model_name+"""("""+model_parameters+""")
    clf = log_model.fit(data_execute,target_column)
    y_pred = clf.predict(data_execute)
```

```
    f1 = f1_score(target_column, y_pred)
    score.append(f1)
except ValueError:
    if model_name=='MultinomialNB':
        score.append(-999)
    else:
        score.append(-888)"""
        exec(text_01)
        listToStr = float(' '.join(map(str, score)))
    return listToStr , term, model_parameters


def simmulated_anneling_professor(_config_dict,term,data_execute,target_column):
    '''Modelo con prints'''
    score_best, model_indv_best,param_indv_best=neighbor_score(_config_dict,term,data_execute,target_column)
    if score_best != -999.0:
    # best=current
        score_curr, model_indv_curr ,param_indv_curr= score_best, model_indv_best,param_indv_best
        print(term)
        print('CUREENT ',0,0,'----',param_indv_curr,'----',score_curr)
        print('BEST    ',0,0,'--->',param_indv_curr,'----',score_curr)
        i=1
        while i < 10:
            for l in range(1,5):
            #       # take a step

                score_candidate,
model_indv_candidate,param_indv_candidate=neighbor_score(_config_dict,term,data_execute,target_column)
                print('NEIGHBOR',i,'--',l,'----',param_indv_candidate,'----',score_candidate)

                #difference between candidate and current point evaluation
                diff=abs(score_candidate - score_curr)

                # calculate metropolis acceptance criterion
                metropolis = math.exp( -diff / (i+1) )

                rand_comp=random.uniform(0, 1)
                # check for new best solution
                if score_candidate > score_best and score_candidate !=1:
                    # store new best point
```

```
                score_best, model_indv_best,param_indv_best = score_candidate,
model_indv_candidate,param_indv_candidate
                # keep track of scores
                # report progress

            # store the new current point
                print('BEST    ',i,'--',l,'--->',param_indv_best,'----',score_best)
            elif  rand_comp < metropolis:
                # store the new current point
                score_curr, model_indv_curr ,param_indv_curr= score_candidate,
model_indv_candidate,param_indv_candidate
                print('RULE    ',i,'--',l,'----','RAND=',rand_comp ,'----',' exp( -diff / c )=',metropolis)
            else:
                print('STOP    ',i,'--',l,'----','RAND=',rand_comp ,'----',' exp( -diff / c )=',metropolis)
                break
        i += 1
        print('BEST    ','--->',term,param_indv_best,'----',score_best)
    else:
        pass

    return param_indv_best,score_best


def creation_dictionary(models,parameters):
    'Genera un diccionario con los parametros en el formato de la libreria'
    string = '.anneling'
    models_1 = [x + string for x in models]
    final_dic={}
    for i in range(len(models)):

        uniqueparameters = parameters[i].split(",")
        print(uniqueparameters)

        dic_temp={}
        for subp in uniqueparameters:

            key = subp.split("=")
            res = []

            for ele in key:
```

```
            j = ele.replace(' ', '')
            t = j.replace("'",'"')
            res.append(t)

        if len(res)==2:
            if res[1]=='True':
                dic_temp[res[0]] = [True]
            elif res[1]=='False':
                dic_temp[res[0]] = [False]
            else:
                try:
                    res_num=float(res[1])
                    dic_temp[res[0]] = [res_num]
                except:
                    dic_temp[res[0]] = [res[1]]

    if uniqueparameters==['']:
        final_dic
    else:
        final_dic[models_1[i]]=dic_temp

return final_dic

def main(target_dic,data_execute,target_column,config_dict_run):
    start_SA=timeit.default_timer()

    dic=target_dic
    modelos = model_extractor(dic)

    lista_modelos=[]
    parametres=[]
    score_model=[]
    for modelo in modelos:
        param_indv_best,score_best=simmulated_anneling_professor(dic,modelo,data_execute,target_column)

        if score_best != -999.0:
            lista_modelos.append(modelo)
            parametres.append(param_indv_best)
            score_model.append(score_best)
```

```
        dic_param_n_mod=creation_dictionary(lista_modelos,parametres)

    print(dic_param_n_mod)

    total_time_SA=timeit.default_timer()-start_SA
    print(total_time_SA)
    file_name=config_dict_run.replace(" ","_")

    #open text file
    text_file = open("/Users/lauraramos/Documents/AA_machine_execution_TEST/annealing/tpot-
master/AA_datasets/dic_"+file_name, "w")

    #write string to file
    text_file.write(str(dic_param_n_mod)+'    time seconds: '+str(total_time_SA))

    #close file
    text_file.close()

    return  dic_param_n_mod

if __name__ == '__main__':
    main(target_dic,data_execute,target_column,config_dict_run)
```

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

Universidade Nova de Lisboa