Performance and Competitiveness of Tree-Based Pipeline Optimization Tool

Marija Grjazniha

A thesis presented as a partial requirement for obtaining a master's degree in Data Science and Advanced Analytics

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

2022

Title: Performance and Competitiveness of Tree-Based Pipeline Optimization Tool:

Marija Grjazniha

MDSAA

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

Performance And Competitiveness of Tree-Based Pipeline Optimization Tool

by

Marija Grjazniha

The thesis presented as a partial requirement for obtaining a master's degree
in Data Science and Advanced Analytics with a specialization in Data Science

Advisor: 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 fully acknowledge the Rules of Conduct and Code of Honor from the NOVA Information Management School.

[illegible]

Antwerp. November 29, 2022

ABSTRACT

Automated machine learning (AutoML) is the process of automating the entire machine learning workflow when applied to real-world problems. AutoML can increase data science productivity while keeping the same performance and accuracy, allowing non-experts to use complex machine learning methods. Tree-based Pipeline Optimization Tool (TPOT) was one of the first AutoML methods created by data scientists and is targeted to optimize machine learning pipelines using genetic programming. While still under active development, TPOT is a very promising AutoML tool. This Thesis aims to explore the algorithm and analyse its performance using real word data. Results show that evolution-based optimization is at least as accurate as TPOT initialization. The effectiveness of genetic operators, however, depends on the nature of the test case.

KEYWORDS

Genetic Programming; Tree-Based Pipeline Optimization Tool; Machine Learning

TABLE OF CONTENTS

1. Introduction 1
2. Theoretical background 3
    2.1. Genetic programming 3
    2.2. TPOT 5
        2.2.1. TPOT operators 6
        2.2.2. TPOT pipeline 10
        2.2.3. Individual fitness evaluation 11
        2.2.4. Evolution algorithm 11
3. Methodology 13
    3.1. Approach 13
    3.2. Objectives 13
    3.3. Data 13
    3.4. Hypothesis testing 16
4. Experimental study 17
    4.1. Experimental settings 18
    4.2. Test cases 19
        4.2.1. Breast cancer 19
        4.2.2. Waveform 19
        4.2.3. Bioavailability 20
        4.2.4. Concrete 21
    4.3. Experimental results 21
        4.3.1. Initialization vs evolution 21
        4.3.2. Operator effectiveness 36
    4.4. Results and discussion 64
5. Conclusions and recommendations 69
6. Bibliography 71
7. Appendices 73

LIST OF FIGURES

Figure 2.1 Example of GP tree with four primitive nodes (white) and five terminal nodes (grey). 4

Figure 2.2 Example of GP crossover. Parent trees (a) and (b) produce offspring (c) and (d). .. 4

Figure 2.3 Example of GP mutation. 5

Figure 2.4 Example of a tree-based representation of TPOT pipeline. 10

Figure 4.1 Basic waveforms and 100 instances of each of the three classes. 20

Figure 4.2 Breast cancer training and testing accuracy distributions for each testing scenario (n=30). 21

Figure 4.3 Best internal CV score distribution by generation for three scenarios. Breast cancer dataset. 22

Figure 4.4 Number of times classifiers appear in the root of TPOT optimal pipelines for the breast cancer dataset. 24

Figure 4.5 Test score distribution for each classifier of TPOT optimal pipelines for the breast cancer dataset. 25

Figure 4.6 Waveform training and testing accuracy distributions for each testing scenario (n=30). 25

Figure 4.7 Best internal CV score distribution by generation for three scenarios. Waveform dataset. 26

Figure 4.8 Number of times classifiers appear in the root of TPOT optimal pipelines for the Waveform dataset. 27

Figure 4.9 Test score distribution for each classifier of TPOT optimal pipelines for the Waveform dataset. 28

Figure 4.10 Bioavailability training and testing negative root mean squared error distributions for each testing scenario (n=30). 29

Figure 4.11 Best internal CV score distribution by generation for three scenarios. Bioavailability dataset. 30

Figure 4.12 Number of times regressors appear in the root of TPOT optimal pipelines for the bioavailability dataset. 31

Figure 4.13 Test score distribution for each regressor of TPOT optimal pipelines for the bioavailability dataset. 31

Figure 4.14 Concrete training and testing negative root mean squared error distributions for each testing scenario (n=30). 31

Figure 4.15 Best internal CV score distribution by generation for three scenarios. Concrete dataset. 32

Figure 4.16 Number of times regressors appear in the root of TPOT optimal pipelines for the concrete dataset. 33

Figure 4.17 Test score distribution for each regressor of TPOT optimal pipelines for the concrete dataset. 34
Figure 4.18 Selected individual size distribution by generation. Breast cancer. 37
Figure 4.19 Amount and performance of operations by genetic operators. Breast cancer. 38
Figure 4.20 Distribution of median accuracies by the generation before and after genetic operation. Breast cancer. 38
Figure 4.21 Amount and performance of mutations by pipeline generation. Breast cancer. .. 39
Figure 4.22 Amount and performance of mutations by pipeline size increase. Breast cancer. 39
Figure 4.23 Amount and performance of mutation operations by type. Breast cancer. 40
Figure 4.24 Amount and performance of mutation operations by the added object. Breast cancer. 40
Figure 4.25 Amount and performance of mutation operations by the changed object (short). Breast cancer. 40
Figure 4.26 Amount and performance of mutation operations by the removed object. Breast cancer. 41
Figure 4.27 Amount and performance of mutation operations by the level of operation. Breast cancer 41
Figure 4.28 Amount and performance of crossover operations by generation. Breast cancer. 41
Figure 4.29 Amount and performance of crossover operations by pipeline size increase. Breast cancer. 42
Figure 4.30 Amount and performance of crossover operations by crossover type. Breast cancer. 42
Figure 4.31 Amount and performance of crossover operations by crossover base. Breast cancer. 42
Figure 4.32 Amount and performance of crossover operations by crossover base and type. Breast cancer. 43
Figure 4.33 Amount and performance of crossover operations by the level of operation. Breast cancer. 43
Figure 4.34 Selected individual size distribution by generation. Waveform. 44
Figure 4.35 Amount and performance of operations by genetic operators. Waveform 44
Figure 4.36 Distribution of median accuracies by the generation before and after genetic operation. Waveform. 45
Figure 4.37 Amount and performance of mutations by pipeline generation. Waveform. 45
Figure 4.38 Amount and performance of mutations by pipeline size increase. Waveform. 46
Figure 4.39 Amount and performance of mutation operations by type. Waveform. 46
Figure 4.40 Amount and performance of mutation operations by the added object. Waveform. 46
Figure 4.41 Amount and performance of mutation operations by the changed object (short). Waveform. 47

Figure 4.42 Amount and performance of mutation operations by the removed object. Waveform. 47
Figure 4.43 Amount and performance of mutation operations by the level of operation. Waveform. 48
Figure 4.44 Amount and performance of crossover operations by generation. Waveform. 48
Figure 4.45 Amount and performance of crossover operations by pipeline size increase. Waveform. 48
Figure 4.46 Amount and performance of crossover operations by crossover type. Waveform. 49
Figure 4.47 Amount and performance of crossover operations by crossover base. Waveform. 49
Figure 4.48 Amount and performance of crossover operations by crossover base and type. Waveform. 50
Figure 4.49 Amount and performance of crossover operations by the level of operation. Waveform. 50
Figure 4.50 Selected individual size distribution by generation. Bioavailability. 51
Figure 4.51 Amount and performance of operations by genetic operators. Bioavailability. 51
Figure 4.52 Distribution of median accuracies by the generation before and after genetic operation. Waveform. 51
Figure 4.53 Amount and performance of mutations by generation. Bioavailability 52
Figure 4.54 Amount and performance of mutations by pipeline size increase. Bioavailability. 52
Figure 4.55 Amount and performance of mutation operations by type. Bioavailability. 53
Figure 4.56 Amount and performance of mutation operations by the added object. Bioavailability. 53
Figure 4.57 Amount and performance of mutation operations by the changed object (short). Bioavailability. 54
Figure 4.58 Amount and performance of mutation operations by the removed object. Bioavailability. 54
Figure 4.59 Amount and performance of mutation operations by the level of operation. Bioavailability. 55
Figure 4.60 Amount and performance of crossover operations by generation. Bioavailability. 55
Figure 4.61 Amount and performance of crossover operations by pipeline size increase. Bioavailability. 55
Figure 4.62 Amount and performance of crossover operations by crossover type. Bioavailability. 56
Figure 4.63 Amount and performance of crossover operations by crossover base. Bioavailability. 56

Figure 4.64 Amount and performance of crossover operations by crossover base and type. Bioavailability. 57
Figure 4.65 Amount and performance of crossover operations by the level of operation. Bioavailability. 57
Figure 4.66 Selected individual size distribution by generation. Concrete. 58
Figure 4.67 Amount and performance of operations by genetic operators. Concrete. 58
Figure 4.68 Distribution of median accuracies by the generation before and after genetic operation. Concrete. 58
Figure 4.69 Amount and performance of mutations by generation. Concrete. 59
Figure 4.70 Amount and performance of mutations by pipeline size increase. Concrete. 59
Figure 4.71 Amount and performance of mutation operations by type. Concrete. 60
Figure 4.72 Amount and performance of mutation operations by the added object. Concrete. 60
Figure 4.73 Amount and performance of mutation operations by the changed object (short). Concrete. 61
Figure 4.74 Amount and performance of mutation operations by the removed object. Concrete. 61
Figure 4.75 Amount and performance of mutation operations by the level of operation. Concrete. 61
Figure 4.76 Amount and performance of crossover operations by generation. Concrete. 62
Figure 4.77 Amount and performance of crossover operations by pipeline size increase. Concrete. 62
Figure 4.78 Amount and performance of crossover operations by crossover type. Concrete. 63
Figure 4.79 Amount and performance of crossover operations by crossover base. Concrete. 63
Figure 4.80 Amount and performance of crossover operations by crossover base and type. Concrete. 63
Figure 4.81 Amount and performance of crossover operations by the level of operation. Concrete. 64

LIST OF TABLES

Table 2.1 *Number of classifier hyperparameters (H), hyperparameter combinations (HC) and classifier appearance in configurations.* 7
Table 2.2 *Number of regressor hyperparameters (H), hyperparameter combinations (HC) and regressor appearance in configurations.* 7
Table 2.3 *Number of preprocessor hyperparameters (H), hyperparameter combinations (HC) and preprocessor appearance in configurations for classification (•) and regression (∗).* 8
Table 2.4 *Number of selector hyperparameters (H), hyperparameter combinations (HC) and selector appearance in c configurations for classification (•) and regression (∗).* 9
Table 2.5 *Number of feature constructor hyperparameters (H), hyperparameter combinations (HC) and their appearance configurations for classification (•) and regression (∗).* 9
Table 4.1 *Experimental scenario settings* 18
Table 4.2 *Test case descriptions* 19
Table 4.3 *Example of a drug expressed in linear notation (SMILES) and as a set of molecular descriptors.* 20
Table 4.4 *Breast cancer test accuracy mean, standard deviation and Mann-Whitney U p-values.* 22
Table 4.5 *Waveform test accuracy mean ±standard deviation and T-test p-values.* 26
Table 4.6 *Bioavailability test RMSE mean ±standard deviation and T-test p-values.* 29
Table 4.7 *Concrete test RMSE mean ±standard deviation and T-test p-values.* 32
Table 4.8 *Summary of the first experiment results: mean accuracy/negative RMSE and standard deviation by scenario.* 34
Table 4.9 *Summary of internal CV score comparison by generation.* 35
Table 4.10 *Number of pipelines when a certain number of operators is used.* 35
Table 4.11 *Number of operations by the test case.* 37

LIST OF ABBREVIATIONS AND ACRONYMS

BC	Breast Cancer
BIO	Bioavailability
CON	Concrete
CV	Cross-Validation
DL	Deep Learning
GP	Genetic Programming
ML	Machine Learning
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
NB	Naïve Bayes
RFE	Recursive Feature Elimination
RMSE	Root Mean Squared Error
SGD	Stochastic Gradient Descent
SVC	Support Vector Classification
TPOT	Tree-Based Pipeline Optimization Tool
WF	Waveform
XGB	Extreme Gradient Boosting

1. INTRODUCTION

Machine learning (ML) can still be challenging when implementing existing models to work well on your new dataset. The development of an ML model requires significant human expertise. It requires being aware of available algorithms and models and their constraints. It is necessary to use knowledge and experience-based intuition to determine the best ML architecture for each new application. However, even an expert has to iteratively implement alternative pipelines to find a good enough solution, which alone exposes an opportunity for automation. Besides the repetitive nature of many ML processes, the high demand for ML expertise and the shortage of ML specialists have inspired the creation of tools that reduce the manual efforts of developing different ML pipelines.

Automated machine learning (AutoML) is the process of automating one or several ML workflow processes like model training, hyperparameter optimization and feature engineering. AutoML allows non-experts to use complex ML methods, increasing the productivity of data scientists, ML engineers, and ML researchers while keeping the same performance and accuracy. AutoML methods vary from hyperparameter optimization to meta-learning and neural architecture search (Hutter et al., 2019). AutoML is usually built on one or several ML frameworks and specializes in automating standard ML (like Auto-Sklearn) or deep learning (DL) models (like AutoKeras). Recent works compare different AutoML tools based on their performance and features (Balaji & Allen, 2018; Truong et al., 2019).

Tree-based Pipeline Optimization Tool (TPOT) was one of the first AutoML methods created by data scientists and is targeted to optimize ML pipelines using genetic programming, i.e., evolving pipelines by means of a process that resembles the natural evolution of species. While still under active development, TPOT is a very promising AutoML tool. TPOT is not limited to model selection and parameter optimization but automatically constructs and optimizes tree-based ML pipelines, including data and feature preprocessing and selection and new feature construction. Although it has already proved its potential against standard ML analyses (Olson et al., 2016ab) and other AutoML tools (Balaji & Allen, 2018), scientists and ML practitioners may still not understand how it actually works.

This Thesis aims to explore the algorithm and analyse its performance using four test cases. TPOT exploration will show how pipelines are constructed and how genetic programming operators optimize the solution. The further study will be split into two independent experiments. Both of them are performed on the same datasets. The first experiment will determine if evolution benefits pipeline optimization by comparing the TPOT-optimized solutions under different scenarios: one scenario that initialises 250 pipelines and others that evolve pipelines for 5 and 10 generations. The second experiment will uncover how mutation and crossover determine optimization effectiveness. The performance of each genetic operator will be measured by the share of successful operations and analysed by its type, level of operation and generation.

The following chapter provides a theoretical background for genetic programming—the foundation of TPOT—and an in-depth overview of the TPOT optimization procedure. Chapter 3 describes the main components of further analysis: the objectives, data, and approaches to achieve stated objectives. Chapter 4 discusses experimental settings, test cases and results of the experiments. The thesis is concluded in chapter 5. The last chapter also includes recommendations for future works.

2. THEORETICAL BACKGROUND

2.1. Genetic programming

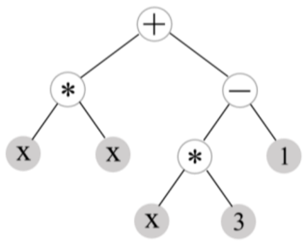
First introduced and popularized by Koza (1992, 1994, 2010), genetic programming (GP) is a subfield of evolutionary algorithms. Like other evolutionary algorithms, GP is inspired by the process of natural selection and applies the principles of Darwin's evolution theory. GP often uses tree-based data structures to represent the computer programs instead of the list structures typical for genetic algorithms. It applies biologically inspired operators: mutation, crossover, and selection, to solve optimization problems. A mutation is an innovation operator as it introduces new genetic material variations inside the population. Crossover is a conservation operator as it keeps existing genetic material and passes it through generations.

GP evolution process follows the steps described in Algorithm 2.1. Usually, the termination criterion is the number of generations the population is set to evolve.

Algorithm 2.1. *Genetic programming evolution algorithm.*

```
1. Initialization: generate a population of computer programs (individuals).
2. Until termination criteria are not met, perform the following steps:
   2.1. Evaluate the performance of each individual and assign it a fitness value.
   2.2. Apply the following steps to create a new population:
        2.2.1. Selection: select a set of individuals probabilistically based on their fitness values.
        2.2.2. Reproduction: copy some of the selected individuals to the new population.
        2.2.3. Crossover: recombine randomly chosen parts of two selected individuals to create new ones.
        2.2.4. Mutation: substitute randomly chosen parts of a selected individual with randomly generated new ones to create a new individual.
3. Return the single best computer program existing in any generation. This result of the run
   of GP may be a solution or an approximate solution to the problem.
```

Any GP individual represented as a tree is defined by a set of primitive functions and terminals. Functions may include mathematical functions, boolean, conditional and iterative operations. Terminals typically consist of problem variables and constants. Trees are generated randomly and, therefore, can vary in shape and size. During the evolution process, trees may consist of any combination of primitives and terminals while that combination can express a solution to the problem (sufficiency requirement), and each function can handle any combination of inputs it can encounter (closure requirement). Figure 2.1 illustrates an example of a three with arithmetical operators as primitives.



x^2+3x-1

Figure 2.1 Example of GP tree with four primitive nodes (white) and five terminal nodes (grey).

The selection algorithm decides which individuals will be replicated, mated, and mutated. That decision is made based on individuals' fitness values. The fitness value is a numeric score assigned by a fitness function and reflects how well a program (individual) fits its purpose. The fitter the individual, the higher are chances of surviving or passing its genes to the next generation. The likelihood of being selected can be proportional to the fitness value (roulette wheel selection), based on the rank individual takes according to their fitness value (ranking selection) or based on a competition between a randomly chosen subset of individuals (tournament selection).

The crossover operator creates variations in the population by producing offspring that take parts taken from each parent. Figure 2.2 illustrates an example of a crossover between trees (a) and (b) that produce offspring: trees (c) and (d). Trees exchange subtrees below randomly chosen crossover points to produce the offspring who inherit a part of each parent tree. The crossover points of parent trees are marked with a dot in Figure 2.2. Illustrated crossover is called one-point crossover and is the most basic implementation.

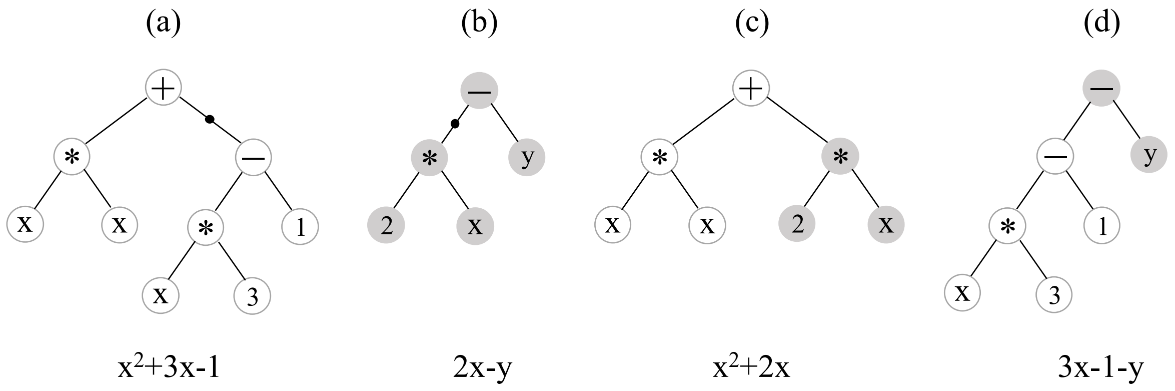


Figure 2.2 Example of GP crossover. Parent trees (a) and (b) produce offspring (c) and (d).

GP mutation randomly selects a mutation point and swaps a subtree located under that point with a randomly generated subtree. It requires only one parent. Figure 2.3 shows the mutation of a tree (a) whose subtree was replaced with a randomly generated tree (e). A mutation point is illustrated as a dot on edge.

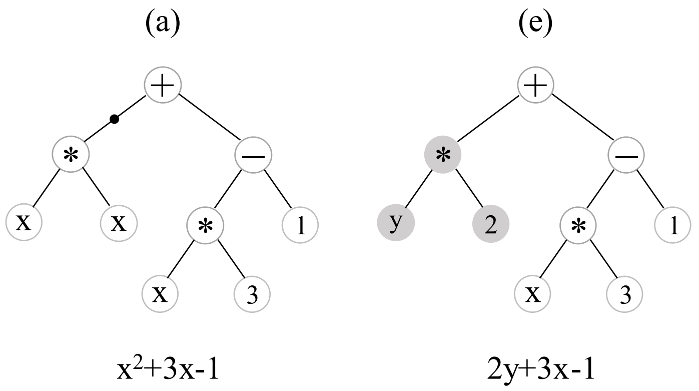


Figure 2.3 Example of GP mutation.

More advanced mutation and crossover algorithms can control the depth of the newly created trees, for example, by limiting the size of randomly created trees during mutation. In the case of crossover, different probabilities to be chosen as a crossover point can be assigned to parent tree nodes.

2.2. TPOT

Tree-based Pipeline Optimization Tool (TPOT) is an open-source library for Python that optimizes ML pipelines using genetic programming (GP). GP primitives in TPOT are ML pipeline operators, i.e., existing ML algorithms like random forest regressor or standard scaler from Python packages like scikit-learn (Pedregosa et al., 2011). TPOT uses GP procedures to discover the best-performing pipeline for a given dataset. In order to automatically generate and optimize ML pipelines via GP algorithms, TPOT utilizes the Python package DEAP (Fortin et al., 2012). TPOT was first introduced by Olson et el. (2016).

TPOT automates the part of the pipeline that includes feature creation, preprocessing and selection, model selection and parameter optimization. Therefore, input data must be valid, accurate, consistent, and complete. Median values impute all missing values.

TPOT has frequently outperformed standard ML analyses (Olson et al., 2016ab, Le et al., 2020). It was also proved to perform better than other automated ML frameworks in solving regression problems and is second after auto-sklearn in classification problems (Balaji & Allen, 2018). Other distinctive features of TPOT include the possibility to set a maximum execution time, pausing and resuming the optimization process, and exporting a model as Python code.

Although TPOT search space can be adjusted through a custom configuration file, the following chapter describes the operation of the default configurations. The TPOT process description is based on studying the TPOT 0.11.7 code (https://github.com/EpistasisLab/tpot) and official documentation (http://epistasislab.github.io/tpot/).

2.2.1. TPOT operators

Depending on the selected configuration, TPOT applies a different set of operators. TPOT operators include classifiers, regressors, preprocessors, selectors and a special type of operator that combines two copies of a dataset. Besides default configurations for classification and regression, TPOT configurations include (1) cuML, which uses classifiers and regressors from cuml library, (2) light, (3) MDR, which uses feature constructors from MDR library, (4) NN, which uses PyTorch neural networks, and (5) sparse, which supports sparse matrices and uses One Hot Encoder. While classification has all five additional configurations, regression has all but "NN". As will be seen later, the same operator can have different hyperparameter options under different configurations.

Classifiers are supervised ML algorithms that learn to predict a class label of a given data point. TPOT uses a set of 17 classifiers. Twelve of those are algorithms from the sklearn library: three ensemble algorithms, two linear models, three naïve Bayes models, and one of each: neural networks, neighbour, tree, and support vector algorithms. Cuml is an ML library that allows running models on GPU effortlessly, which improves large dataset training time 10-50 times compared to CPU-based equivalents like sklearn (Nolet et al., 2020). Two cuml algorithms are used under cuML configuration. Extreme Gradient Boosting (XGB) classifier comes from the xgboost library. Two algorithms: Logistic Regression and Multilayer Perceptron, are based on PyTorch but adjusted for TPOT.

ML algorithms typically have multiple hyperparameters. Like grid search, TPOT search space is limited to hyperparameter values specified in configuration settings. Hyperparameter combinations are calculated as a multiplication of all hyperparameter options. For example, K Neighbors Classifier (sklearn) has three hyperparameters specified in TPOT configurations: (1) number of neighbours, which can be one of 100 values ranging from 1 to 100, (2) weight function, which can be either "uniform" or "distance", and (3) a power parameter for the Minkowski metric: 1 for Manhattan distance and 2 for Euclidean distance. Therefore, K Neighbors Classifier (sklearn) model can have 400 different hyperparameter combinations. Unlike other classifiers, Extreme Gradient Boosting (XGB) has a different number of hyperparameters and, thus, a different number of hyperparameter combinations, depending on the configuration: nine hyperparameters in the case of cuML and 7 in others. Table 2.1 summarises classifier algorithms, their appearance in TPOT configurations, source libraries, and the number of hyperparameters and their value combinations.

Table 2.1 *Number of classifier hyperparameters (H), hyperparameter combinations (HC) and classifier appearance in configurations.*

Classifier	**Library**	**H**	**HC**	**Applied in configuration**					
				default	**cuML**	**light**	**MDR**	**NN**	**sparse**
Logistic Regression	cuml	2	33		●				
K Neighbors Classifier	cuml	2	100		●				
Gradient Boosting Classifier	sklearn	7	7600000	●				●	
Extra Trees Classifier	sklearn	6	30400	●				●	
Random Forest Classifier	sklearn	6	30400	●				●	●
Logistic Regression	sklearn	3	44	●		●	●	●	●
SGD Classifier	sklearn	8	6300	●				●	
Bernoulli NB	sklearn	2	12	●		●		●	●
Gaussian NB	sklearn	0	1	●		●		●	
Multinomial NB	sklearn	2	12	●		●		●	●
K Neighbors Classifier	sklearn	3	400	●		●		●	●
MLP Classifier	sklearn	2	20	●				●	
Linear SVC	sklearn	5	440	●				●	●
Decision Tree Classifier	sklearn	4	7600	●		●		●	
XGB Classifier	xgboost	7	20000	●				●	●
		9	22400		●				
Pytorch LR Classifier	tpot/pytorch	4	240					●	
Pytorch MLP Classifier	tpot/pytorch	4	240					●	

Regressors are ML algorithms that investigate relationships between variables and thus learn to predict continuous outcomes. Like in the case of classifiers, Extreme Gradient Boosting (XGB) regressor has a more extensive set of hyperparameters under cuML configuration than other configurations. TPOT uses a set of 16 regressors, 11 of which are sklearn models: 4 ensembles, four linear models, one neighbour model, one tree model, and one support vector model. Four algorithms are from the cuml library, and one from xgboost. None of the neural algorithms is currently used to solve regression problems. Table 2.2 summarises regressor algorithms used in TPOT, its source library, the number of hyperparameters and hyperparameter combinations used in TPOT and regressor appearance in configurations.

Table 2.2 *Number of regressor hyperparameters (H), hyperparameter combinations (HC) and regressor appearance in configurations.*

Regressor	**Library**	**H**	**HC**	**Applied in configuration**				
				default	**cuML**	**light**	**MDR**	**sparse**
Elastic Net	cuml	2	105		∗			
Lasso	cuml	1	2		∗			
Ridge	cuml	0	1		∗			
K Neighbors Regressor	cuml	2	100		∗			
Ada Boost Regressor	sklearn	3	15	∗				
Extra Trees Regressor	sklearn	5	15200	∗				

Regressor	Library	H	HC	Applied in configuration				
				default	cuML	light	MDR	sparse
Gradient Boosting Regressor	sklearn	9	182400000	∗				
Random Forest Regressor	sklearn	5	15200	∗				∗
Elastic Net CV	sklearn	2	105	∗		∗	∗	∗
Lasso Lars CV	sklearn	1	2	∗		∗		
Ridge CV	sklearn	0	1	∗		∗		∗
SGD Regressor	sklearn	8	3780	∗				
K Neighbors Regressor	sklearn	3	400	∗		∗		∗
Linear SVR	sklearn	5	1100	∗		∗		∗
Decision Tree Regressor	sklearn	3	3800	∗		∗		
XGB Regressor	xgboost	8	20000	∗				∗
		10	22400		∗			

Preprocessing operators transform the dataset in some way. For example, scaling variables might neglect the effect of varying magnitudes on distance-based and variance-sensitive models. Most of the preprocessors are from the sklearn library, which includes seven value preprocessors like normalizer, binarizer and scalers, two kernel approximations (Nystroem, RBF Sampler), two decomposition algorithms like independent component analysis (ICA) and principal component analysis (PCA), feature agglomeration algorithm that recursively merges pair of clusters of features. Table 2.3 summarises preprocessor algorithms used in TPOT, its source library, the number of hyperparameters and hyperparameter combinations used in TPOT and preprocessor appearance in configurations. TPOT does not use any preprocessors under MDR configuration and only uses One Hot Encoder under sparse configuration.

Table 2.3 *Number of preprocessor hyperparameters (H), hyperparameter combinations (HC) and preprocessor appearance in configurations for classification (●) and regression (∗).*

Preprocessor	Library	H	HC	Applied in configuration					
				Default	cuML	light	MDR	NN	sparse
Binarizer	sklearn	1	21	●∗	●∗	●∗		●	
Fast ICA	sklearn	1	21	●∗	●∗			●	
Feature Agglomeration	sklearn	2	15	●∗	●∗	●∗		●	
Max Abs Scaler	sklearn	0	1	●∗	●∗	●∗		●	
Min Max Scaler	sklearn	0	1	●∗	●∗	●∗		●	
Normalizer	sklearn	1	3	●∗	●∗	●∗		●	
Nystroem	sklearn	3	1890	●∗	●∗	∗		●	
PCA	sklearn	2	10	●∗	●∗	●∗		●	
Polynomial Features	sklearn	3	1	●∗				●	
RBF Sampler	sklearn	1	21	●∗	●∗	●∗		●	
Robust Scaler	sklearn	0	1	●∗	●∗	●∗		●	
Standard Scaler	sklearn	0	1	●∗	●∗	●∗		●	
Zero Count	tpot	3	1	●∗	●∗	●∗		●	
One Hot Encoder	tpot	3	5	●∗	●∗			●	●∗

Selecting a relevant subset of variables can reduce the training time and overfitting and improve accuracy and model interpretability. TPOT uses scikit-learn (sklearn) selection algorithms in most configurations, including the default one. Those algorithms apply common approaches for dimensionality reduction: removing features with low variance (Variance Threshold), univariate feature selection (Select Percentile, Select FWE), recursive feature elimination (RFE) and selection from a model based on importance weights. Under MDR configuration, TPOT uses scikit-rebate (skrebate) Relief-based feature selection algorithms. Table 2.4 summarises selector algorithms used in TPOT, its source library, the number of hyperparameters and hyperparameter combinations used in TPOT and selector appearance in configurations. The “Select from model” algorithm has a higher number of hyperparameters and hyperparameter combinations under default and sparse classification configurations than regression configurations. Recursive feature elimination (RFE) can only be used to solve classification problems.

Table 2.4 *Number of selector hyperparameters (H), hyperparameter combinations (HC) and selector appearance in c configurations for classification (•) and regression (∗).*

Selector	Library	H	HC	Applied in configuration					
				Default	cuML	light	MDR	NN	sparse
Select Fwe	sklearn	2	50	•∗	•∗	•∗		•	•∗
Select Percentile	sklearn	2	99	•∗	•∗	•∗		•	•∗
Variance Threshold	sklearn	1	8	•∗	•∗	•∗		•	•∗
RFE	sklearn	4	800	•				•	•
Select From Model	sklearn	4	840	•				•	•
		3	420	∗					∗
ReliefF	skrebate	2	30				•∗		
SURF	skrebate	1	5				•∗		
SURFstar	skrebate	1	5				•∗		
MultiSURF	skrebate	1	5				•∗		

Multifactor dimensionality reduction (MDR) algorithms convert two or more variables to a single attribute, which changes the data representation space (Sohn et al., 2017). Two algorithms: MDR and continuous MDR, are used to engineer new features for classification and regression problems, respectively. Feature construction operator is available only under MDR configuration. Table 2.5 summarises details about feature constructor algorithms used in TPOT.

Table 2.5 *Number of feature constructor hyperparameters (H), hyperparameter combinations (HC) and their appearance configurations for classification (•) and regression (∗).*

Feature Constructor	Library	H	HC	Applied in configuration
				MDR
MDR	mdr	2	4	•
Continuous MDR	mdr	2	4	∗

Default configuration covers a sufficient number of classifiers, regressors, preprocessors and selectors. CuML configuration has much fewer options in classifiers and regressors, however, included algorithms are more efficient than default ones. Light configuration algorithms are limited only to computationally efficient ones. MDR is the only configuration that allows feature construction and relief-based feature selection. It uses no preprocessors and single classifier and regressor options: Logistic regression and Elastic Net CV, respectively. NN is an extended version of the default configuration that allows running neural network algorithms. Currently, it can only be used for classification. Running TPOT on sparse data requires using a separate configuration with a limited number of classifiers and regressors and the only preprocessor—One Hot Encoder. Additionally, all configurations have a special operator that combines two copies of input data. However, this is not included in the total number of operators in individual fitness calculations.

2.2.2. TPOT pipeline

Operators discussed in the previous chapter are sequentially combined to create a pipeline. Tree representation of TPOT pipelines allows performing genetic operations: mutation and crossover. Each pipeline has at least one primitive node, either a classifier or a regressor, and a set of terminals: input matrix and hyperparameters. The initially generated population consists only of trees with at most two operators, but with a custom template, it is possible to initialize pipelines in different sizes. Pipelines can grow or shrink with generations.

KNeighborsClassifier(input_matrix, n_neighbors=10, p=2, weights=distance) (2.1)

GradientBoostingClassifier(CombineDFs(Nystroem(SelectPercentile(input_matrix, percentile=33), gamma=0.35000000000000003, kernel=linear, n_components=5), input_matrix), learning_rate=1.0, max_depth=6, max_features=0.4, min_samples_leaf=13, min_samples_split=13, n_estimators=100, subsample=0.8) (2.2)

A pipeline can be expressed as a string. A simple example is Pipeline 2.1: a classification model with one primitive node and three hyperparameters. Pipeline 2.2 is more complex and much less intuitive for a human. Figure 2.4 visualizes Pipeline 2.2 as a horizontal tree with a root node on the left. Primitives and input matrices are highlighted in red and blue, respectively. Figure 2.4 shows Pipeline 2.2 consists of a classifier, a preprocessor, and a selector:

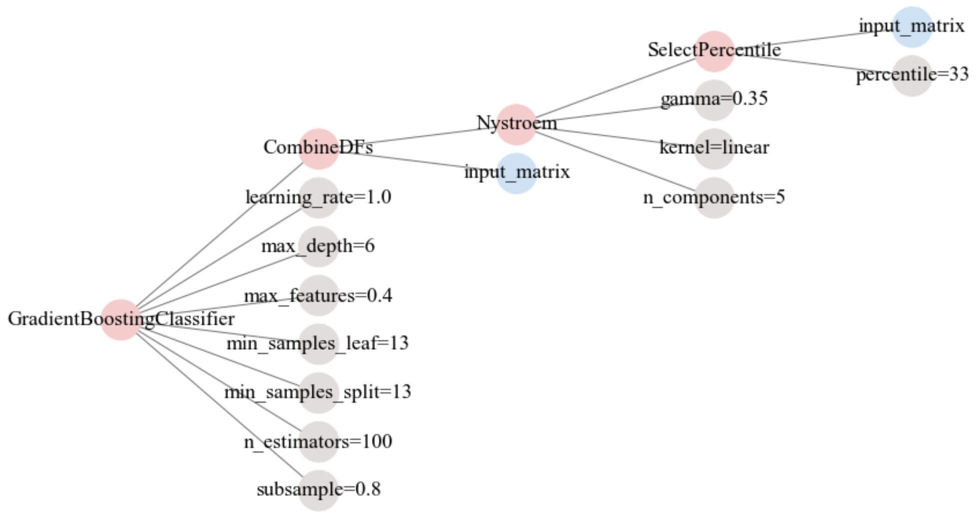


Figure 2.4 Example of a tree-based representation of TPOT pipeline.

Gradient Boosting classifiers combine two data copies as input, one of which is row data. At the same time, the other is a preprocessed set of selected variables.

The optimal pipeline discovered by TPOT is then converted into Python code. The code generated by TPOT only requires specifying the input data source. Otherwise, it is ready to run. The code imports all necessary functions, and the data extracts features and target values, splits data into training and testing sets, creates a pipeline using a pipeline constructor from the scikit-learn library, trains the pipeline and predicts the result. An example of a python code generated by TPOT can be found in Appendix 1.

2.2.3. Individual fitness evaluation

TPOT optimizes the accuracy achieved by the pipeline while accounting for its complexity, making an optimization process a two-objective problem. By default, model performance is evaluated in terms of accuracy (Equation 2.1) for classification problems and negative mean squared error (Equation 2.2) for regression problems. Thus, model performance is always a maximization problem. Other classification and regression score options vary from average precision to ROC AUC score and from negative median absolute error to R^2. K-fold cross-validation is used to get a more accurate performance evaluation. TPOT uses a 5-fold CV by default. Sacrifice for evaluation accuracy is increased evaluation time.

$$accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{2.1}$$

$$-MSE = -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.2}$$

Pipeline complexity is evaluated by a number of operators, which include classifiers, regressors, preprocessors, and selectors. Operators that combine datasets do not contribute to the number of operators. The complexity objective is the minimization problem. TPOT uses the Pareto front to evaluate individual fitness by finding a trade-off between two objectives.

2.2.4. Evolution algorithm

Like any other genetic programming algorithm, TPOT evolves a population over a given number of populations, and the fittest individual from the last generation is selected as an optimal solution. However, every new generation is selected from a combination of previous population individuals and its offspring. Offspring is generated by mutation, crossover, or reproduction with given probabilities. Individuals subject to mutation and crossover are chosen randomly regardless of their fitness values. Algorithm 2.2 briefly summarises the process of discovering the TPOT optimal pipeline.

Algorithm 2.2. *TPOT evolution algorithm.*

```
0. Set a number of individuals in the population: population_size (default 100), number of
   generations to evolve the population: generation (default 100), number of offspring to
   produce in each generation: lambda (default equal to population_size), probability rate
   for mutation mutation_rate (default 0.9), probability rate for crossover crossover_rate
   (default 0.1).
1. Initialize a population set of population_size individuals.
2. Repeat generation times:
  2.1. Repeat lambda times:
      2.1.1. Randomly choose one of three options:
          2.1.1.1. With probability mutation_rate, randomly select an individual from the
              population, mutate the individual, and add a mutated individual to the
              offspring set.
          2.1.1.2. With probability crossover_rate, randomly select a pair of individuals
              from a set of eligible crossover pairs, mate them, and add the first outputting
              individual to the offspring set.
          2.1.1.3. With probability 1 - mutation_rate - crossover_rate, randomly select an
              individual from the population, and add it to the offspring set.
  2.2. Merge the population set with the offspring set.
  2.3. Perform a selection of population_size individuals based on individual fitness values
     to create a new population set.
3. Select the fittest individual.
```

The selection operator used by TPOT is NSGA-II from the DEAP library (known as selNSGA2 in DEAP), which takes *k* individuals from a set of individuals larger than *k* and selects each individual at most once. NSGA-II or Non-dominated Sorting GA-II is a fast multi-objective evolutionary algorithm that uses non-dominated sorting and sharing. It has a better convergence to the near-Pareto-optimal front than older (Deb, 2002).

The crossover operator is a one-point crossover based on a similar algorithm from the DEAP library (cxOnePoint). First, a set of individual pairs eligible for crossover is created. Individuals are eligible for crossover if they are not the same and share at least one primitive. Then crossover is performed on a randomly chosen pair of eligible individuals. If no pairs are eligible for crossover, the mutation is performed instead.

The mutation operator randomly selects between three options: insert, shrink and replace a node. Insert and shrink options add and remove a tree element. If replacement is chosen, the algorithm randomly selects a node within an individual. If that node is terminal (i.e., a hyperparameter), the operator replaces it with a random allowed value. If the selected node is a primitive, the operator replaces the whole subtree, which has that primitive as a root, with another randomly created subtree.

3. METHODOLOGY

3.1. Approach

TPOT aims to discover the optimal pipeline by evolving a population of individuals or pipelines over generations. As discussed before, evolution involves performing genetic operations—mutation and crossover—and selection over randomly initialized pipelines. The value of TPOT is based on the idea that genetic operators benefit pipeline optimization.

The first experiment determines if evolving pipelines with TPOT output better-performing results than just initializing pipelines. For results to be comparable, both scenarios must have the same number of pipelines involved. For example, evolving 50 pipelines for five generations involves the same number of pipelines as initializing 250 pipelines.

A deeper analysis would require comparing model performances on different stages of evolution, i.e., generations. That would answer the question: if and when (i.e., in which generation) evolution scenario performance reaches the value of initialization scenario performance?

The second experiment aims to investigate the effectiveness of TPOT mutations and crossovers and discover what makes GP operators effective. Operator effectiveness is determined by the share of operators that improve the fitness value of the pipeline.

3.2. Objectives

The main objectives of this thesis are:

- To determine if evolving pipelines with TPOT output better-performing results than just initializing pipelines.
- To compare different scenario performances over generations and determine the number of generations necessary for evolving pipelines to reach the level of initialized pipelines.
- To evaluate the performance of mutations and crossovers.

The first experiment will cover the first two objectives, and the second experiment the last one. Experiments are run independently from each other. Nevertheless, both experiments are repeated 30 times and run on the same four test cases described in detail in chapter 4.2.

3.3. Data

After running the TPOT optimization process on some training data, TPOT returns a fitted TPOT object, which can be exported as a python code or used to get a prediction score. If the TPOT verbosity parameter is set to 2 or higher, then TPOT prints the best internal CV score for each generation and the best pipeline as a string. Therefore, available information about TPOT optimization includes pipeline performance scores, internal CV scores and best pipelines. Two later are available in the program output stream. TPOT library code manipulations allowed extracting the best pipeline and evolution process details: genetic operator (mutation, crossover), pipeline, size and score of a selected individual (parent) and an individual after the operation (offspring). The rest of the metrics are calculated from extracted data: operation success rate, size increase, mutation type, mutated object, crossover type, crossover base, pipeline tree depth and operation level.

Scores

The analysis includes three types of scores: training, test and internal. The test score is the main performance indicator for a predictive model. Training score helps to detect overfitting. Internal CV score is a way to evaluate pipeline performance while training the pipeline.

First, a dataset is split into training and test subsets. The training set accounts for 90% of all data and is used for TPOT optimization. A training score evaluates the performance of the resulting pipeline trained on training data to predict target values of the same data. A test score is based on the pipeline's ability to predict target values of new data—testing subset.

Internal CV tests the pipelines on different folds of training data and returns the average score. Internal CV score is included in the pipeline fitness value and plays a crucial role in the pipeline selection. The best internal score is the CV score of the fittest pipeline in its generation, and it can be accessed from the TPOT output stream if it is run with high verbosity. Scores of individuals before and after mutation and crossover are also internal CV scores. They are used to calculate the improvement rate.

Test cases include both classification and regression problems. In both cases, default scores are used: accuracy for classification and negative MSE for regression. Negative MSE is converted into negative RMSE (Formula 3.1) for interpretability.

$$-\mathrm{RMSE} = -\sqrt{-(-\mathrm{MSE})} \tag{3.1}$$

Success rate

Success or improvement rate is a ratio of the number of successful or improved operations to the total number of operations (Equation 3.2). Successful operations are crossovers and mutations that improve the score of the pipeline. In other words, if the score of the individual selected for mutation or crossover is lower than the score after mutation or crossover, then the operation is considered successful (Equation 3.3).

$$success\_rate = \frac{\sum_{i=1}^{N} success_i}{N} \tag{3.2}$$

$$success_i = \begin{cases} 1 & , if\ score\_after_i > score\_before_i \\ 0 & , otherwise \end{cases} \tag{3.3}$$

While mutation requires one input individual (pipeline), crossover requires two. Crossover outputs two modified individuals; however, TPOT takes only the first one to the offspring set. In order to determine if a crossover is successful, the resulting individual (first offspring) is compared to the parent with the highest score.

Mutation metrics

Mutation operations differ by the type of mutation and the mutated object. During mutation, tree nodes or subtrees can be added to, removed, or replaced in the pipeline. Having both pipelines before and after the operation allows us to identify which part of the pipeline was changed: classifier, regressor, preprocessor, selector, combiner or hyperparameter.

In order to determine mutation type, pipelines before and after the operation are compared. Pipeline strings are converted into lists of pipeline elements, comparing which gives lists of added and removed elements. A mutation is additive if only a list of added elements has values. A mutation is subtractive if only a list of removed elements has values. If both lists have values, then mutation has replaced an object. Empty lists mean no changes. Matching resulting list elements with the pipeline operator dictionary identifies the mutated object.

Crossover metrics

As mutation has only one input pipeline, it was also decided to keep only one input pipeline for crossover: the fittest parent, which score is compared with the score of the offspring. Any of two parents can be the fittest. The first offspring typically has a root of the first parent and a subbranch from the second. Sometimes the root node of the offspring is inherited from the individual before crossover—the fittest parent or alpha. In other cases, the root note is inherited from the “unseen” parent—parent beta. If both trees are the same or none of the nodes match, it is impossible to distinguish the root origin. For an individual to be a root parent, two conditions should be met: (1) root classifier (or regressor) match and (2) at least all but one (n-1) root classifier (or regressor) hyperparameters match.

Crossovers can also be characterized by the type of switched pipeline tree parts: one node (leaf) or a subtree (branch). The crossover type is “leaf” if the difference between the pipelines before and after the operation is one hyperparameter. It is a “branch” type if the difference is multiple connected nodes or a subtree. If both trees are the same or none of the nodes match, the type is not available.

Operation level

The range of possible operation levels depends on the pipeline tree depth. The maximal operation level equals the tree's depth and implies changes in tree leaves (hyperparameters). The operation level is equal to zero if changes are performed in the root node. TPOT pipeline depth is often equal to the number of pipeline operators. The depth is smaller than the number of operators when a combiner is used in the pipeline because combiners can create forks in the tree that allow multiple operators to be on the same level.

3.4. HYPOTHESIS TESTING

The main performance indicators in this thesis are a score indicating how accurately the TPOT-generated pipeline can predict target values of unseen data and genetic operator success rate. Pipeline performance scores can be normally distributed or skewed in any direction if the problem is complex or the model is far from optimal. Scores of easy problems tend to be close to optimal (accuracy = 1 or -RMSE = 0) and negatively skewed. The Shapiro-Wilk test is used in this work to check whether scenario score or improvement rate distributions are Gaussian.

Mann-Whitney U test, also known as the Wilcoxon-Mann-Whitney test, is a nonparametric statistical significance test for two independent samples. It is applied if at least one of the distributions is not Gaussian. If both distributions are normal, Student’s T test determines if their means are significantly different. Each experiment is repeated 30 times and thus outputs 30 values, which is considered enough for a central limit theorem to hold. That makes the mean and standard deviation of observed scores an accurate approximation of the actual ones.

P-value is the probability of obtaining test results at least as extreme as the result observed. The threshold for p-values used in this thesis is 0.05. Any value below indicates that compared distributions are significantly different. In cases where the significance level is not stated, the value of 5% should be assumed.

4. EXPERIMENTAL STUDY

The study consists of two experiments. Both experiments aim to evaluate the effect of evolution (mutation and crossover) on the quality of the model trained using TPOT.

The first experiment compares the scores of models selected as a result of either evolution or initialization according to the following four scenarios:

1. Initializing 250 individuals.
2. Evolving 50 individuals for 5 generations with default mutation and crossover rates (0.9 and 0.1, respectively).
3. Evolving 25 individuals for 10 generations with default mutation and crossover rates (0.9 and 0.1, respectively).
4. Evolving 50 individuals for 5 generations with custom mutation and crossover rates (0.5 and 0.5).

All the scenarios involve exactly 250 individuals in total. That is because preliminary experiments showed that it is enough to get accurate model scores (see Results section) for simple problems like ”Breast cancer” and “Concrete” and decent scores for more complex problems: like “Waveform” and “Bioavailability” (see Chapter 4.2).

Each scenario was run 30 times on each of four test cases: two classification problems (“Breast cancer” and “Waveform”) and two regression problems (“Bioavailability” and “Concrete”). The conclusion about the significance of evolution-added value is based on two analyses. First is an analysis of the distribution of accuracy and RMSE scores for each test case separately. The second is the development of the best internal Cross-Validation (later CV) score over generations compared to the best internal CV scores of initializations—the first scenario.

The second experiment analyses the performance of genetic operators: crossover and mutation. A crossover takes two individuals as parents and returns two offspring, but in TPOT, only the first (by order) offspring is taken as an output. Mutation, however, inputs and outputs one individual. Every time an operation happens, the fitness values of input and output individuals are evaluated and compared. TPOT always aims to maximize fitness; therefore, operations that output individual fitness is higher than input individual fitness are called successful or improved. In the case of crossover, the highest parent fitness is compared with the first offspring’s fitness. TPOT uses a Pareto front that evaluates two fitnesses: model best internal CV score (accuracy or MSE) and the size of the individual. In operator effectiveness evaluation, only the CV score is taken into account. Operator effectiveness is based on a relative number of successful operations in each test case.

4.1. EXPERIMENTAL SETTINGS

As discussed above, the study consists of two experiments. The first experiment compares the performance of the models generated according to three scenarios: (1) initialization of 250 individuals, (2) evolution of 50 individuals for 5 generations, and (3) evolution of 25 individuals for 10 generations. Both evolution scenarios use default TPOT parameters for mutation and crossover rates: 0.9 and 0.1 (Table 4.1). In the second experiment, the performance of mutation and crossover operations is evaluated; therefore, mutation and crossover rates are adjusted to equal 0.5. This results in about 125 operations of each kind per run per test case and about 3750 operations in total (30 runs, one test case). The second experiment setting corresponds to scenario 4 (Table 4.1). Besides extra computation regarding operator effectiveness, the second experiment also produces the output valid for the analysis of the first experiment. Therefore, scenario 4 is also included in this analysis.

Table 4.1 *Experimental scenario settings*

Scenario	**Repetitions**	**Generations**	**Population size**	**Mutation rate**	**Crossover rate**
1	30	1	250	0	0
2	30	5	50	0.9	0.1
3	30	10	25	0.9	0.1
4	30	5	50	0.5	0.5

The number of generations cannot be set to zero in TPOT. Therefore, to simulate the initialization of 250 individuals, TPOT is run for one generation with mutation and crossover rates equal to zero. Thus, only the selection was used in scenario 1. In this thesis, only scenarios with mutation or crossover operations are called evolution.

TPOT optimization is run with default configurations for both classification and regression problems. That implies that solution search space includes classifiers and regressors from sklearn and xgboost libraries, selectors, and preprocessors from sklearn. It excludes feature construction operators and neural network-inspired models.

The random seed is set to be equal to the microsecond of the current timestamp and is reset every time (1) before splitting the dataset into training and testing and (2) before running the TPOT algorithm. TPOT is trained on 90% of the dataset and tested on the rest 10%. Each experiment is run 30 times for each of the 4 test cases.

Experiments are written and run with Python 3.9 using TPOT 0.11.7 and DEAP 1.3.1 as the basis and with the help of NumPy 1.22.3 and scikit-learn 1.0.2.

4.2. TEST CASES

An experimental study is performed on four test cases—two classifications and two regression problems. None of the cases has missing values nor required pre-processing. The basic information about the test cases is summarized in Table 4.2.

Table 4.2 *Test case descriptions*

Name	**Type**	**Attributes**	**Instances**	**Output range**
Breast cancer	Classification	30	569	1 = malignant, 0 = benign
Waveform	Classification	40	5000	0 = Class 1, 1 = Class 2, 2 = Class 3
Bioavailability	Regression	241	359	[0.4; 100]
Concrete	Regression	8	1030	[2.33; 80.6]

4.2.1. Breast cancer

Breast Cancer Wisconsin (Diagnostic) Data Set (Wolberg et al., 1995) (later: breast cancer) is a classification dataset applicable for breast tumour diagnoses, which was first introduced in the research by Street et al. in 1993. It consists of 569 instances of described characteristics of the cell nuclei extracted from the digitized image of fine needle aspirate of a breast mass via the image analysis program Xcyt (Mangasarian et al., 1995). 357 (or 63%) of the instances are malignant, and the rest (212 or 37%) are benign. Each image consisted of multiple nuclei; therefore, each instance is described as a mean, standard error, and the worst value of each of 10 features: radius, texture, perimeter, area, smoothness, compactness, concavity, number of concave points, symmetry, and fractal dimension. There are 30 attributes in total.

Street et al. (1993) performed classification using MSM-T (Multi-surface method – Tree), and the resulting classifier with three features—mean texture, worst area, and worst smoothness—had an accuracy of 97.3%.

4.2.2. Waveform

Waveform Database Generator (Version 2) Data Set (Breiman et al., 1988) (later: waveform) is a synthetically generated dataset with 5000 instances equally split into three classes that are described with 40 features—consecutive “height” values, which plot has a waveform. Three basic waveforms gradually increase, reach their peaks in the 7th, 11th, and 15th attributes, and decrease to the initial value of zero (Figure 4.1 Base Waves). All base waveforms are six units high at their peaks and are zeros after the 21st attribute. Three dataset classes are linear combinations of two different base waveforms with added noise generated with a normal distribution centred around 0 with a standard deviation of 1. Class 1 is a combination of 1st and 3rd waveform, class 2: 1st and 2nd, and class 3: 2nd and 3rd. Figure 4.1 shows 100 instances of each class.

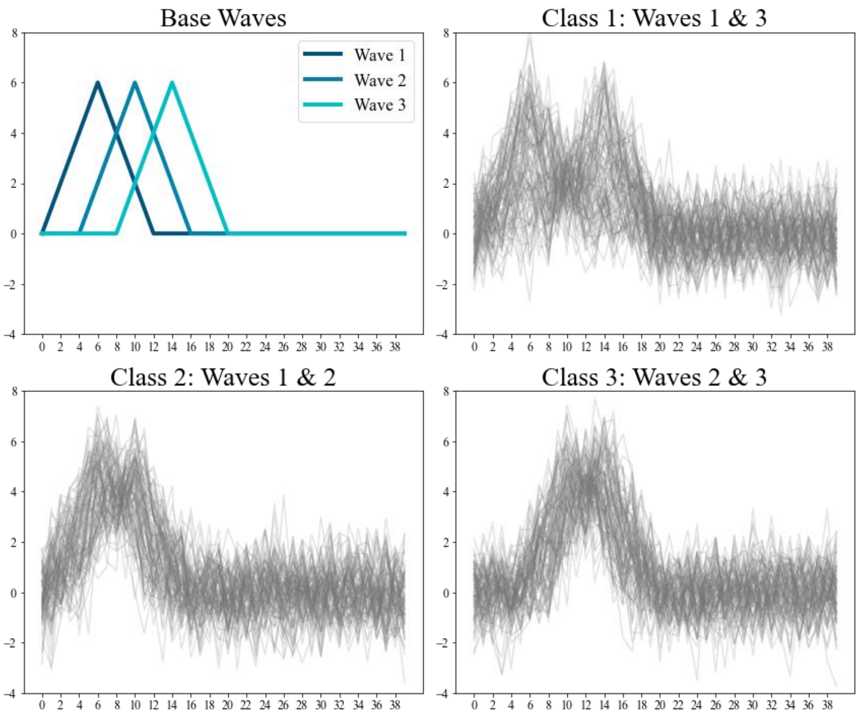


Figure 4.1 Basic waveforms and 100 instances of each of the three classes.

4.2.3. Bioavailability

Human oral bioavailability measures the percentage of an oral administered drug dose that effectively reaches systemic blood circulation. The bioavailability dataset contains 359 drug molecule observations with measured bioavailability values and 241 molecular descriptors that identify a drug. Each of 359 drug molecular formulas is expressed in a line notation called simplified molecular-input line-entry system (SMILES) that allows extracting bi-dimensional molecular descriptors using ADMET Predictor software. The dataset is the same as in Archetti et al. (2007).

Table 4.3 *Example of a drug expressed in linear notation (SMILES) and as a set of molecular descriptors.*

Drug	SMILES	Molecular descriptors	
Dopamine $C_8H_{11}NO_2$	NCCc1cc(O)c(O)cc1	$\log D_{6.5}$	−3.37
		$\Delta \log D^b$	−0.90
		pK_a	8.9
		...	...

Source: Yoshida & Topliss (2000).

4.2.4. Concrete

The dataset was first introduced in the research by Yeh (1998). It consists of 1028 observations of concrete mixtures and eight variables that describe the compressive strength of concrete: cement (kg/m^3), fly ash (kg/m^3), blast furnace slag (kg/m^3), water (kg/m^3), superplasticizer (kg/m^3), coarse aggregate (kg/m^3), fine aggregate (kg/m^3) and age of testing (days). Yeh (1998) uses artificial neural networks that result in R^2 equal to 0.9143 on average for randomly sampled data (testing set), which was concluded to outperform the model based on regression analysis.

4.3. Experimental results

4.3.1. Initialization vs evolution

As discussed in the Experimental study chapter, the first experiment aims to provide a basis for the comparison of multiple scenarios. Scenario 1 initializes 250 individuals, scenario 2 evolves 50 individuals for 5 generations, and scenario 3 evolves 25 individuals for 10 generations. Both scenarios 2 and 3 use the default mutation and crossover rates: 0.1 and 0.9. Scenario 4 is similar to scenario 2, but with mutation and crossover rates equal to 0.5. Having a total of 250 individuals involved in each scenario makes scenarios comparable.

4.3.1.1. Breast cancer

Each of the four scenarios (Table 4.1) was trained 30 times on randomly selected 90% of the breast cancer dataset data and tested on the rest 10%. Training score is spread next to 1.00 in all scenarios (Figure 4.2), and it is equal to 1.00 in 10, 13, 16, and 14 of 30 cases in scenarios 1, 2, 3, and 4, respectively. The narrowing distribution and higher mean (▲ in Figure 4.2) and median signify a positive effect of the growing number of generations (from none in scenario 1 to 10 in scenario 3) on the training score. Training scores in scenarios 2 and 4 are very similar (p=0.858), which means that the difference in mutation and crossover rates (0.9 and 0.1 in scenario 1, 0.5 and 0.5 in scenario 2) does not have a significant impact on the training score in the

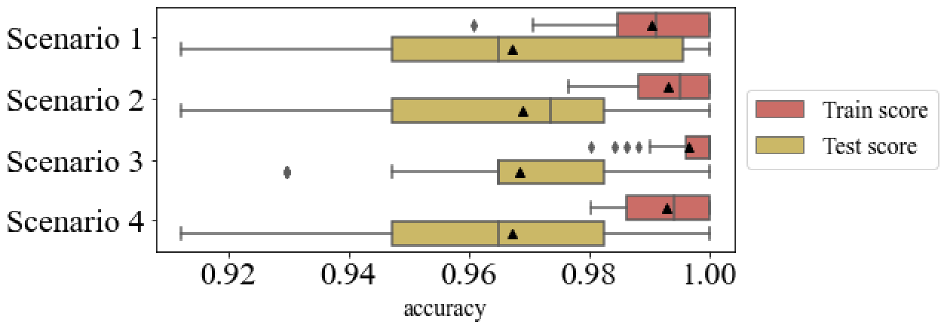


Figure 4.2 Breast cancer training and testing accuracy distributions for each testing scenario (n=30).

current test case. Train accuracy scores in scenarios 1 and 3 are significantly different (Mann-Whitney U test p-value is 0.01), despite any other scenario pairs whose p-values vary from 0.118 (scenario 2 vs 3) to 0.858 (scenario 2 vs 4).

Test accuracy scores vary from 0.9123 (scenarios 1, 2 and 4) and 0.9298 (scenario 3) to 1.0, with medians equal to 0.9649 (scenarios 1, 3 and 4) and 0.9737 (scenario 2). Scenario 1 had the most pipelines that scored 100% when tested, i.e., 8 cases, compared to 4 in scenarios 2 and 4, and 2 in scenario 3.

Table 4.4 *Breast cancer test accuracy mean, standard deviation and Mann-Whitney U p-values.*

	S	**Scenario 1** **Pop 250 Gen 0**	**Scenario 2** **Pop 50 Gen 5**	**Scenario 3** **Pop 25 Gen 10**	**Scenario 4** **Pop 50 Gen 5***
Accuracy mean (n=30)		0.9673 ±0.0264	**0.9690** ±0.0219	0.9684 ±0.0192	0.9673 ±0.0224
p-value	H_0: S = S_1	-	0.832	0.934	0.964
	H_0: S = S_2	0.832	-	0.841	0.747
	H_0: S = S_3	0.934	0.841	-	0.884
	H_0: S = S_4	0.964	0.747	0.884	-

Mean test accuracy is the highest (0.969) in scenario 2. However, Mann-Whitney U test results show no significant difference between any scenarios. Table 4.4 shows test accuracy means for each scenario and Mann-Whitney U test p-values for each pair of different scenarios. Neither of the scenarios has a mean accuracy higher than a baseline result of 0.973; however, almost half (46.7%) of the results surpass the baseline: 43.3%, 50%, 46.7%, and 46.7% cases in scenarios 1 to 4, respectively.

TPOT is trained on 90% of the Breast cancer dataset, i.e., 512 instances. Each pipeline is evaluated in a 5-fold CV during the TPOT optimization process. Therefore, the CV score is the average of 5 accuracy test results. The best-in-population CV score value is tracked for each

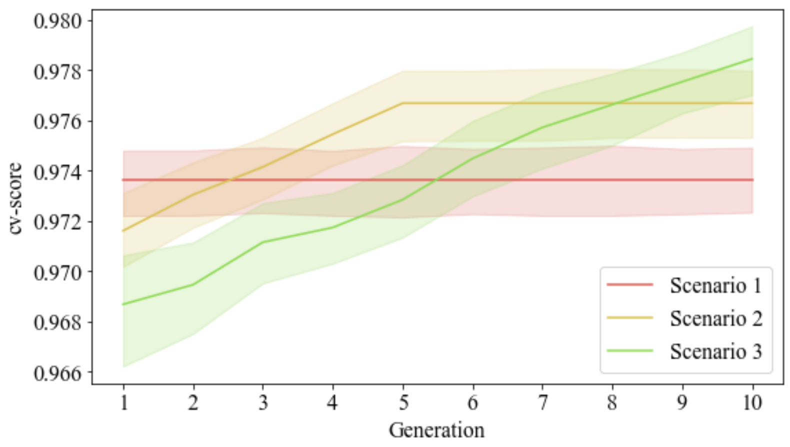


Figure 4.3 Best internal CV score distribution by generation for three scenarios. Breast cancer dataset.

generation. Each scenario is run 30 times, which results in 30 values for each generation. Information on the best internal CV score distribution is illustrated in Figure 4.3. As scenarios differ in the number of generations (0, 5, and 10 for scenarios 1, 2, and 3, respectively), last generation values are extended until Generation 10, which allows us to compare distributions of the scenarios. In scenario 1, initialization CV score values are copied over generations 1-10; in scenario 2, 5th generation CV score values are copied over generations 6-10. The shaded area in Figure 4.3 is a 95% confidence interval.

As expected, the number of individuals in the population positively correlates with the best CV score in generation 1: initialization of 250 individuals (scenario 1) having the highest result and 25 individuals (scenario 3) having the lowest. It takes about 3 generations for scenario 2 results to reach the level of scenario 1, about 6 generations for scenario 3 to reach the level of scenario 1 and 8 generations to reach the level of scenario 2. Scenario 1 has an average CV score of 0.9736. Scenario 2 started with an average CV score of 0.9716 in generation 1 and improved to 0.9767 in generation 5 (0.5% improvement). In scenario 3, the CV score improved by 1% by the 10th generation: from 0.9687 to 0.9785. According to Mann-Whitney U tests, CV score distributions are significantly different (5% significance level) in generation 1: p-values are 0.043 for Scenario 1 vs 2, 0.043 for Scenario 2 vs 3, and less than 0.001 for Scenario 1 vs 3. CV score distributions are also significantly different (α=5%) in generation 10: p-values 0.002 for scenario 1 vs 2, 0.044 for scenario 2 vs 3 and less than 0.001 for scenario 1 vs 3.

Running TPOT 120 times in total resulted in 118 unique pipelines. Two pipelines with Logistic Regression resulted twice each. One of the pipelines used Min Max Scaler for pre-processing, and inverse regularization strength (hyperparameter C) equal to 20 (Pipeline 4.1), and the other one uses Max Absolute Scaler and C equal to 15 (Pipeline 4.2). Pipeline 4.1 appeared both times under scenario 1, but Pipeline 4.2 appeared in scenarios 1 and 2.

LogisticRegression(MinMaxScaler(input_matrix), C=20.0, dual=False, penalty=l2) *(4.1)*

LogisticRegression(MaxAbsScaler(input_matrix), C=15.0, dual=False, penalty=l2) *(4.2)*

Regardless of a variety of classifiers and pre-processors, many pipelines differ with just one or several hyperparameters. If hyperparameters are disregarded, there are only 69 unique combinations of classifiers, pre-processors, and selectors, top of which are Gradient Boosting classifier without pre-processing (19 pipelines) and Linear SVC with Min Max Scaler applied on the dataset (7 cases).

In most of the cases (82 out of 120 pipelines), one classifier was used in the pipeline, but in several cases, two (34/120 cases) or even three classifiers (4/120 cases). In four cases, the same classifier was used twice. 42% of the pipelines (i.e., 50/120 cases) do not use any of the 14 different pre-processing algorithms, but in most cases, pre-processors are used at least once: one pre-processor in 58 cases, two in 11 cases, and three once. In one pipeline, two same preprocessors were used. The vast majority (110/120 cases or 92%) of pipelines do not involve any of the five selection algorithms; it is only used in 10 pipelines: once in 8 pipelines and twice in 2. Four resulting pipelines combined two data inputs: twice raw input data was combined with pre-processed data, once a combination consisted of pre-processed data and data with a selector applied, and once a combination of two raw data inputs was used. The average pipeline size is 2.175, and almost half (63 cases) of the pipelines are size 2. An equal number of pipelines are either of a size 1 or 3 (24 cases each). A few individuals are of a greater size: six are of size 4, and three are of size 5.

TPOT optimal pipeline uses Gradient Boosting Classifier as a root classifier in most of the cases of all scenarios (from 10/30 cases in Scenario 1 to 16/30 in Scenario 3, 51/120 in total). Other classifiers in all scenarios are the Extra Trees Classifier and Stochastic Gradient Descent (SGD) Classifier; however, they appear much less frequently (14/120 and 6/120 total cases, respectively). Linear Support Vector Classification (Linear SVC) is the second most popular classifier in all scenarios but Scenario 3 where it does not appear at all. Figure 4.4 shows all classifier popularity by scenario.

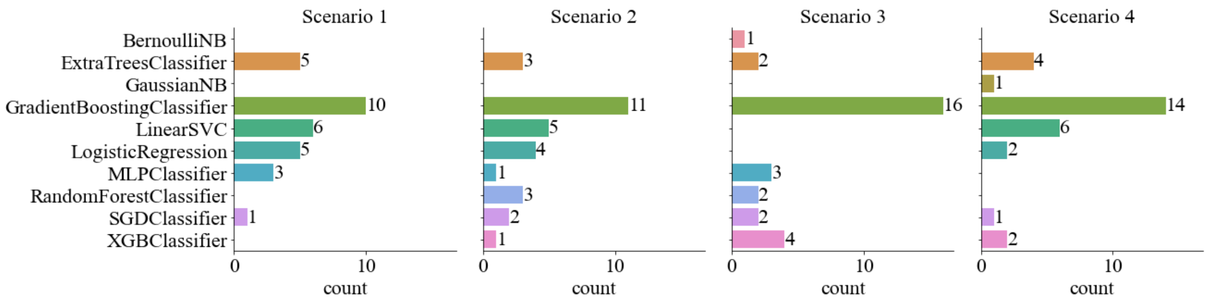


Figure 4.4 Number of times classifiers appear in the root of TPOT optimal pipelines for the breast cancer dataset.

The variance of test scores in scenario 3 is the lowest (Figure 4.5), which could indicate a convergence of the classifier models. Each of the algorithms is not expected to result in significantly different test accuracy if they were the result of the evolution of more than ten generations. However, most classifiers have fewer than ten observations, which is not enough to make a confident conclusion.

Almost half of the results exceed the baseline accuracy of 0.973; however, the distribution across classifiers varies—64% (7/11) of Logistic Regression, 53% (9/17) of Linear SVC, 43% of (22/51) Gradient Boosting Classifier and (6/14) Extra Trees Classifier results. Figure 4.5 summarizes classifier performance considering the baseline value by scenarios.

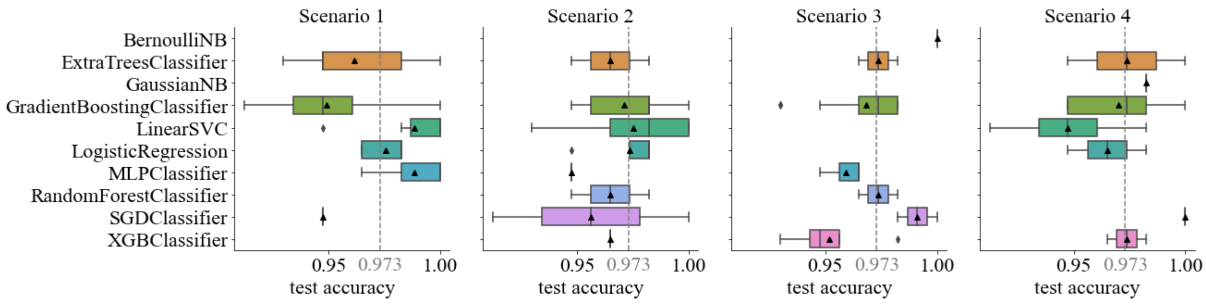


Figure 4.5 Test score distribution for each classifier of TPOT optimal pipelines for the breast cancer dataset.

4.3.1.2. Waveform

Randomly selected 4500 waveform dataset instances (90% of the dataset) were used to find an optimal TPOT pipeline, and the remaining 500 instances were used to test it. The process was repeated 30 times, and the resulting accuracy distributions are presented in Figure 4.6. Training accuracies vary from 0.868 (scenario 3) to 1.0 (scenario 2) and are positively skewed with medians ranging from 0.8756 (scenario 3) to 0.8857 (scenario 2) and means (▲ in Figure 4.6) within 0.8850 (scenario 1) - 0.9067 (scenario 2) range. Scenario 2 training results are higher and are significantly different from scenario 1 & 3 results (Mann-Whitney U test p-values 0.016 and 0.022, respectively). The difference between all other scenario pairs is not significant (5% significance level): p-values range from 0.105 in case scenario 2 vs scenario 4 to 0.988 in case scenario 1 vs scenario 3.

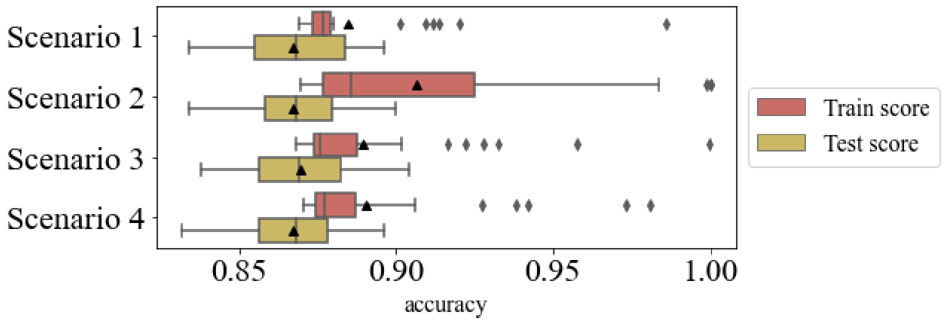


Figure 4.6 Waveform training and testing accuracy distributions for each testing scenario (n=30).

Scenario 3 has the highest mean test accuracy (0.8695), but it is not significantly different from any other scenario test results. According to the Shapiro-Wilk test, test accuracy distributions are normal; therefore, the T-test is used. It shows that there is no significant difference between any scenario pairs. Table 4.5 shows test accuracy means for each scenario and T-test p-values for each pair of different scenarios.

Table 4.5 *Waveform test accuracy mean ±standard deviation and T-test p-values.*

	S	**Scenario 1 Pop 250 Gen 0**	**Scenario 2 Pop 50 Gen 5**	**Scenario 3 Pop 25 Gen 10**	**Scenario 4 Pop 50 Gen 5***
Accuracy mean (n=30)		0.8675 ±0.0177	0.8675 ±0.0174	**0.8695** ±0.0179	0.8672 ±0.0148
p-value	H_0: S = S_1	-	1.000	0.654	0.950
	H_0: S = S_2	1.000	-	0.652	0.949
	H_0: S = S_3	0.654	0.652	-	0.584
	H_0: S = S_4	0.950	0.949	0.584	-

Figure 4.7 illustrates the distribution of average accuracy results of the 5-fold CV performed during the TPOT optimization process. Results are shown for three scenarios and are split by generation. As scenarios differ in the number of generations, the last available generation values are extended until Generation 10. The shaded area in Figure 4.7 is a 95% confidence interval.

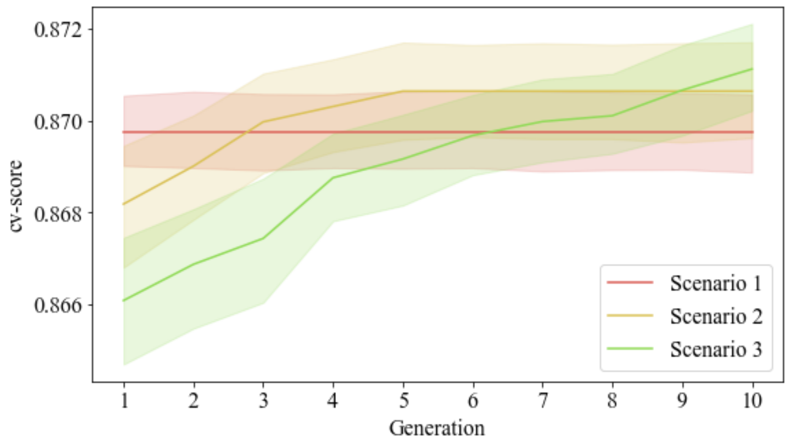


Figure 4.7 Best internal CV score distribution by generation for three scenarios. Waveform dataset.

The number of individuals in the population positively correlates with the best CV score in generation 1: 250 individuals and 0.8698 mean accuracy, 50 individuals and 0.8682 mean accuracy, 25 individuals and 0.8661 mean accuracy in scenarios 1 to 3, respectively. Accuracy distributions in generation one are not significantly different between scenario 1 and 2 (Mann-Whitney U test p-value=0.107) and is significantly different between scenarios 1 and 3 (p-value<0.001) and scenarios 2 and 3 (p-value=0.031). It takes about 3 generations for scenario 2 results to reach the level of scenario 1, about 6 generations for scenario 3 to reach the level of scenario 1 and 9 generations to reach the level of scenario 2. Scenario 2 scores improved by about 0.28% from an average of 0.8682 to 0.8706. Scenario 3 results improved by about 0.58% from 0.8661 to 0.8711. According to Mann-Whitney U tests, final results are not significantly different at a significance level of 5%, but scenarios 1 and 3 are significantly different at a significance level of 10% (p-value=0.055).

Running TPOT 120 times in total resulted in 106 unique pipelines. Pipelines 4.3-4.6 appear 11, 3, 2, and 2 times, respectively. If hyperparameters are disregarded, 61 unique combinations of classifiers, pre-processors, and selectors exist. Top pipelines use a Logistic Regression classifier without any pre-processors or selectors (14 cases), Extra Trees Classifier with PCA (14 cases), and Linear SVC (12 cases).

LogisticRegression(input_matrix, C=0.001, dual=False, penalty=l2) (4.3)

LinearSVC(input_matrix, C=1.0, dual=False, loss=squared_hinge, penalty=l2, tol=0.01) (4.4)

LogisticRegression(input_matrix, C=0.01, dual=False, penalty=l2) (4.5)

LogisticRegression(CombineDFs(input_matrix, input_matrix), C=0.001, dual=False, penalty=l2) (4.6)

Although most of the pipelines use one classifier (83 out of 120 cases), 32 use two, and 5 use three classifiers. In 8 cases, the same classifier was used twice in the same pipeline. Most of the pipelines use either one or none of the pre-processors or selectors: 80 without pre-processing, 81 without feature selection, and 52 without both. In the rest 40 cases, a preprocessor is used: once in 38 cases and twice in 2 cases. Similarly, one selector is used in 38 cases, and two selectors once. Six pipelines use a combination of two copies of the dataset as an input: in five cases, two raw datasets are combined; in one case, a raw dataset is combined with a pre-processed one. The average pipeline size is 2.083. Just 26 of the resulting pipelines are of the size one. The majority of the pipelines are of size two: 65 cases. In 22 cases, pipelines have three operators. Just seven individuals have a larger size of 4.

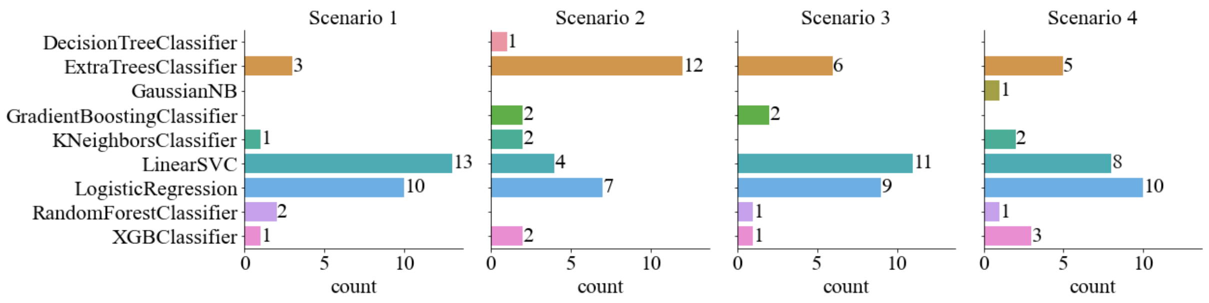


Figure 4.8 Number of times classifiers appear in the root of TPOT optimal pipelines for the Waveform dataset.

Waveform TPOT optimal pipelines use nine different classifiers as root classifiers. Still, in the vast majority (82%) of cases, either Linear SVC (36/120 cases), Logistic Regression (36/120), or Extra Trees Classifier (26/120) are used. All three top classifiers and XGB

Classifiers are used at least once in each scenario. Decision Tree Classifier and Gaussian NB are used only once. The best pipeline classifier used by scenarios is summarized in Figure 4.8.

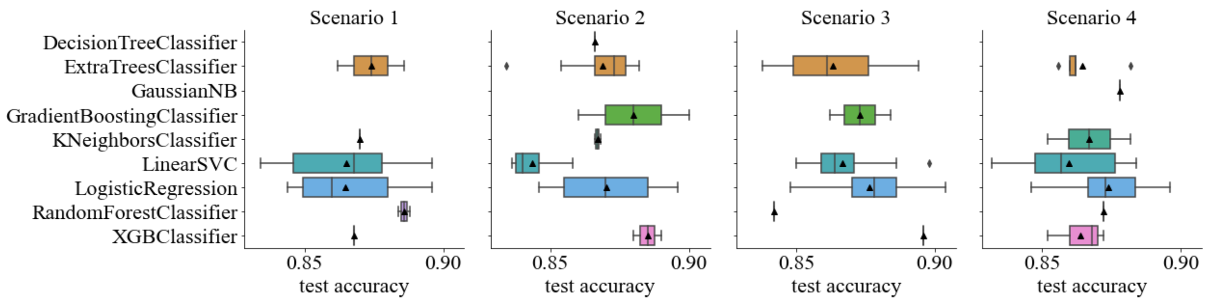


Figure 4.9 Test score distribution for each classifier of TPOT optimal pipelines for the Waveform dataset.

According to the Shapiro-Wilk test, accuracy distributions for pipelines with Linear SVC, Logistic Regression, or Extra Trees Classifier look Gaussian. T-test shows that Linear SVC (mean accuracy 0.8620) and Logistic Regression (mean accuracy 0.8713) model accuracies have significantly different distributions (p-value 0.028). Still, Extra Trees Classifier (mean accuracy 0.8674) is somewhat similar to both Linear SVC (p-value=0.213) and Logistic Regression (p-value=0.351). Linear SVC is significantly outperformed by Logistic Regression and Extra Trees Classifier in Scenario 2, although in other cases, those three classifiers have similar accuracy distribution (Figure 4.9).

4.3.1.3. Bioavailability

Bioavailability is a regression problem with 359 instances, 90% of which (or 323 instances) are randomly selected 30 times to find and train an optimal pipeline. Pipelines are tested on the remaining 36 instances. The distribution of negative root mean squared errors (RMSE) is summarized in Figure 4.10. Train scores range from -24.8272 (scenario 4) to -3.5877 (scenario 3), with means between -18.7462 (scenario 4) and -17.1990 (scenario 3) and test medians between -18.8193 (scenario 4) and -17.7607 (scenario 3). The difference between different scenario test results is not significant: the smallest T-test p-value is between scenarios 3 and 4 and is equal to 0.184. Test results range between -33.9780 (scenario 2) and -21.5612 (scenario 4), with scenario score medians between -28.2991 (scenario 1) and -27.4905 (scenario 3).

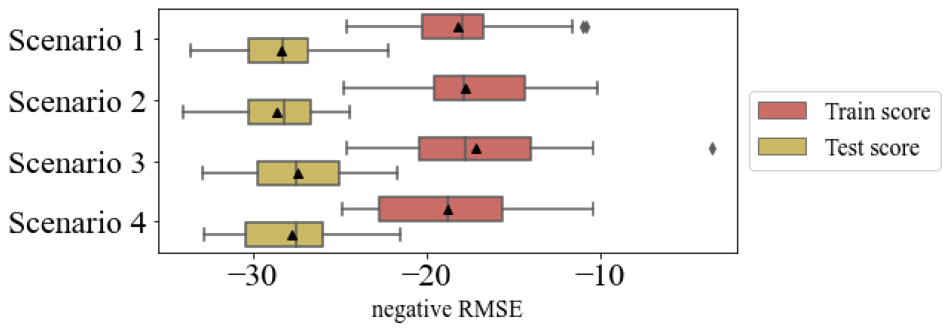


Figure 4.10 Bioavailability training and testing negative root mean squared error distributions for each testing scenario (n=30).

The highest negative RMSE was achieved under scenario 3; however, it is not significantly different from any other results. Scores within each scenario are normally distributed. As shown in Table 4.6, none of the hypotheses about scenario result similarity was rejected at a 5% significance level.

Table 4.6 *Bioavailability test RMSE mean ±standard deviation and T-test p-values.*

	S	**Scenario 1 Pop 250 Gen 0**	**Scenario 2 Pop 50 Gen 5**	**Scenario 3 Pop 25 Gen 10**	**Scenario 4 Pop 50 Gen 5***
RMSE mean (n=30)		-28.378 ±2.645	-28.647 ±2.551	**-27.453** ±3.100	-27.731 ±3.251
p-value	H_0: S = S_1	-	0.690	0.219	0.402
	H_0: S = S_2	0.690	-	0.109	0.230
	H_0: S = S_3	0.219	0.109	-	0.735
	H_0: S = S_4	0.402	0.230	0.735	-

The distribution of average negative RMSE results of 5-fold CV performed during the TPOT optimisation process and split by generation is shown in Figure 4.11. As scenarios differ in the number of generations, the last available generation values are extended until Generation 10. The shaded area in Figure 4.11 is a 95% confidence interval.

As expected, the number of initialised individuals positively correlates with the best CV score in generation 1: mean scores -27.3415, -27.4464, and -27.8801 in scenarios 1 to 3 with 250, 50, and 25 individuals, respectively. According to Mann-Whitney U, Test score distributions are not significantly different between scenarios 1 and 2 in generation one (p-value 0.387). However, the CV scores of scenario 3 are significantly different from the results of scenarios 1 and 2 (p-values 0.002 and 0.016). It takes about three generations for scenario 2 results to reach the level of scenario 1. Still, the average score of Scenario 3 does not reach the level of scenarios 1 and 2 during ten generations of evolution. Scenario 2 scores improved by about 0.57% from an average of -27.4464 to -27.2913. Scenario 3 improved by about 1.4% from an average of -27.8801 to -27.4951. According to Mann-Whitney U tests, final results are not significantly different at a significance level of 10%.

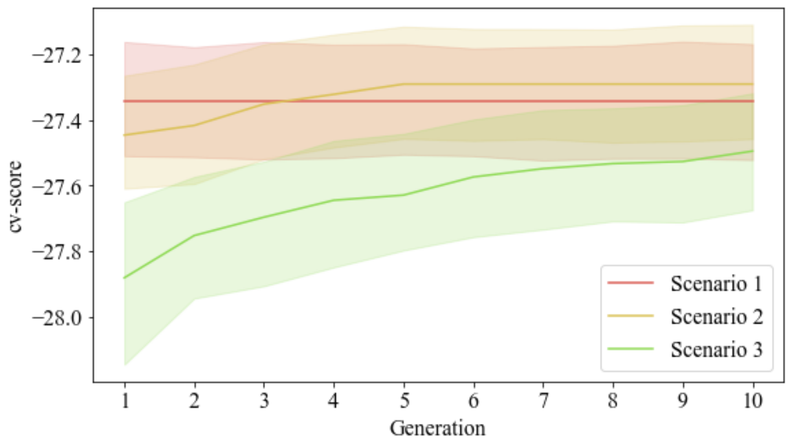


Figure 4.11 Best internal CV score distribution by generation for three scenarios. Bioavailability dataset.

Running TPOT 120 times in total resulted in all 120 unique pipelines. However, in terms of operator combinations, there are 40 unique pipelines. The most common operator combination is the Extra Trees regressor with no pre-processing and no selection, which exists in 40 different versions thanks to various hyperparameter combinations. The second most common is the Random Forest regressor without pre-processing or selection, which has 25 different hyperparameter combinations.

Like other test cases, pipelines might have one or more regressors and none or more pre-processors and selectors. The use of the only regressor is the most typical and accounts for 97 pipelines, 59 of which are Extra Trees and 38 Random Forests. In 3 cases, the same regressor was used twice within a pipeline. In 20 out of 120 cases, the combination of two regressors is used; combinations of three and four are uncommon and appear 2 and 1 times, respectively. Most of the regression pipelines (95 out of 120) use unprocessed data as an input; in 20 cases, the data was pre-processed once, and in 5 cases, twice. The selector was used in just ten or about 8% of cases. In 4 cases, input data is a combination of two copies of the dataset, and in all cases, it is a combination of raw (unprocessed) data. All operations combined, most of the resulting pipelines for the Bioavailability problem have the size of 1, which means that 65 of the pipelines (or 54.2%) only have one regressor run on the data as it is. A large portion (i.e., 41 cases or 34%) of the pipelines have a size of 2; however, just a few have a size of 3 and 4 (12 and 2 pipelines, respectively). One of the largest pipelines has four regressors; the other consists of 3 regressors and an operator that combines two copies of the dataset.

All but one optimal TPOT pipelines use either Extra Trees or Random Forest as a root regressor. Extra Tree regressor is used in 77 out of 120 (or 64%) pipelines, Random Forest in 42 (35%), and Gradient Boosting regressor in one (Figure 4.12). Extra Trees are used more frequently than Random Forest in the case of scenario 1, but in the rest of the scenarios, regressors are used more. Scenarios 2 and 4, in which pipelines were evolved for the same number of generations, have the same regressor distributions: 18 Extra Trees and 12 Random Forests. Moreover, the performance of those regressors is not significantly different: T-test p-values are 0.748 for Extra Trees score distributions in scenarios 2 and 4 and 0.170 for Random Forest.

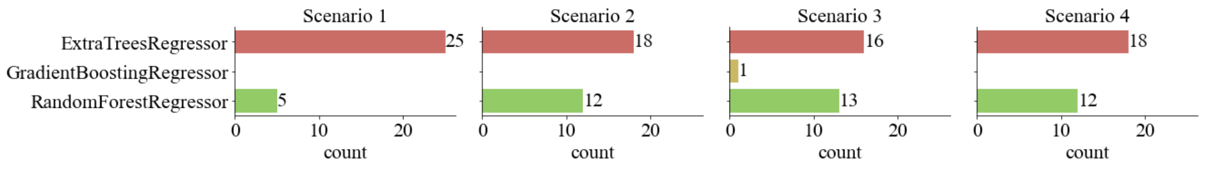


Figure 4.12 Number of times regressors appear in the root of TPOT optimal pipelines for the bioavailability dataset.

Despite minor differences in test negative RMSE distributions, Extra Trees and Random Forest perform the same when both the same regressors are compared within different scenarios, and different regressors are compared within the same scenario (Figure 4.13).

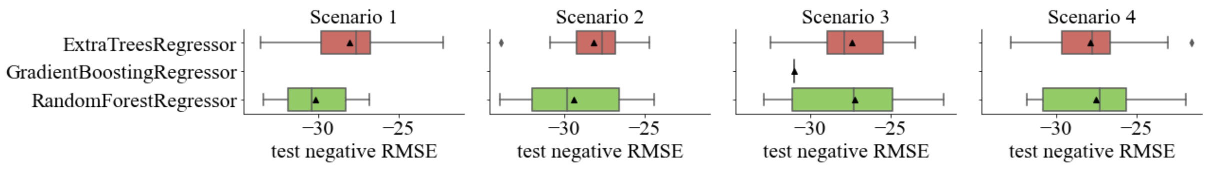


Figure 4.13 Test score distribution for each regressor of TPOT optimal pipelines for the bioavailability dataset.

4.3.1.4. Concrete

Concrete is a regression problem with 1028 observations, 90% of which (or 925 observations) are randomly selected 30 times to find and train an optimal pipeline. Pipelines are tested on the remaining 103 observations. The distribution of negative root mean squared errors (RMSE) are summarized in Figure 4.14. Train scores range from -3.7188 (scenario 4) to -0.6387 (scenario 3), with means between -2.0699 (scenario 4) and -1.7671 (scenario 3) and test medians between

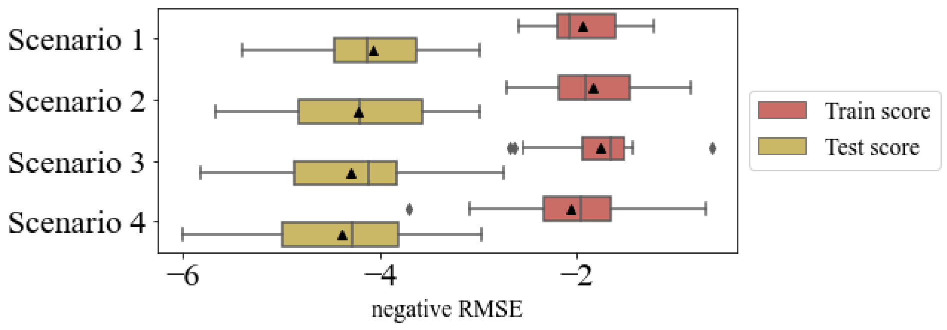


Figure 4.14 Concrete training and testing negative root mean squared error distributions for each testing scenario (n=30).

-2.0713 (scenario 1) and -1.6644 (scenario 3). Test results range between -5.9963 (scenario 4) and -2.7511 (scenario 3), with scenario score medians between -4.2890 (scenario 4) and -4.1218 (scenario 3). Within all scenarios, a significant difference exists only between the 3rd and 4th (Mann-Whitney U test p-value 0.026).

The highest average negative RMSE was achieved under scenario 1. However, it is not significantly different from any other results. Scores within each scenario are normally distributed. As shown in Table 4.7, none of the hypotheses about scenario result similarity was rejected at a 5% significance level.

Table 4.7 *Concrete test RMSE mean ±standard deviation and T-test p-values.*

	S	**Scenario 1 Pop 250 Gen 0**	**Scenario 2 Pop 50 Gen 5**	**Scenario 3 Pop 25 Gen 10**	**Scenario 4 Pop 50 Gen 5***
Accuracy mean (n=30)		**-4.076 ±0.607**	-4.226 ±0.808	-4.299 ±0.716	-4.399 ±0.783
p-value	H_0: S = S_1	-	0.420	0.198	0.080
	H_0: S = S_2	0.420	-	0.710	0.404
	H_0: S = S_3	0.198	0.710	-	0.612
	H_0: S = S_4	0.080	0.404	0.612	-

Figure 4.15 illustrates the best internal 5-fold CV score distribution by generation. First-generation values of scenario 1 and 5th-generation values of scenario 2 are copied over further generations, enabling distribution comparison of the scenarios. The shaded area in Figure 4.15 is a 95% confidence interval.

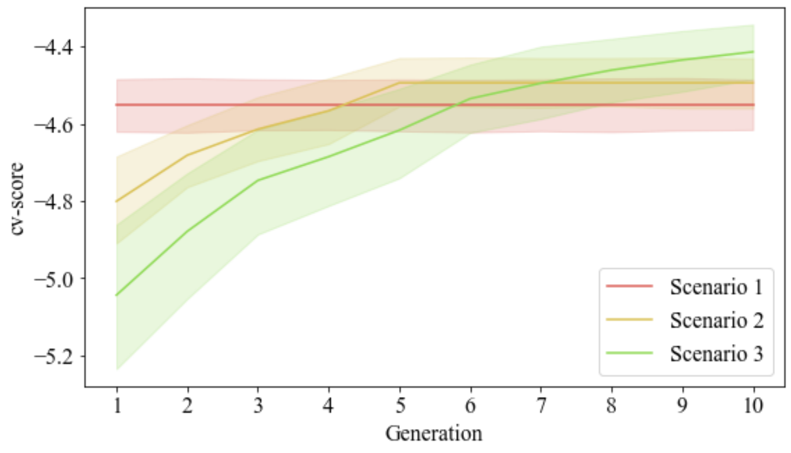


Figure 4.15 Best internal CV score distribution by generation for three scenarios. Concrete dataset.

First-generation CV scores predictably correlate with the number of individuals in the population in each scenario: Scenarios 1 to 3 have mean scores of -4.5501, -4.8011, and -5.0441. Score distributions in generation one are significantly different between scenarios 1 and 2 (Mann Whitney U test p-value 0.002), 1 and 3 (p-value < 0.001). In generation 1, CV scores are different for scenarios 2 and 3 only with a significance level of 10%: the p-value is 0.085. It takes about four generations for scenario 2 results to reach the level of scenario 1, about 6 generations for scenario 3 results to reach the level of scenario 1, and about 7 to reach the level of scenario 2. Scenario 2 results increased by 6.8% and reached the value of -4.4942 in generation 5; scenario 3 results improved by 14.3% and reached the level of -4.4140 in generation 10. The final CV scores are similar for scenarios 1 and 2, 2 and 3, but scenario 3 results are significantly higher than scenario 1 (U test p-value 0.013).

Running TPOT 120 times in total resulted in all 120 unique pipelines with 46 different operator combinations and a variety of hyperparameters, which make all pipelines unique. Extreme Gradient Boosting (XGB) regressor without pre-processing is the most common solution, appearing with 45 different hyperparameter combinations. Gradient Boosting regressor without pre-processing is another common solution for the concrete problem. It accounts for 14 unique solutions. A typical solution (i.e., 74 cases) consists of one regressor, which does not use any pre-processing or feature selection in 60 instances and uses either one in 14 cases. Oddly, the same regressor was used twice within a pipeline in 7 cases. Two regressors are used in 39 cases and three in 6 cases. Once four regressors are used. The dataset is pre-processed in 21 out of 120 pipelines: once in 20 cases and twice in 1 case. Selectors rarely appear in the final pipeline – only in 5 cases out of 120. A combination of two dataset copies was used as a regressor input in 5 pipelines: combinations of two raw dataset copies, raw and pre-processed datasets, two different regression outputs, regression output with a raw dataset, and regression output with a pre-processed dataset. The average pipeline size is 1.717. The resulting pipelines are mostly small: 59 pipelines are of a size one and 42 of a size two. In 14 cases, pipelines have three operators. Just five individuals have a larger size.

Optimal solution pipelines use nine regressors at the root to solve a concrete problem: 4 to 6 regressors per scenario (Figure 4.16). In most cases—scenario-wise and in total—Extreme Gradient Boosting (XGB) is the most popular solution (73 out of 120 cases or 60.8% in

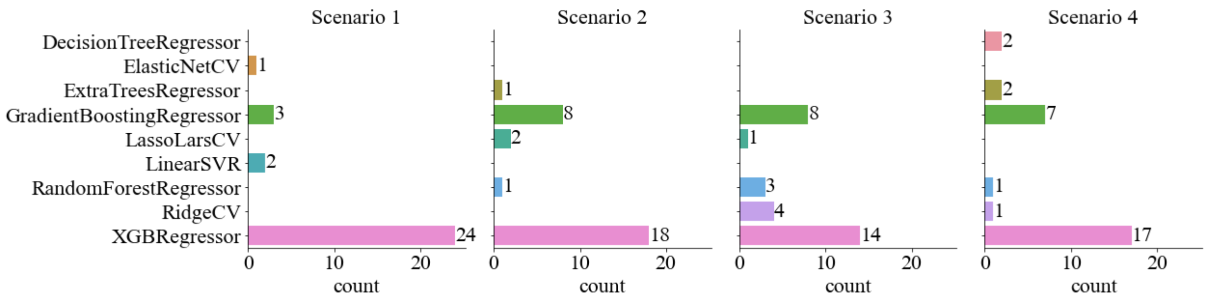


Figure 4.16 Number of times regressors appear in the root of TPOT optimal pipelines for the concrete dataset.

total). Gradient Boosting is the second most popular regressor (26 of 120 or 21.7%). The rest of the regressors appear up to 5 times and never in all four scenarios. Figure 4.16 shows all regressors by scenario.

Figure 4.17 shows no specific pattern in regressor performance by scenario. The only significant difference is between XGB performances in scenarios 1 and 4 (T-test p-value 0.034). Other regressors have too few observations to be comparable.

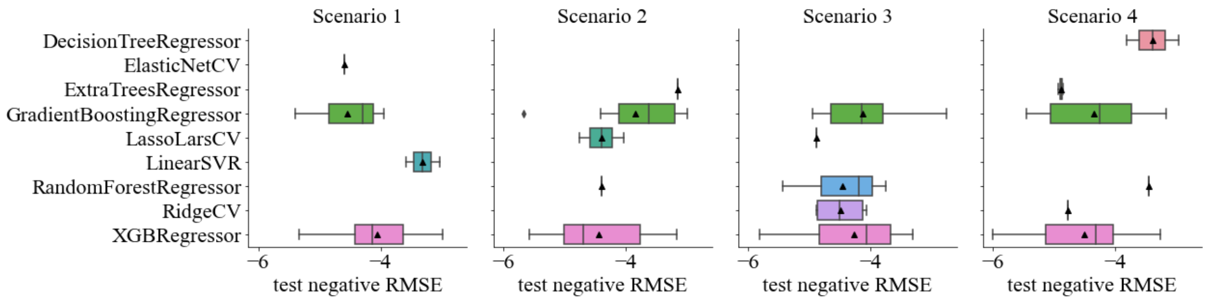


Figure 4.17 Test score distribution for each regressor of TPOT optimal pipelines for the concrete dataset.

4.3.1.5. Summary

The first experiment compares the performances of the TPOT-optimized pipelines under four scenarios: under the first scenario, the best pipeline is selected from the initialized one; under the other scenarios, pipelines first evolved for 5 and 10 generations. Although training scores of evolved pipelines (scenarios 2-4) are higher than those initialized (scenario 1) in classification cases, there is no significant difference in the testing scores. Nevertheless, scenario 3 resulted in the highest score for two of four test cases, scenarios 1 and 2 in one test case each. Table 4.8 summarises the mean testing scores for each test case: mean accuracy for breast cancer and waveform test cases and negative RMSE for bioavailability and concrete test cases.

Table 4.8 *Summary of the first experiment results: mean accuracy/negative RMSE and standard deviation by scenario.*

Test case	**Scenario 1 Pop 250 Gen 0**	**Scenario 2 Pop 50 Gen 5**	**Scenario 3 Pop 25 Gen 10**	**Scenario 4 Pop 50 Gen 5***
Breast cancer	0.9673 ±0.0264	**0.9690** ±0.0219	0.9684 ±0.0192	0.9673 ±0.0224
Waveform	0.8675 ±0.0177	0.8675 ±0.0174	**0.8695** ±0.0179	0.8672 ±0.0148
Bioavailability	-28.378 ±2.645	-28.647 ±2.5510	**-27.453** ±3.1000	-27.731 ±3.2510
Concrete	**-4.076** ±0.6070	-4.226 ±0.8080	-4.299 ±0.7160	-4.399 ±0.7830

Three scenario internal CV scores are compared by generation in three parts. The first and the second parts show which scenarios have significantly different score distributions after evolving the first and the last generations. In scenario 1, pipelines were not evolved; therefore, its only score distribution is compared with different generation results of scenarios 2 and 3. In Table 4.9, significantly different results at a 5% significance level are denoted with mathematical operator signs: < and >. Significant difference at a significance level of 10% is indicated as ≲. The approximate equality sign (≈) means no significant difference. The last part of the analysis shows when (and if) previously lower-performing scenario reaches the level of the better-performing scenarios. Table 4.9 shows at which generation a pair of scenarios intersect.

Initialized pipelines outperform evolved pipelines in generation 1 due to the larger size of the population. Although the second scenario reaches the level of the first one in generation 3 (4 in the concrete case), the final generation (here: 5) results exceed initialized scores only in one of four test cases. The third scenario reaches the level of scenario 1 in generation 6 and has significantly better final results in 3 of 4 cases.

Table 4.9 *Summary of internal CV score comparison by generation.*

Test Case	**First generation**			**Last generation**			**Intersection**		
Scenarios	**1&2**	**2&3**	**1&3**	**1&2**	**2&3**	**1&3**	**1&2**	**2&3**	**1&3**
Breast cancer	>	>	>	<	<	<	3	8	6
Waveform	≈	>	>	≈	≈	≲	3	9	6
Bioavailability	≈	>	>	≈	≈	≈	3	-	-
Concrete	>	≈	>	≈	≈	<	4	7	6

The resulting pipelines are analyzed to understand their construction better. Pipelines have more than one classifier or regressor in about a third of the cases. Classification problem solutions apply preprocessors more often and generally seem to have larger pipelines. As waveform data includes meaningless variables, solutions include selectors untypically often. Table 4.10 summarizes the TPOT-optimized pipeline size and composition by a test case. The table shows the number of pipelines where a certain number of classifiers, regressors, preprocessors, selectors and combiners appear. For example, there is one pipeline with four regressors in bioavailability and concrete cases.

Table 4.10 *Number of pipelines when a certain number of operators is used.*

Case	**Class./Reg.**				**Preprocessors**				**Selectors**			**Com.**	**Size**				
	1	**2**	**3**	**4**	**0**	**1**	**2**	**3**	**0**	**1**	**2**	**1**	**1**	**2**	**3**	**4**	**5**
BC	82	34	4	0	50	58	11	1	110	8	2	4	24	63	24	6	3
WF	83	32	5	0	80	38	2	0	81	38	1	6	26	65	22	7	0
BIO	97	20	2	1	95	20	5	0	110	10	0	4	65	41	12	2	0
CON	74	39	6	1	99	20	1	0	115	5	0	5	59	42	14	4	1

All regression solutions are unique, and a few classification solutions appear multiple times. If hyperparameters are disregarded, about half of classification and a third of regression pipelines have unique combinations of pipeline operators.

4.3.2. Operator effectiveness

Scenario 4 is specifically created to test the performance of the genetic operators while TPOT finds an optimal pipeline by evolving 50 individuals over five generations. Mutation and crossover rates are changed to 50% each, which allows generating enough observation of both operators. That is also the main difference between scenario 4 and scenario 2, which uses the default rate values (90% for mutation and 10% for crossover). Like in the other scenarios, the TPOT algorithm is run 30 times and produces 30 pipelines. The performance of those pipelines is discussed and analysed in chapter 4.3.1 Initialization vs evolution.

As the population size in each generation is 50, each generation of individuals is exposed to 50 genetic operations, which can be a mutation or a crossover with the same probability, summing up to 250 operations per iteration or 7,500 operations per test case. TPOT library code manipulations enabled tracking input and output of each operation: pipeline or individual selected to perform an operation on (parent individual with the highest fitness (score only) in case of crossover), pipeline after the operation (first offspring in case of crossover), and fitness values before and after the operation. As discussed previously, fitness value consists of the score—accuracy (for classification problems) or negative MSE (for regression problems)—and the size of the pipeline (i.e., number of pipeline operators). The type of genetic operation performed (crossover or mutation), iteration, and generation numbers are also tracked.

Data collected on each operation allows calculating its performance. Operation is considered successful if the score of the output pipeline is higher than the score of the pipeline after the genetic operation. Comparing pipelines before and after the operation allows identifying at what level the change was performed: 0 or root level if the final (i.e., the one that outputs the result) classifier or regressor is changed; the maximal level depends on the depth of the pipeline. Pipeline comparison before and after mutation showed whether pipeline operation was added, removed, or changed and what was the type of the added/removed/changed operator: classifier/regressors, selector, preprocessor, hyperparameter, or a “combiner” that combines multiple copies of the input data.

Instances with invalid values due to computation time-out are removed from the analysis. The final number of operations for each test case is summarized in Table 4.11.

Table 4.11 *Number of operations by the test case.*

Test case	**Number of operations**	**Number of successful operations**	**Number of crossovers**	**Number of mutations**
Breast cancer	7481	1729 (23.1%)	3756 (50.2%)	3725 (49.8%)
Waveform	7472	1846 (24.7%)	3703 (49.6%)	3769 (50.4%)
Bioavailability	7483	1928 (25.8%)	3798 (50.8%)	3685 (49.2%)
Concrete	7483	2220 (29.7%)	3727 (49.8%)	3756 (50.2%)

Each operation can be analysed in terms of (1) the nature of the operation: types of crossovers and mutation, (2) its locality: which level of the pipeline changes appear; and (3) its impact on the size of the pipeline.

4.3.2.1. Breast cancer

As discussed in the previous chapter, Breast cancer is an easy problem, and TPOT achieves high accuracy with 250 individuals. The mean test score for scenario 4 is 0.9673. Internal scores of selected individuals in generation 1 vary from 0.4004 (0.01 quantile) to 0.9824 (maximum), with median and mean scores equal to 0.9297 and 0.8801. The scores of the individuals generated in generation 5 range between 0.6247 (0.01 quantile) and 0.9844, with median and mean equal to 0.9610 and 0.9498.

The vast majority of all selected individuals are of size 1 (46.7%) or 2 (44.9%), and just a few reach the size of up to six. In generation 1, all the individuals before the operation were of size 1 or 2, but the number decreased to 83.5% by generation 5 (Figure 4.18). Larger pipelines developed over generations. The only individual of size 6 was selected in generation 5. Further evolution could lead to some increase in the sizes of selected individuals but to a limited extent as fitness score is punished for larger sizes.

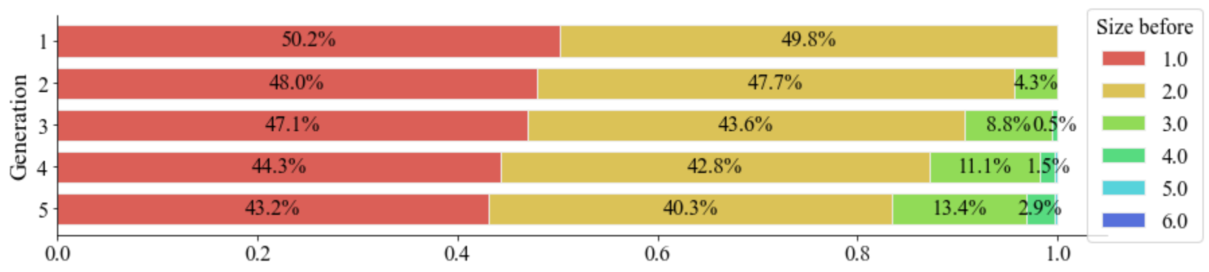


Figure 4.18 Selected individual size distribution by generation. Breast cancer.

Each of the 30 iterations involved about 250 operations which are roughly equally split into crossovers and mutations because of the same probability. Mutations tend to have a better chance of improving the accuracy of an individual (see Figure 4.19). On average, 29% of individuals have a higher score after the mutation than before. That improvement score varies from

16.8% to 37.1% in different iterations. Crossover, however, results in score improvements in 9.3% to 24.8% of cases, i.e., 17.4% on average.

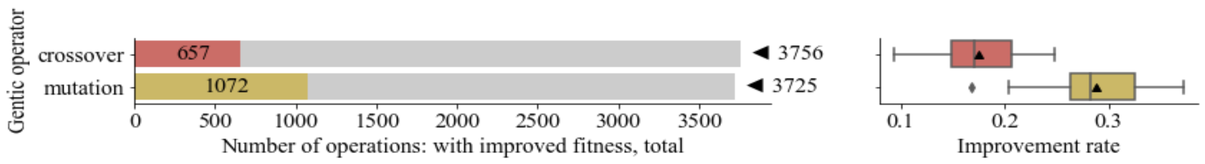


Figure 4.19 Amount and performance of operations by genetic operators. Breast cancer.

Only one-quarter of all chosen pipelines improve their performance after genetic operations, which also means that most operations lower fitness values. Thus, all generations' median scores after the operation are lower than before (Figure 4.20). A pool of selected individuals consists of an average of 29 unique pipelines, which in other cases can be as small as 23 and as large as 36. However, pipelines after an operation are all unique in most cases (107 of 150), averaging the number of unique pipelines to 49.6. Selection of the best portion from the pool of the previous generation and its offspring ensures gradual fitness improvement that slows down as scores approach the maximum value of 1.0. (Figure 4.20).

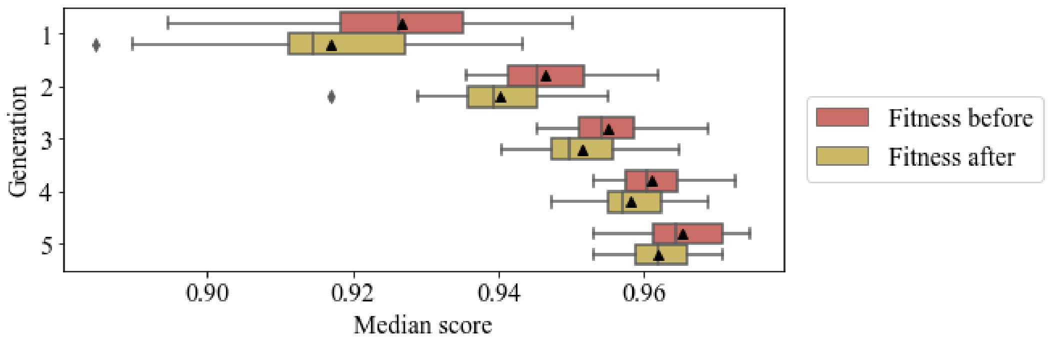


Figure 4.20 Distribution of median accuracies by the generation before and after genetic operation. Breast cancer.

Mutation

High scores are harder to improve. So, when pipelines give high scores with just a few individuals involved in their evolution, it is expected that the highest chance of score improvement is in the early stages of evolution, and chances diminish with generations. It is perfectly illustrated in Figure 4.21: the improvement rate for mutation is decreasing over generations from a mean improvement of 37% in generation 1 to 21% in generation 5.

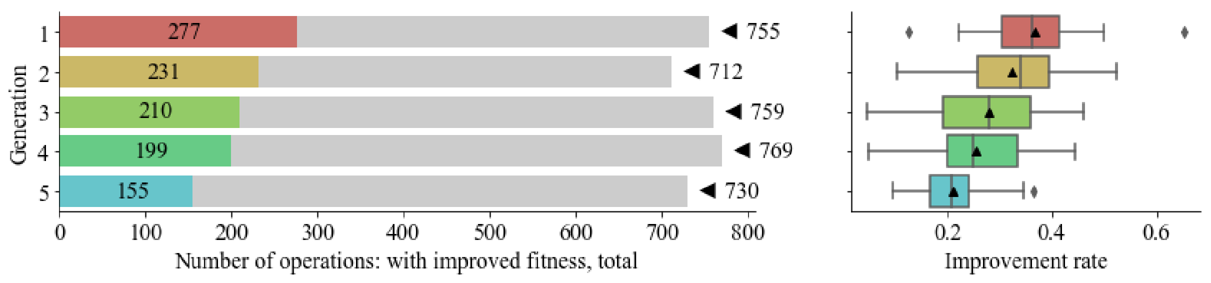


Figure 4.21 Amount and performance of mutations by pipeline generation. Breast cancer.

Mutation can appear in 3 different ways: change, add, and remove one node in the pipeline tree, resulting in possible pipeline change by a maximum of one unit. As can be seen from Figure 4.22, the majority, i.e., 1817 (49%) mutation operations, did not influence the size of the pipeline. The pipeline after mutation grew in 1603 or 43% cases which is much more than diminished cases: 306 or 8% mutations. It is largely because almost half of the chosen individuals have only one pipeline operator. Pipelines that increased as a result of the mutation have, on average, the highest improvement rate, which ranges between 16.98% and 47.62%, with a mean equal to 29.34%. Nevertheless, the same and decreased size pipeline performances are not significantly different from the top results and have an average rate of 28.80% and 26.17%, respectively.

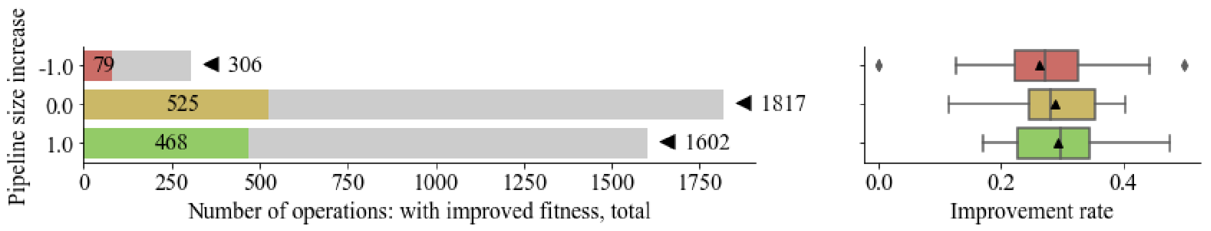


Figure 4.22 Amount and performance of mutations by pipeline size increase. Breast cancer.

All (but several exceptions) mutations that change a pipeline operator do not influence the size of an individual, additive mutations increase it by one, and subtractive mutation reduces it by one. Exceptions are operations that involve a pipeline element that combines two inputs (later: combiner), which is not counted as a pipeline operator in calculations of the size of the individual. Those exceptions account for 46 added, 12 removed, five switched to, and three switched from combiners. That explains the slight difference between pipeline size increase charts (Figure 4.22) and mutation type charts (Figure 4.23). Mean improvement rates for adding, changing, and removing mutations are 29.59%, 28.66%, and 25.51%, respectively. Although additive mutation performs the best and subtractive the worst, the difference between all three performances is not statistically significant.

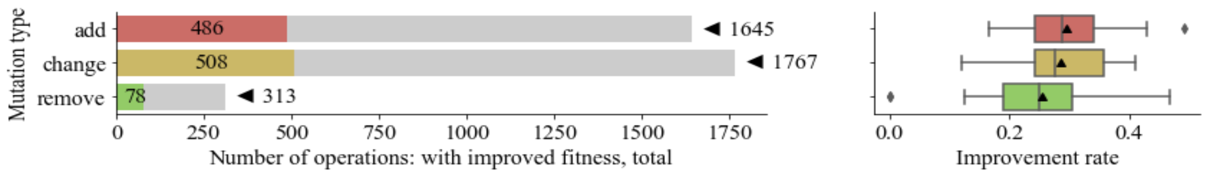


Figure 4.23 Amount and performance of mutation operations by type. Breast cancer.

Pre-processor and classifier are the most added pipeline components, amounting to 82.74% of all additive mutations (Figure 4.24). Adding a classifier contributes the most to an individual's performance—its average improvement rate is 36.83%, significantly higher than the selector's 32.35% and the preprocessor's 21.62%.

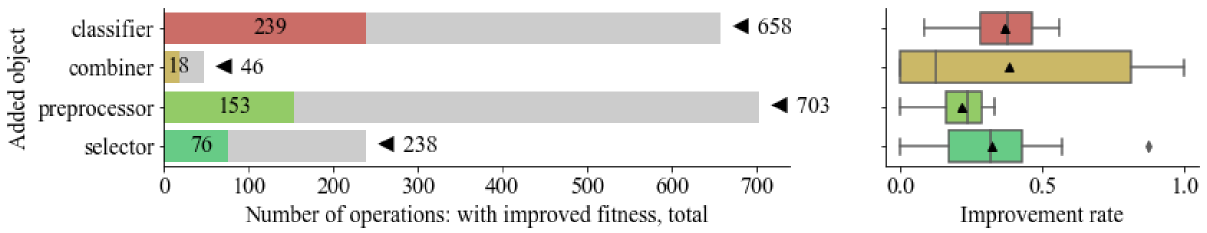


Figure 4.24 Amount and performance of mutation operations by the added object. Breast

Pipeline operators change in 14 ways, but Figure 4.25 shows only five with more than 20 observations. The full version of the chart can be seen in Appendix 2. Three fourth (or 1317) of all change mutations are the change in hyperparameter value, which also accounts for a third of all mutations. Hyperparameter adjustment improves the accuracy in 30.76% of cases on average. Switch of classifiers is the second most common change, which amounts to 343 operations or 19% of change mutations. Its average improvement rate is 22.22%. The rest of the changes do not have enough observations to conclude their effectiveness.

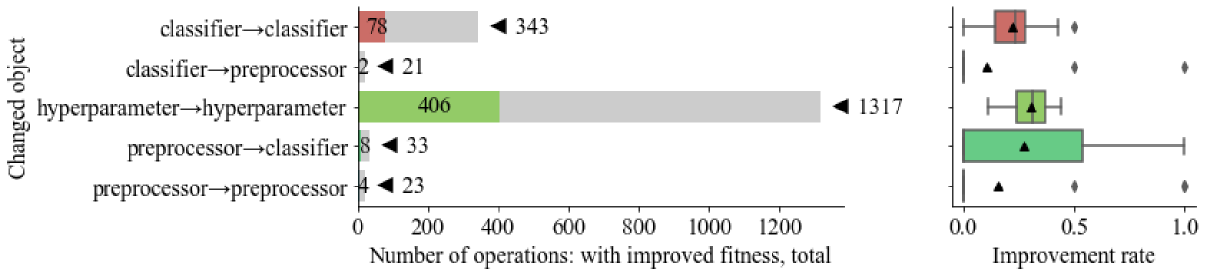


Figure 4.25 Amount and performance of mutation operations by the changed object (short). Breast cancer.

Removing mutation is the least frequent type of mutation. Its most common objects are the classifier and pre-processor (Figure 4.26), like in the case of additive mutation. Removing an extra classifier leads to fitness improvement in 26.8% of cases, which is greater than removing a pre-processor (25.2%). The difference, however, is not significant.

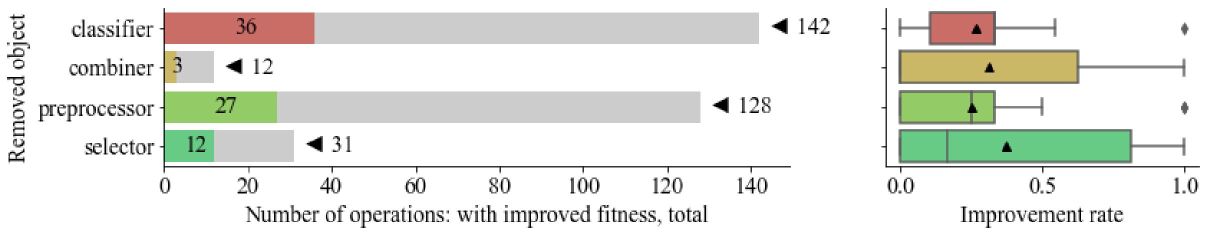


Figure 4.26 Amount and performance of mutation operations by the removed object. Breast

Change of the root classifier is the only change possible on level 0. It improves the performance of an individual in 22.53% of cases on average. Although root classifier changes in pipelines of any depth, it is most common in the pipelines of depth 1 (215 of 316 cases) because of the fewer mutation options small pipelines can have. Most of the changes happen on level 1 (Figure 4.27) because that is where root classifier hyperparameters reside, which is known as one of the most frequently changed parts of the pipeline. It is also where all the additions happen in the pipelines of depth 1, as well as all removals and most of the additions and changes in pipelines with depth 2.

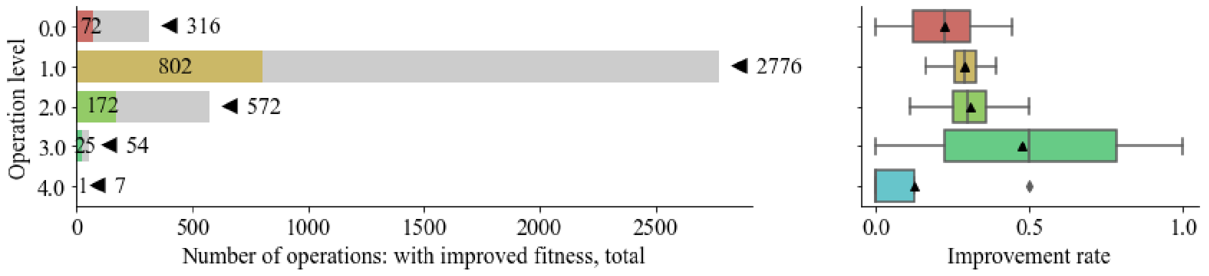


Figure 4.27 Amount and performance of mutation operations by the level of operation. Breast cancer

Crossover

Unlike mutation, the performance of crossover operations does not follow any pattern (Figure 4.28). Average improvement rates for crossover vary from 15% in generation 2 to 21% in generation 3. Crossover performance in generation 3 is significantly different from the ones in generation 1 (t-test p=0.044), 2 (p=0.007) and 4 (p=0.045).

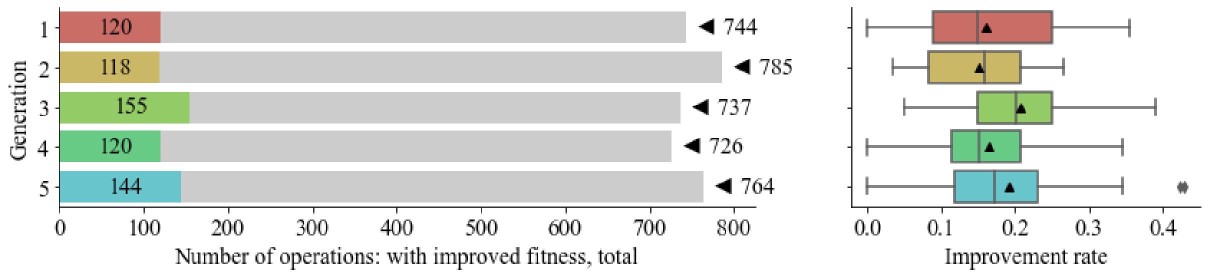


Figure 4.28 Amount and performance of crossover operations by generation. Breast cancer.

While mutation can result in maximum pipeline change by one unit, crossover impact on the pipeline size reaches up to 3 units difference in either direction (Figure 4.29).

Nevertheless, most operations did not change the size of the pipeline. The number of pipeline operators stayed the same in 4275 or 57% of total cases and 2458 (65%) crossover operations.

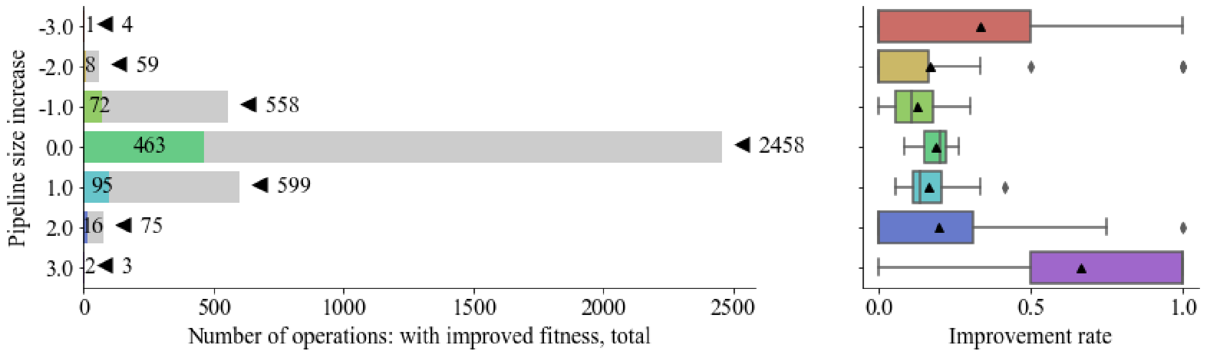


Figure 4.29 Amount and performance of crossover operations by pipeline size increase. Breast cancer.

Most (72.02%) of the crossovers happened on a leaf node level (Figure 4.30), which is only possible for hyperparameter nodes. It is the only option for small (i.e., of size 1) pipelines, which amount to almost half of the population. Mean improvement rates for branch and leaf crossovers are 18.16% and 18.92%, respectively, and are not significantly different. In 215 cases (or 5.7%), the type of crossovers cannot be identified because either: individuals (before and after a crossover) are equal or have no common pipeline operators.

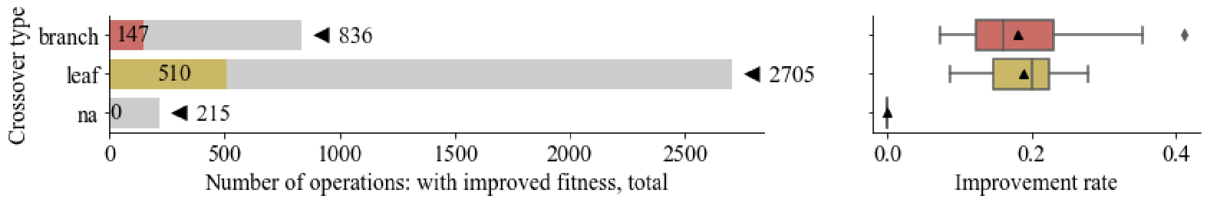


Figure 4.30 Amount and performance of crossover operations by crossover type. Breast cancer.

As can be observed from Figure 4.31, inheriting the root classifier from the fittest parent (alpha) is more common and leads to fitter offspring than from the other parent (beta). Using the root from the alpha parent amounts to 59.32% of crossovers and leads to a 21.77% average improvement rate, which is significantly higher than the 12.99% of beta.

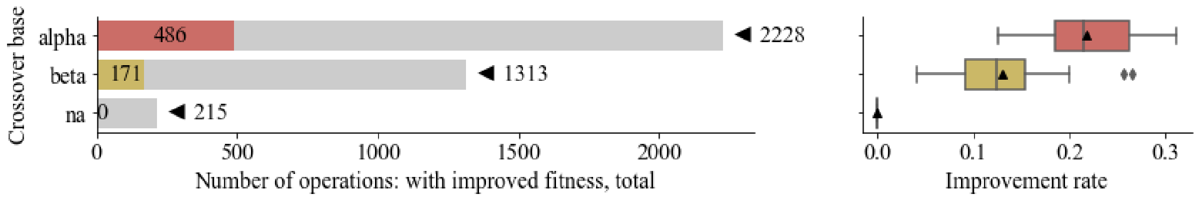


Figure 4.31 Amount and performance of crossover operations by crossover base. Breast cancer.

Branch crossovers take a bigger share (27.78%) of alpha-based operations compared to beta-based (16.53%). Branch crossovers perform significantly worse than leaf crossovers in the case of alpha-base operations but insignificantly better in the case of beta-based (Figure 4.32). Crossovers that change only hyperparameters are significantly more effective when the root classifiers are inherited from the fittest parent: 22.54% vs 12.90% mean improvement rates. Both observations correlate with the idea that the higher the share of the fittest parent is observed in the offspring, the better the latter's performance.

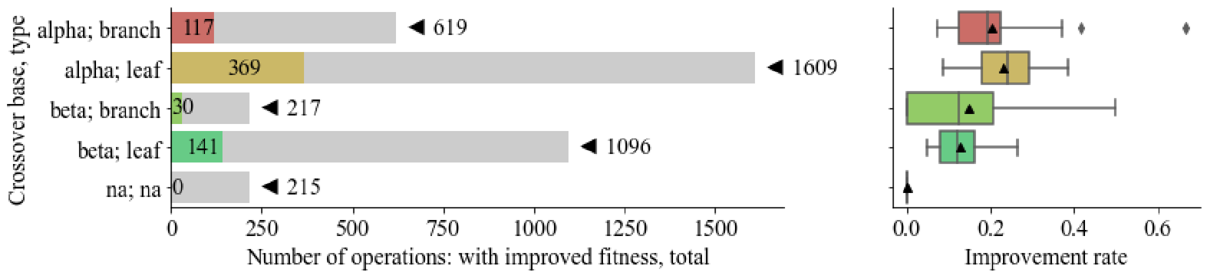


Figure 4.32 Amount and performance of crossover operations by crossover base and type. Breast cancer.

In the case of crossover, a level of operation means the edge on which the fittest parent was split to produce offspring. Crossovers with zero operational level mean that, presumably full parent tree was copied to the offspring. As expected, most operations happened in level 1 (Figure 4.33), where base classifier hyperparameters reside. The difference in performance impact of first and second-level crossovers is not significant.

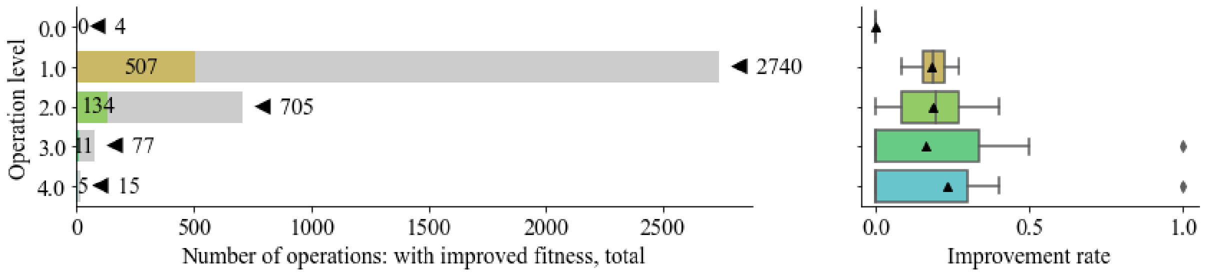


Figure 4.33 Amount and performance of crossover operations by the level of operation. Breast cancer.

4.3.2.2. Waveform

Test accuracy achieved in scenario 4 is 0.8672 on average. Internal scores of selected individuals in generation 1 vary from 0.3338 (0.01 quantile) to 0.8711 (maximum), with median and mean scores equal to 0.8711 and 0.7840. The scores of the individuals generated in generation 5 range between 0.3439 (0.01 quantile) and 0.8736, with median and mean equal to 0.8593 and 0.8447.

The vast majority of all selected individuals are of size 1 (50.16%) or 2 (42.72%), and just a few reach the size of up to 7. In generation 1, all the individuals before the operation were of a size 1 or 2, but its number decreased to 85.6% by generation 5—larger pipelines developed over generations. Individuals with 6 and 7 pipeline operators are generated in generation 4 and thus are selected only in generation 5 and are just 3 cases.

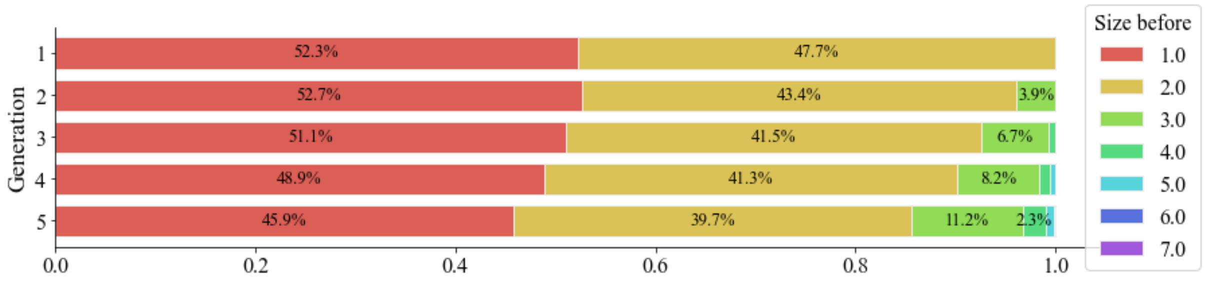


Figure 4.34 Selected individual size distribution by generation. Waveform.

Each of the 30 iterations involved about 250 operations which are roughly equally split into crossovers and mutations because of the same probability. Mutations tend to have a better chance of improving the accuracy of an individual. On average, 31.15% of individuals have a higher score after the mutation than before it. That improvement score varies from 23.53% to 40.16% in different iterations. Crossover, however, results in score improvements in 10.62% to 28.57% of cases, i.e., 18.21% on average.

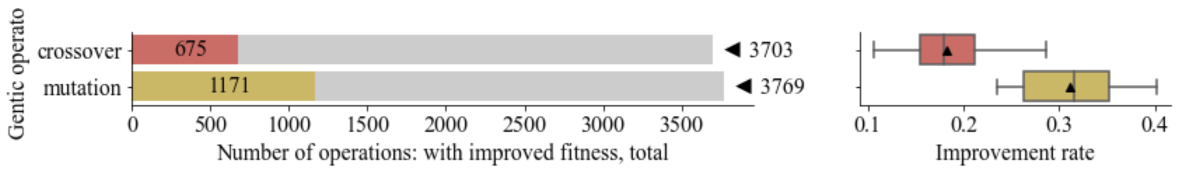


Figure 4.35 Amount and performance of operations by genetic operators. Waveform

Only one quarter (24.4%) of all chosen pipelines improve their performance after genetic operations, which also means that most operations lower fitness values. Thus, all generations' median scores after the operation are lower than before. A pool of selected individuals consists of an average of 29.5 unique pipelines, which in other cases can be as small as 21 and as large as 37. However, pipelines after an operation are all unique in most cases (95 out of 150), averaging the number of unique pipelines to 49.4. Selection of the best portion from the pool of the previous generation and its offspring ensures gradual fitness improvement that slows down as scores approach the maximum value of 1.0 (Figure 4.36).

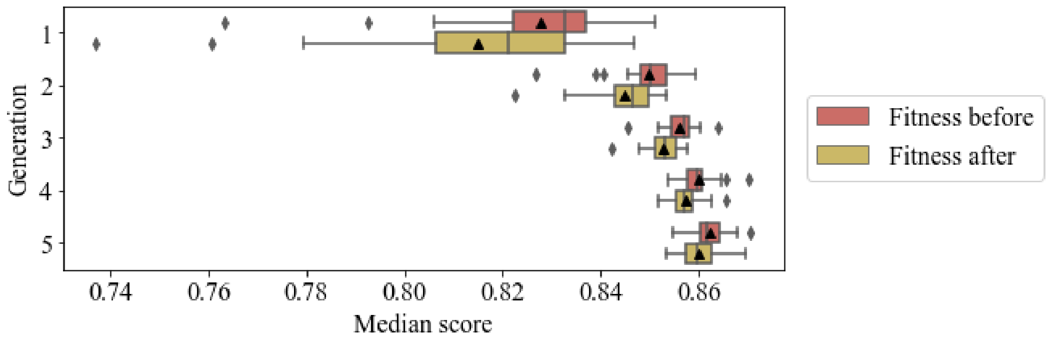


Figure 4.36 Distribution of median accuracies by the generation before and after genetic operation. Waveform.

Mutation

The improvement rate for mutation is decreasing over generations from a mean improvement of 42.31% in generation 1 to 20.93% in generation 5 (Figure 4.37). Mutation performance in generations 1 & 2, 3 & 4, are not significantly different (t-test $p_{1\&2}$=0.189, $p_{3\&4}$=0.896). Significant drops in mutation performances happen after generation 2 and generation 4.

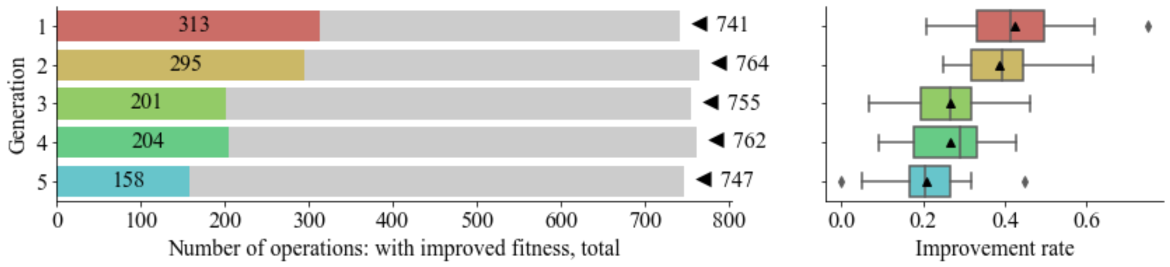


Figure 4.37 Amount and performance of mutations by pipeline generation. Waveform.

Mutation can appear in 3 different ways: change, add, and remove one node in the pipeline tree, resulting in possible pipeline change by a maximum of one unit. As seen from Figure 4.38, the majority, i.e., 1862 (49.4%) mutation operations, did not influence the pipeline size. The pipeline after mutation grew in 1621 or 43% cases, which is much more than in the case of diminished pipelines: 283 or 7.6% mutations. It is largely because almost half of the chosen individuals have only one pipeline operator. Mutations that decrease pipeline size seem to have the highest performance (35.95% average improvements rate) but might have too few observations to make a confident conclusion. Improvement rates of mutations that do not change the size of the pipelines range between 22.22% and 45.90% and average 32.82%, which is significantly higher than the performance of the mutations that increase pipeline size (t-test p-value is 0.022). The latter varies from 14.29% to 41.18% in different iterations and has a mean improvement rate of 28.19%.

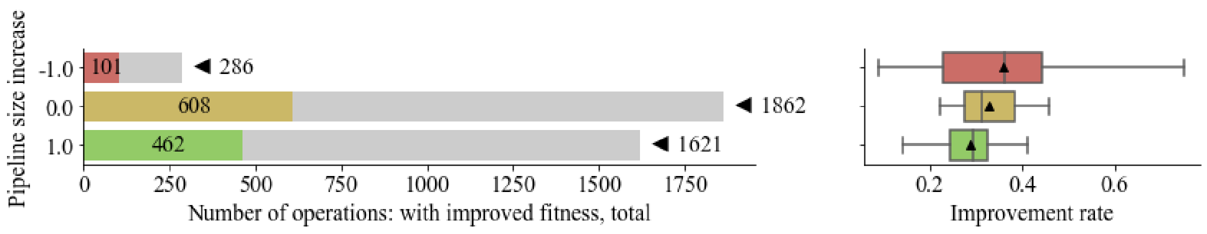


Figure 4.38 Amount and performance of mutations by pipeline size increase. Waveform.

All (but several exceptions) mutations that change a pipeline operator do not influence the size of an individual, additive mutations increase it by one, and subtractive mutation reduces it by one. Exceptions are operations that involve a pipeline element that combines two inputs (later: combiner), which is not counted as a pipeline operator in calculations of the size of the individual. Those exceptions account for 54 added, 12 removed, two switched to, and five switched from combiners. That explains the slight difference between pipeline size increase charts (Figure 4.38) and mutation type charts (Figure 4.39). Mean improvement rates for adding, changing, and removing mutations are 28.69%, 32.84%, and 36.65%, respectively. Like in the case of size increase, the changing mutation is significantly different from the additive (p=0.019).

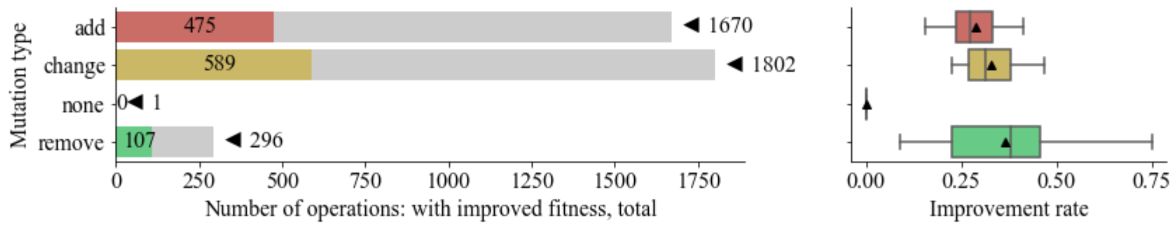


Figure 4.39 Amount and performance of mutation operations by type. Waveform.

Pre-processor and classifier are the most added pipeline components, amounting to 83.17% of all additive mutations (Figure 4.40). Adding a classifier contributes more to an individual's performance—its average improvement rate is 32.6%, which is significantly higher than the preprocessor's 20.9% (p-value<0.001). Adding a selector result on average in the highest 42.3% improvement rate, however, might lack observations to make confident conclusions.

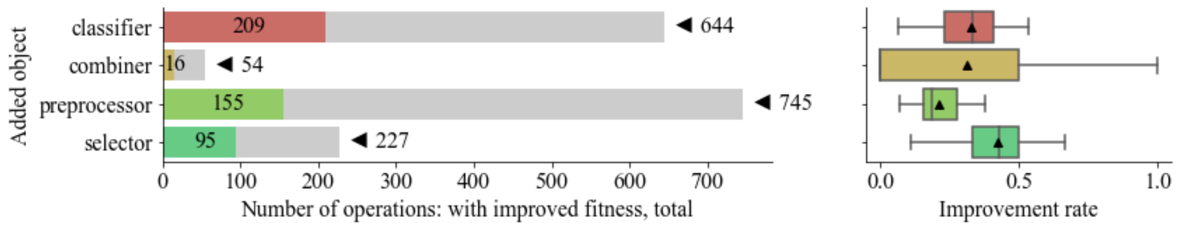


Figure 4.40 Amount and performance of mutation operations by the added object. Waveform.

Pipeline operators change in 13 different ways, but Figure 4.41 shows only three with more than 20 observations. The full version of the chart can be seen in Appendix 3. 1380 (or 76.5%) of all change mutations are the change in hyperparameter value, which also accounts for 36.6% of all mutations. Hyperparameter adjustment improves the accuracy in 34.5% of cases on average. Switch of classifiers is the second most common change, which amounts to 338 operations or 18.7% of change mutations. Its average improvement rate is 26.1%. The rest of the changes do not have enough observations to conclude their effectiveness.

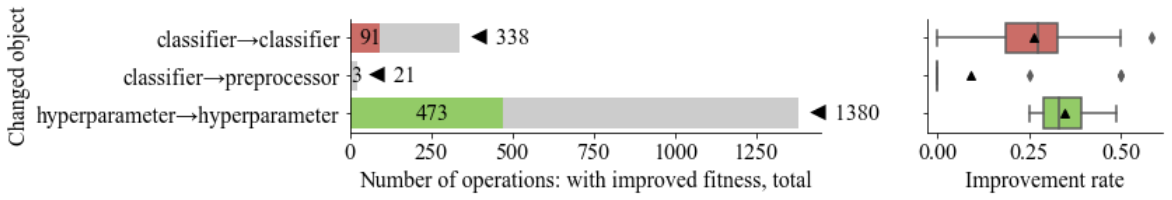


Figure 4.41 Amount and performance of mutation operations by the changed object (short). Waveform.

Removing mutation is the least frequent type of mutation. Its most common objects are the classifier and pre-processor (Figure 4.42), like in the case of additive mutation. Removing an extra classifier leads to fitness improvement in 46.11% of cases, which is significantly greater than removing a pre-processor (30.71%).

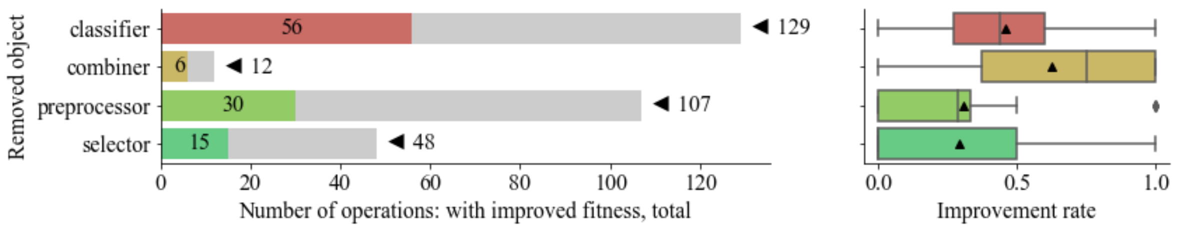


Figure 4.42 Amount and performance of mutation operations by the removed object. Waveform.

Change of the root classifier is the only change possible on level 0. It improves the performance of an individual in 25.98% of cases on average. Although root classifier changes in pipelines of any depth, it is most common in depth 1 (229 of 321 cases) because of the fewer mutation options small pipelines can have. Most of the changes happen on level 1 (Figure 4.43) because that is where root classifier hyperparameters reside, which is known as one of the most frequently changed parts of the pipeline. It is also where all the additions happen in the pipelines of depth 1, as well as all removals and most of the additions and changes in pipelines with depth 2.

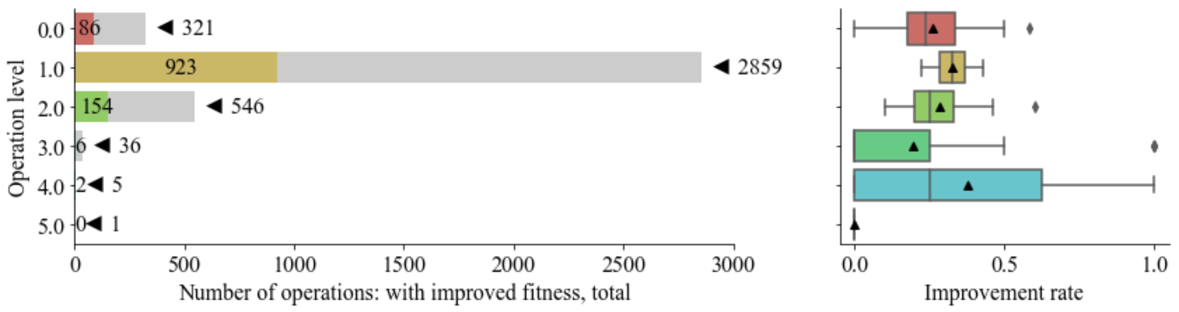


Figure 4.43 Amount and performance of mutation operations by the level of operation. Waveform.

Crossover

The improvement rate for crossover is slightly decreasing over generations, from a mean improvement of 21.10% in generation 1 to 14.96% in generation 5. It takes three generations for crossover performance to decrease significantly (t-test p-value$_{1\&4}$=0.007, p-value$_{2\&5}$=0.015).

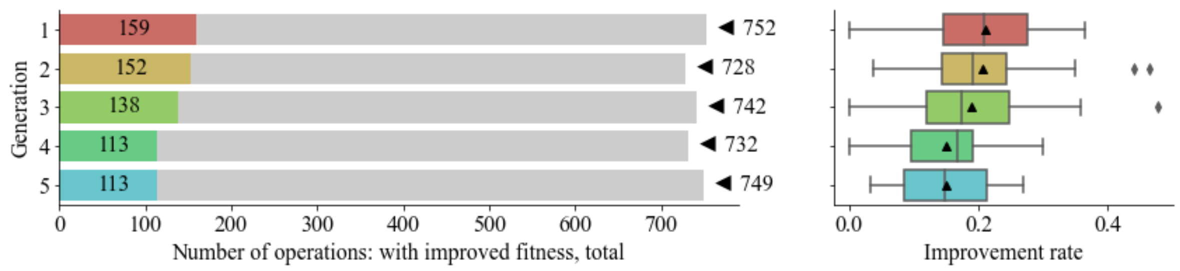


Figure 4.44 Amount and performance of crossover operations by generation. Waveform.

While mutation can result in maximum pipeline change by one unit, crossover impact on the pipeline size reaches up to 3 units to the positive and 4 to the negative direction (Figure 4.45). Nevertheless, most operations did not change the size of the pipeline. The number of pipeline operators stayed the same in 4367, 58.44% of total cases, and 2505 (66.46%) crossover operations.

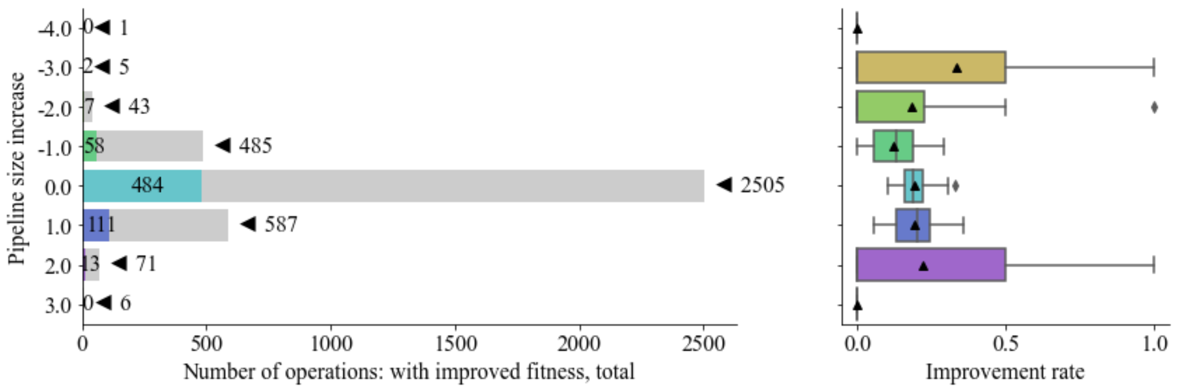


Figure 4.45 Amount and performance of crossover operations by pipeline size increase. Waveform.

Most (71.4%) of the crossovers happened on a leaf node level (Figure 4.46), which is only possible for hyperparameter nodes. It is the only option for small (i.e., of size 1) pipelines, which amount to almost half of the population. Mean improvement rates for branch and leaf crossovers are 17.94% and 19.59%, respectively, and are not significantly different. In 217 cases (or 5.9%%), the type of crossovers cannot be identified because either: individuals (before and after a crossover) are equal or have no common pipeline operators.

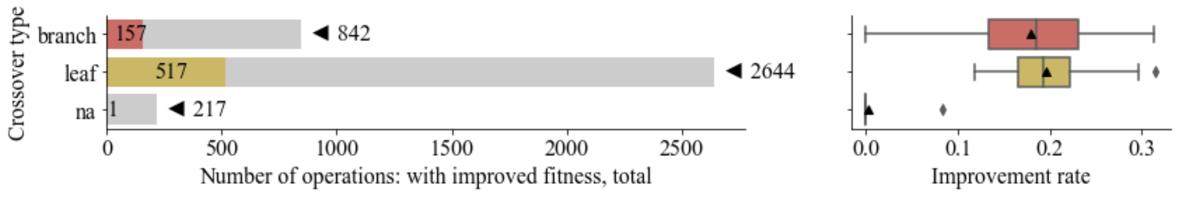


Figure 4.46 Amount and performance of crossover operations by crossover type. Waveform.

As can be observed from Figure 4.47, inheriting the root classifier from the fittest parent (alpha) is more common and leads to fitter offspring than from the other parent (beta). Using the root from the alpha parent amounts to 61.22% of crossovers and leads to a 21.71% average improvement rate, which is significantly higher than the 15.18% improvement rate of beta.

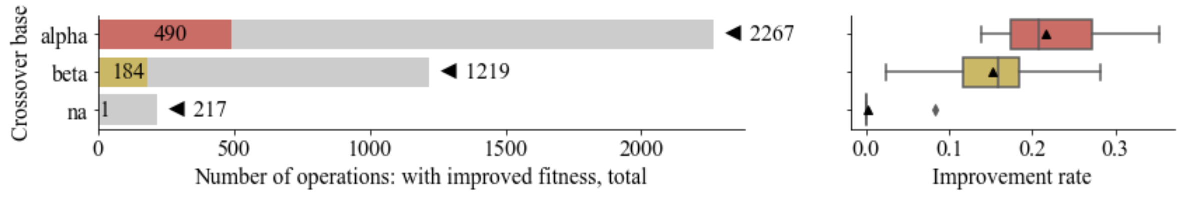


Figure 4.47 Amount and performance of crossover operations by crossover base. Waveform.

Branch crossovers take a bigger share (28.1%) of alpha-based operations than beta-based (16.9%). Branch crossovers perform significantly worse than leaf crossovers in the case of alpha-base operations (p=0.002) but insignificantly better in the case of beta-based (p=0.7) (Figure 4.48). Crossovers that change only hyperparameters are significantly more effective when the root classifiers are inherited from the fittest parent: 23.28% vs 13.5% mean improvement rates. Both observations correlate with the idea that the higher the share of the fittest parent is observed in the offspring, the better the latter's performance.

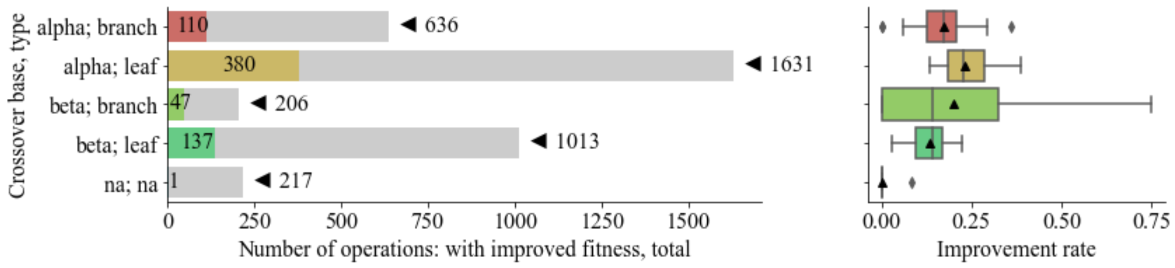


Figure 4.48 Amount and performance of crossover operations by crossover base and type. Waveform.

In the case of crossover, a level of operation means the edge on which the fittest parent was split to produce offspring. Crossovers with zero operational level mean that, presumably full parent tree was copied to the offspring. As expected, most operations happened in level 1 (Figure 4.49), where base classifier hyperparameters reside. The difference in performance impact of first and second-level crossovers is not significant (p=0.576).

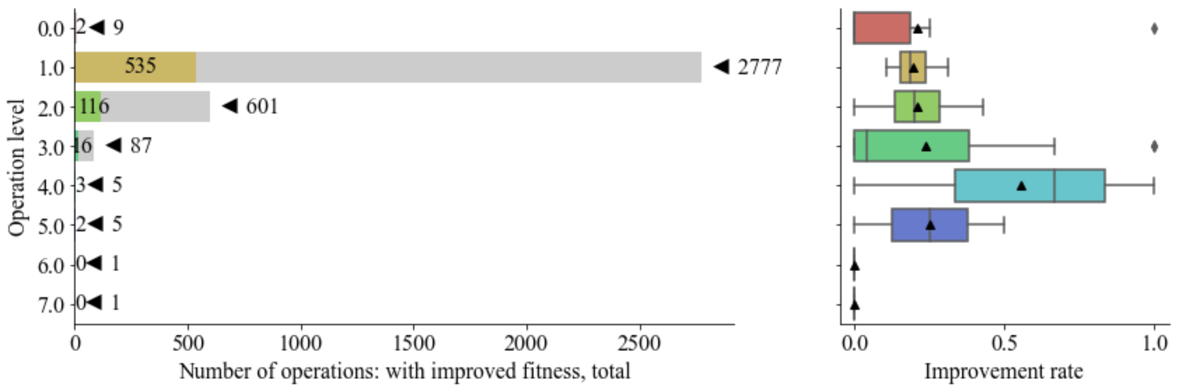


Figure 4.49 Amount and performance of crossover operations by the level of operation. Waveform.

4.3.2.3. Bioavailability

Test negative RMSE achieved in scenario 4 is -27.731 on average. Internal scores of selected individuals in generation 1 vary from $-2.17 \cdot 10^{23}$ (0.01 quantile) to -724.15 (maximum), with median and mean scores equal to -908.29 and $-1.56 \cdot 10^{36}$. The scores of the individuals generated in generation 5 range between -1507.29 (0.01 quantile) and -713.76 (max), with median and mean equal to -780.34 and $-1.11 \cdot 10^{36}$.

The vast majority of all selected individuals are of a size 1 (50.11%) or 2 (42.78%) and reach a size of up to 4. In generation 1, all the individuals before the operation were of a size 1 or 2. Still, its total number decreases to 91.6% by generation 5 due to the drop in the number of pipelines of size 2. The number of one-operator pipelines is, on the contrary, increasing over generations. Although pipelines of size five are getting generated by crossovers and mutations since generation 2, they are never selected, and the largest size of the individuals remains equal to 4.

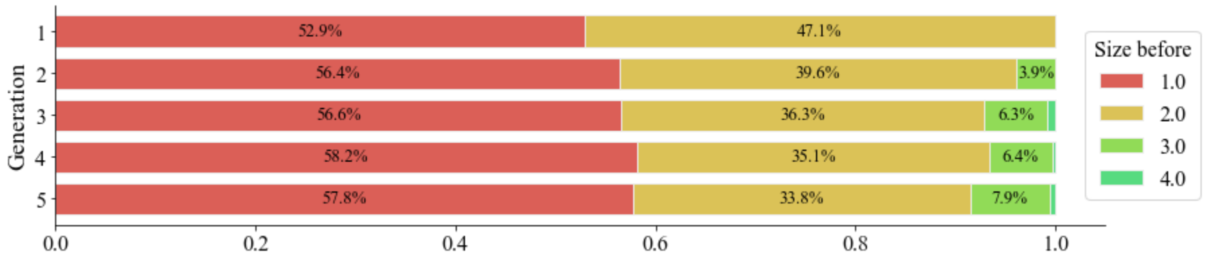


Figure 4.50 Selected individual size distribution by generation. Bioavailability.

Each of the 30 iterations involved about 250 operations which are roughly equally split into crossovers and mutations because of the same probability. Mutations tend to have a better chance of improving the accuracy of an individual. On average, 30.86% of individuals have a higher score after the mutation than before it (Figure 4.51). That improvement score varies from 21.26% to 38.97% in different iterations. Crossover, however, results in score improvements in 12.90% to 27.83% of cases, i.e., 20.79% on average.

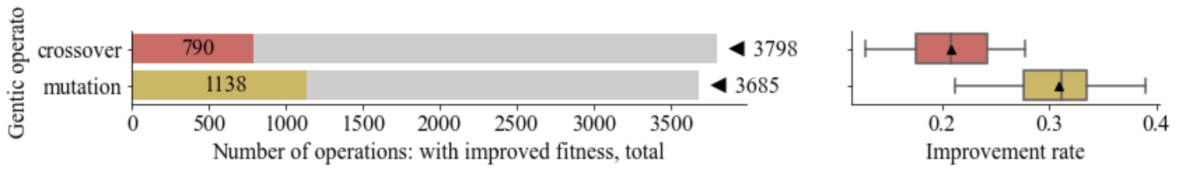


Figure 4.51 Amount and performance of operations by genetic operators. Bioavailability.

Only one quarter (25.76%) of all chosen pipelines improve their performance after genetic operations, which also means that most operations lower fitness values. Thus, all generations' median scores after the operation are lower than before. A pool of selected individuals consists of an average of 29.5 unique pipelines, which in other cases can be as small as 22 and as large as 37. However, pipelines after an operation are all unique in most cases (101 of 150),

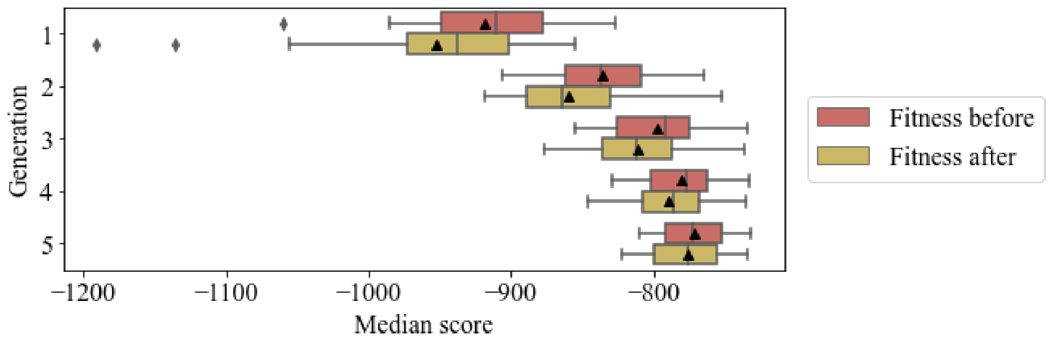


Figure 4.52 Distribution of median accuracies by the generation before and after genetic operation. Waveform.

averaging the number of unique pipelines to 49.4. Selection of the best portion from the pool of the previous generation and its offspring ensures gradual fitness improvement that slows down as scores approach the maximum value of 0 (Figure 4.52).

Mutation

The improvement rate for mutation is decreasing over generations from a mean improvement of 43.84% in generation 1 to 23.41% in generation 5 (Figure 4.53). The drop in the operator performance is sharper until generation 3 and less distinguished between generations 3 and 5. Mutation performance in generations 1&2 and 2&3 are significantly different at 0.05 significance level, and 3&5 are significantly different at 0.1 significance level (p=0.065).

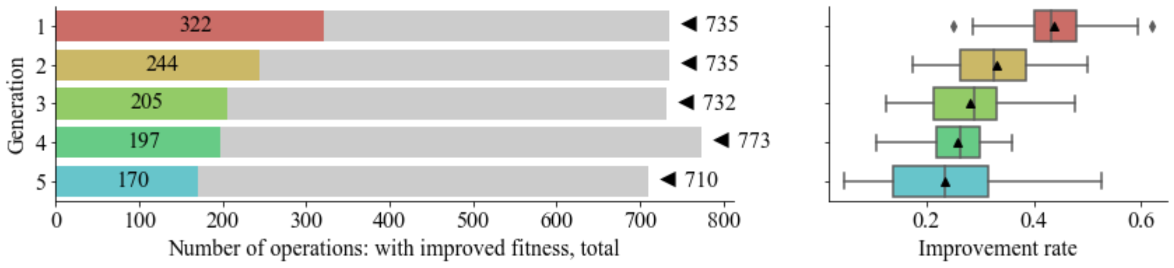


Figure 4.53 Amount and performance of mutations by generation. Bioavailability.

Mutation can appear in 3 different ways: change, add, and remove one node in the pipeline tree, resulting in possible pipeline change by a maximum of one unit. As seen from Figure 4.54, the majority, i.e., 1870 (50.75%) mutation operations, did not influence the pipeline size. The pipeline after mutation grew to 1566 or 42.50% cases which is much more than diminished cases: 249 or 6.75% mutations. It is largely because almost half of the chosen individuals have only one pipeline operator. Size-increasing mutation improvement rate range between 11.54% and 42.86%, with a mean equal to 27.50%, which is significantly lower than the performance of size-unchanging mutation, which ranges from 19.70% to 43.55% and averages 31.79%. The mean improvement rate of size-decreasing mutations is 45.04% (and ranges from 0 to 1 in different iterations). Its performance seems the highest, but 249 observations might be too few to have a confident conclusion.

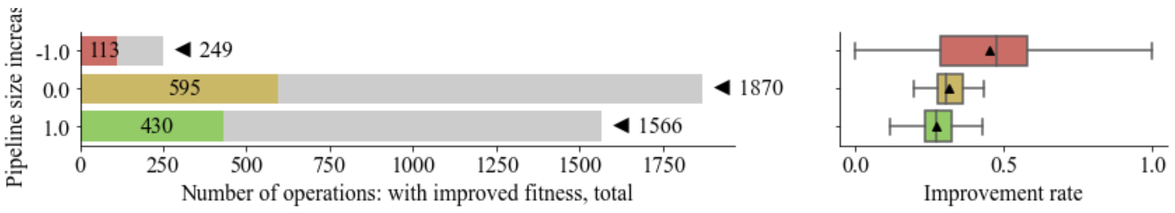


Figure 4.54 Amount and performance of mutations by pipeline size increase. Bioavailability.

All (but several exceptions) mutations that change a pipeline operator do not influence the size of an individual, additive mutations increase it by one, and subtractive mutation reduces it by one. Exceptions are operations that involve a pipeline element that combines two inputs (later: combiner), which is not counted as a pipeline operator in calculations of the size of the individual. Those exceptions account for 76 added, 11 removed, three switched to, and seven switched from combiners. That explains the slight difference between pipeline size increase charts (Figure 4.54) and mutation type charts (Figure 4.55). Mean improvement rates for adding, changing, and removing mutations are 27.69%, 31.67%, and 45.12%, respectively. Like in the case of size increase, all mutation types are significantly different in their performance.

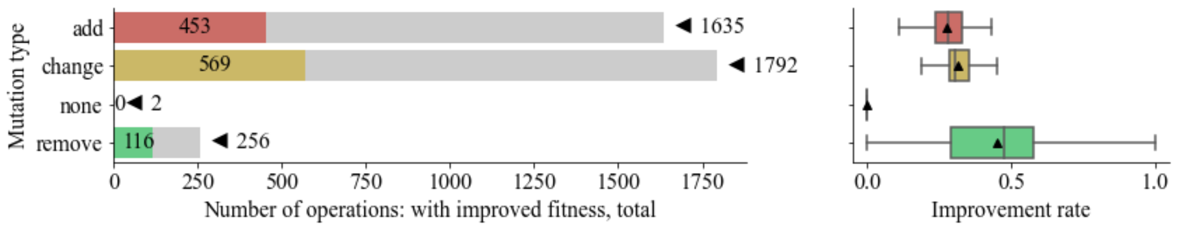


Figure 4.55 Amount and performance of mutation operations by type. Bioavailability.

Pre-processor and classifier are the most added pipeline components, amounting to 86.79% of all additive mutations (Figure 4.56). Adding a selector, however, contributes the most to an individual's performance—its average improvement rate is 36.75%, but it might have too few observations to make a confident conclusion. Performances of adding a preprocessor and a regressor are not significantly different, and their average improvement rates are equal to 25.81% and 27.0%, respectively.

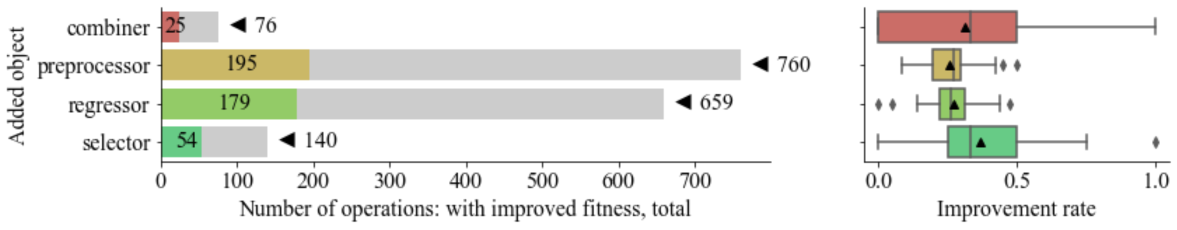


Figure 4.56 Amount and performance of mutation operations by the added object. Bioavailability.

Pipeline operators change in 13 different ways, but Figure 4.57 shows only three with more than 20 observations. The full version of the chart can be seen in Appendix 4. 1254 (or 73%) of all change mutations are the change in hyperparameter value, which accounts for a third of all mutations. Hyperparameter adjustment improves the accuracy in 34.94% of cases on average. Switch of regressors is the second most common change, which amounts to 435 operations or 25% of change mutations. Its average improvement rate is 25.3%. The rest of the changes do not have enough observations to conclude their effectiveness.

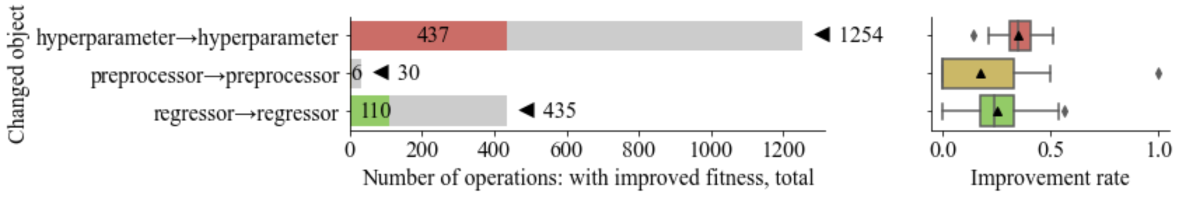


Figure 4.57 Amount and performance of mutation operations by the changed object (short). Bioavailability.

Removing mutation is the least frequent type of mutation. Its most common objects are the pre-processor and regressor (Figure 4.58), like in the case of additive mutation. Removing extra regressor leads to fitness improvement in 49.41% of cases, which is greater than removing the pre-processor (40.12%). The difference, however, is not significant, and there might be too few observations to make a confident conclusion.

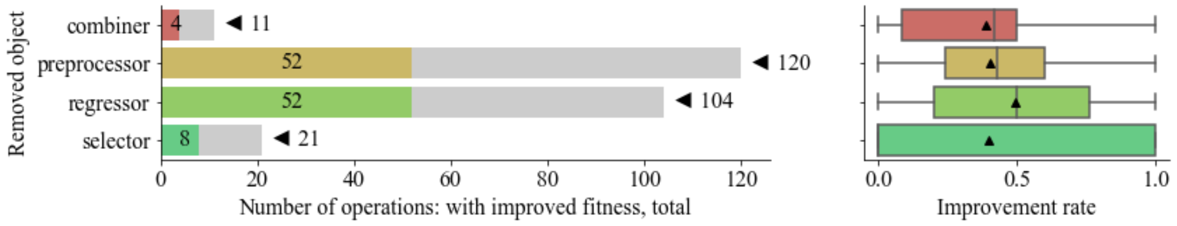


Figure 4.58 Amount and performance of mutation operations by the removed object. Bioavailability.

Change of the root classifier is the only change possible on level 0. It improves the performance of an individual in 24.83% of cases on average. Although root classifier changes in pipelines of any depth, it is most common in the pipelines of depth 1 (329 of 427 cases) because of the fewer mutation options small pipelines can have. As Figure 4.59 shows, most of the changes happen on level 1 (2717 of 3683 cases) because that is where root classifier hyperparameters reside, which is known as one of the most frequently changed parts of the pipeline. It is also where all the additions happen in the pipelines of depth 1, as well as all removals and most of the additions and changes in pipelines with depth 2.

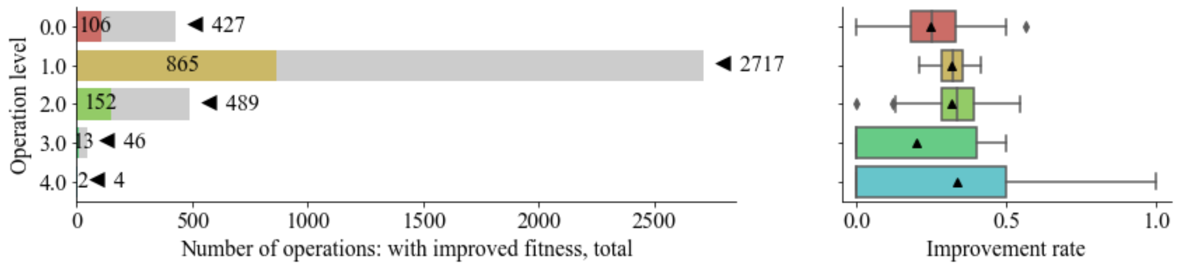


Figure 4.59 Amount and performance of mutation operations by the level of operation. Bioavailability.

Crossover

Figure 4.60 shows that the crossover operations' performance does not follow any pattern. Average improvement rates for crossover vary from 24.63% in generation 3 to 17.18% in generation 5. Crossover performances in generations 1&3, 2&4, and 2&5 are not significant. The largest performance drop happens after generation 1. The other decrease in performance is less steep and takes two generations to become significant.

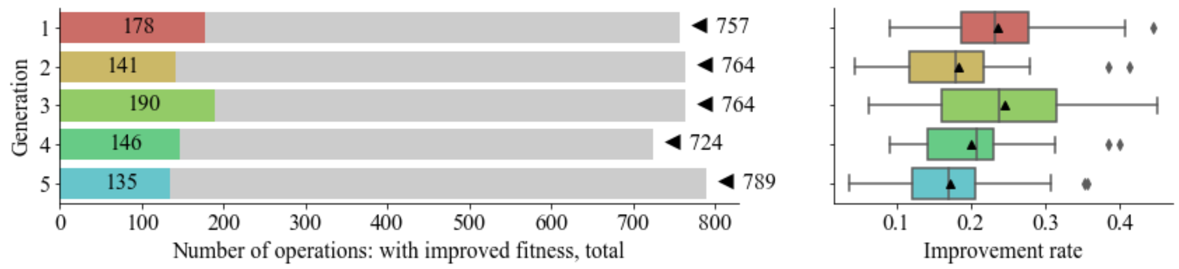


Figure 4.60 Amount and performance of crossover operations by generation. Bioavailability.

While mutation can result in maximum pipeline change by one unit, crossover impact on the pipeline size reaches up to 3 units difference in either direction (Figure 4.61). Nevertheless, most operations did not change the size of the pipeline. The number of pipeline operators stayed the same in 4485 or 59.94% of total cases and 2615 (69.02%) crossover operations. Its

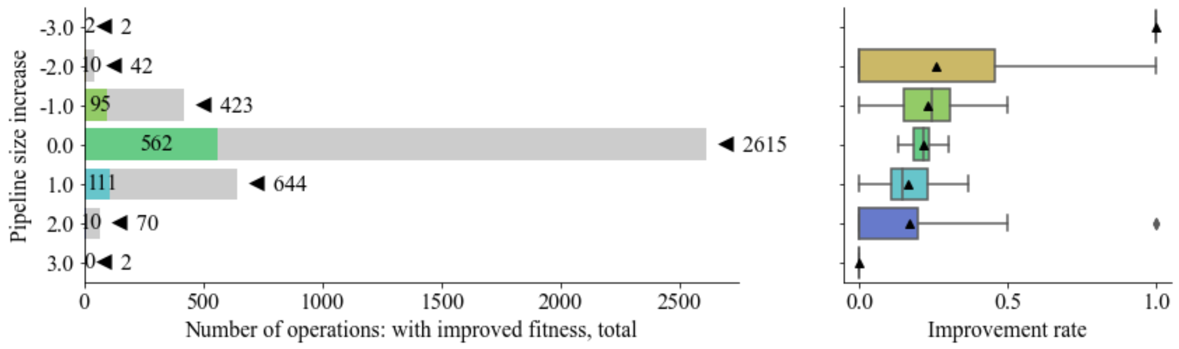


Figure 4.61 Amount and performance of crossover operations by pipeline size increase. Bioavailability.

average improvement rate is 21.5%, which is significantly higher than crossovers that increase the size by 1 but not significantly different from the performance of crossovers that decrease the size by 1. Other cases have too few observations to make confident conclusions.

Most (72%) of the crossovers happened on a leaf node level (Figure 4.62), which is only possible for hyperparameter nodes. It is the only option for small (i.e., of size 1) pipelines, which amount to almost half of the population. Mean improvement rates for branch and leaf crossovers are 20.67% and 22.56%, respectively, and are not significantly different (p=0.291). In 229 cases (or 6%), the type of crossovers cannot be identified because either both individuals (before and after a crossover) are equal or have no common pipeline operators.

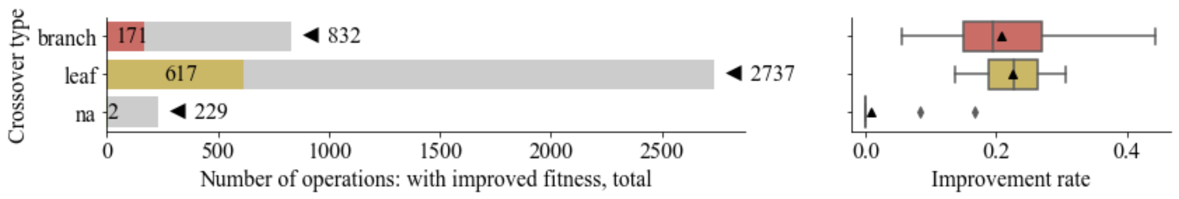


Figure 4.62 Amount and performance of crossover operations by crossover type. Bioavailability.

As can be observed from Figure 4.63, inheriting the root classifier from the fittest parent (alpha) is more common and leads to fitter offspring than from the other parent (beta). Using the root from the alpha parent amounts to 64.3% of crossovers and leads to a 24.56% average improvement rate, which is significantly higher than the 16.75% of beta.

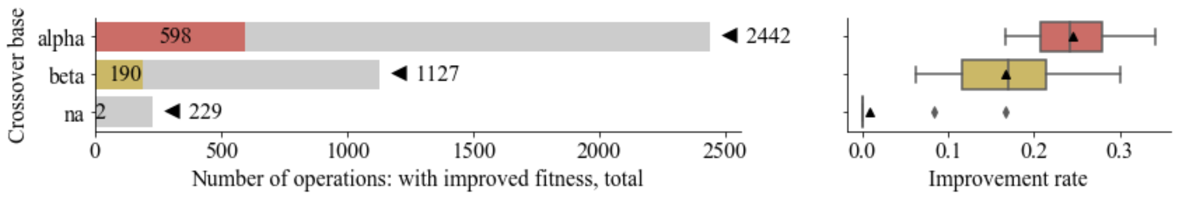


Figure 4.63 Amount and performance of crossover operations by crossover base. Bioavailability.

Branch crossovers take a bigger share (27.4%) of alpha-based operations than beta-based (14.46%). Branch crossovers perform significantly (α=0.1) worse than leaf crossovers in the case of alpha-base operations (p=0.061) and significantly (α=0.1) better in the case of beta-based (p=0.097) (Figure 4.64). Crossovers that change only hyperparameters are significantly more effective when the root classifiers are inherited from the fittest parent: 25.66% vs 16.95% mean improvement rates. Both observations correlate with the idea that the higher the share of the fittest parent is observed in the offspring, the better the latter's performance.

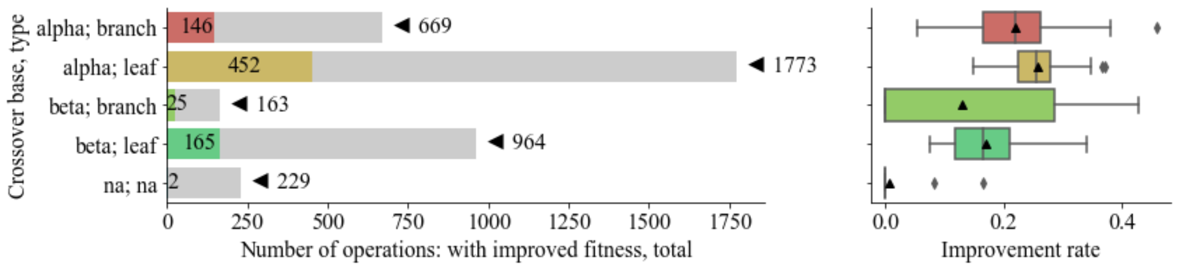


Figure 4.64 Amount and performance of crossover operations by crossover base and type. Bioavailability.

In the case of crossover, a level of operation means the edge on which the fittest parent was split to produce offspring. Crossovers with zero operational level mean that, presumably full parent tree was copied to the offspring. As expected, most operations happened in level 1 (Figure 4.64), where base classifier hyperparameters reside. The difference in performance impact of first and second-level crossovers is not significant (p=0.819).

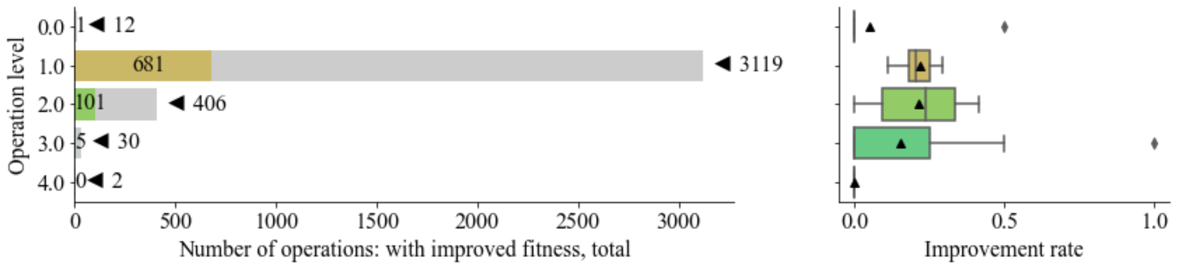


Figure 4.65 Amount and performance of crossover operations by the level of operation. Bioavailability.

4.3.2.4. Concrete

Test negative RMSE achieved in scenario 4 is -4.399 on average. Internal scores of selected individuals in generation 1 vary from $-5.08 \cdot 10^{29}$ (0.01 quantile) to -18.10 (maximum), with median and mean scores equal to -86.54 and $-3.12 \cdot 10^{33}$. The scores of the individuals generated in generation 5 range between -282.3 (0.01 quantile) and -17.57, with median and mean equal to -30.21 and $-9.48 \cdot 10^{32}$.

The vast majority of all selected individuals are of size 1 (42.78%) or 2 (50.11%) and reach a size of up to 4 (Figure 4.66). In generation 1, all the individuals before the operation were of size 1 or 2, but its number decreased to 81.6% by generation 5. The number of two-operator individuals decreased to a larger extent than that of one-operator individuals. By generation 5, they take the same share of the population (on average). Larger pipelines (sizes 5 and 6) have been developed since generation 3 but were never selected for further mutations or crossover (as parent 1). Size 4 remains the largest since generation 3 and amounts to 34 observations in generation 5.

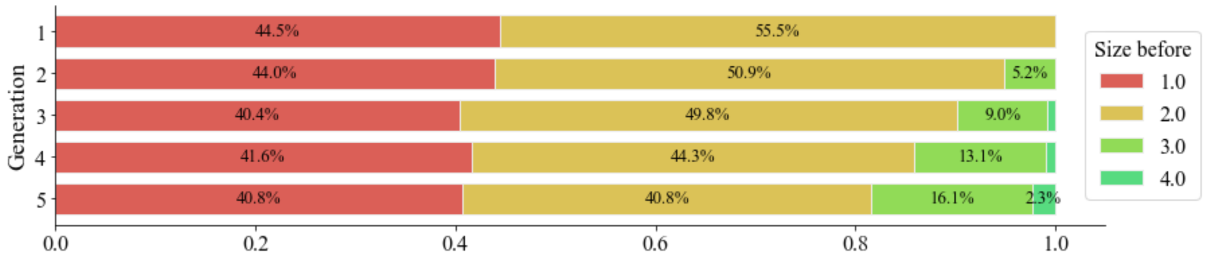


Figure 4.66 Selected individual size distribution by generation. Concrete.

Each of the 30 iterations involved about 250 operations which are roughly equally split into crossovers and mutations because of the same probability. Mutations tend to have a better chance of improving the accuracy of an individual. On average, 36.26% of individuals have a higher score after the mutation than before it (Figure 4.67). That improvement score varies from 25.0% to 46.03% in different iterations. Crossover, however, results in score improvements in 14.42% to 31.67% of cases, i.e., 23.04% on average.

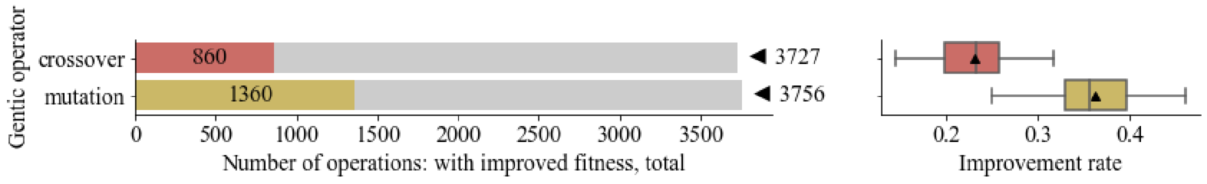


Figure 4.67 Amount and performance of operations by genetic operators. Concrete.

Only one-quarter of all chosen pipelines improve their performance after genetic operations, which also means that most operations lower fitness values. Thus, all generations' median scores after the operation are lower than before. A pool of selected individuals consists of an average of 29.3 unique pipelines, which in other cases can be as small as 21 and as large as 35. However, pipelines after an operation are all unique in most cases (102 of 150), averaging

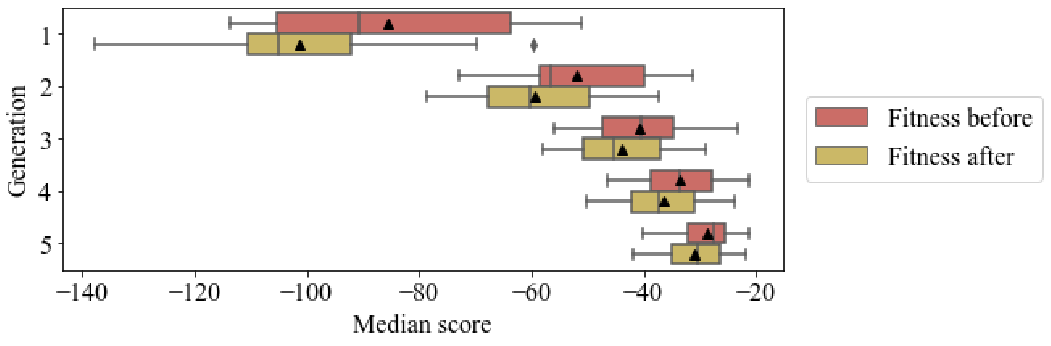


Figure 4.68 Distribution of median accuracies by the generation before and after genetic operation. Concrete.

the number of unique pipelines to 49.5. Selection of the best portion from the pool of the previous generation and its offspring ensures gradual fitness improvement that slows down as scores approach the maximum value of 0 (Figure 4.68).

Mutation

The improvement rate for mutation is decreasing over generations from a mean improvement of 44.29% in generation 1 to 31.23% in generation 5 (Figure 4.69). Average improvement rates for mutation in generations 1&2, 2&3, 3&4, 4&5 are not significantly different (t-test p-values 0.176, 0.089, 0.309, 0.783), but performance between 1&3 and 2&4 are significantly different. So, the decrease in performance after generation 3 is negligible.

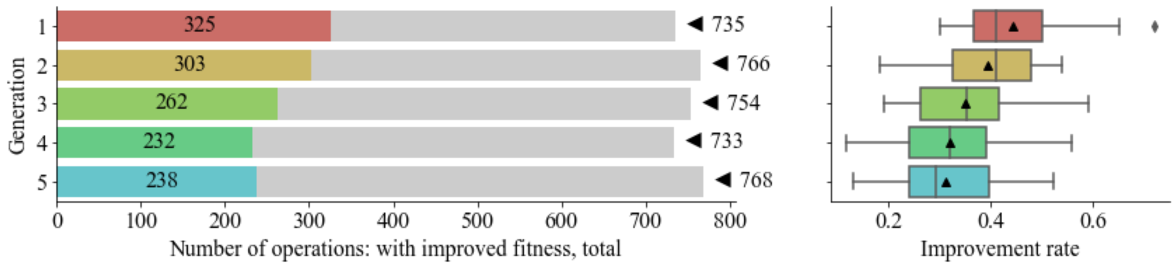


Figure 4.69 Amount and performance of mutations by generation. Concrete.

Mutation can appear in 3 different ways: change, add, and remove one node in the pipeline tree, resulting in possible pipeline change by a maximum of one unit. As seen from Figure 4.70, the majority, i.e., 1780 (47.39%) mutation operations, did not influence the pipeline size. The pipeline after mutation grew to 1655 or 44.06% cases which is much more than diminished cases: 321 or 8.55% mutations. It is largely because almost half of the chosen individuals have only one pipeline operator. Pipelines that have increased as a result of the mutation have, on average, the highest improvement rate, which ranges between 25.0% and 58.0%, with a mean equal to 40.3%. However, same and decreased size mutation performances are significantly lower than size-increasing mutation but are not significantly different compared to each other and have average rates of 33.2% and 30.9%, respectively.

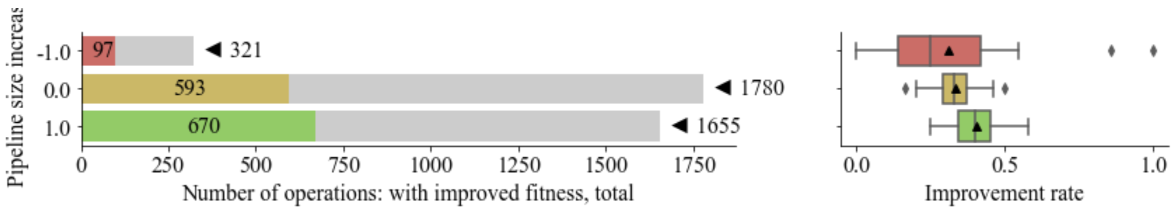


Figure 4.70 Amount and performance of mutations by pipeline size increase. Concrete.

All (but several exceptions) mutations that change a pipeline operator do not influence the size of an individual, additive mutations increase it by one, and subtractive mutation reduces it by one. Exceptions are operations that involve a pipeline element that combines two inputs (later: combiner), which is not counted as a pipeline operator in calculations of the size of the individual. Those exceptions account for 57 added, eight removed, eight switched to, and two switched from combiners. That explains the slight difference between pipeline size increase charts (Figure 4.70) and mutation type charts (Figure 4.71). Mean improvement rates for adding, changing, and removing mutations are 40.33%, 33.14%, and 29.57%, respectively. Additive mutation performs significantly better than subtractive and changing. However, the difference between the performances of the two later is not statistically significant.

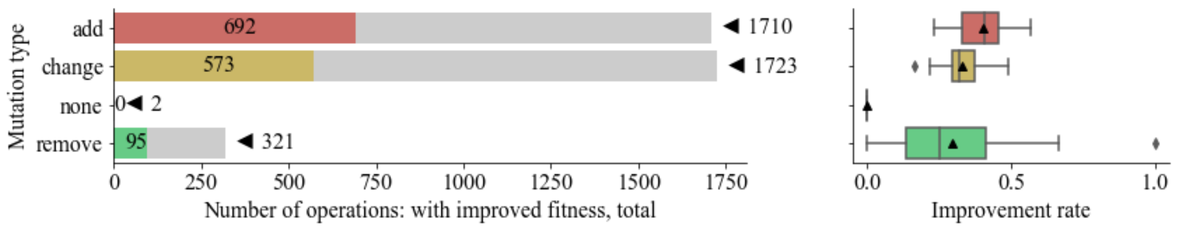


Figure 4.71 Amount and performance of mutation operations by type. Concrete.

Pre-processor and regressor are the most added pipeline components, amounting to 86.2% of all additive mutations (Figure 4.72). Adding a regressor, however, contributes the most to an individual's performance—its average improvement rate is 63.1%, which is significantly higher than the selector's average of 18.17% and the preprocessor's 25.66%.

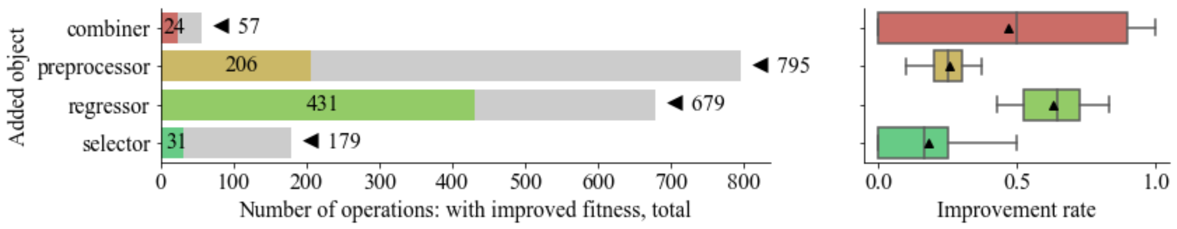


Figure 4.72 Amount and performance of mutation operations by the added object. Concrete.

Pipeline operators change in 15 unique ways, but Figure 4.73 shows only three with at least 20 observations. The full version of the chart can be seen in Appendix 5. 1254 (or 73%) of all change mutations are the change in hyperparameter value, which accounts for a third of all mutations. Hyperparameter adjustment improves the accuracy in 34.98% of cases on average. Switch of regressors is the second most common change, which amounts to 371 operations or 21.5% of change mutations. Its average improvement rate is 27.89%. The rest of the changes do not have enough observations to conclude their effectiveness.

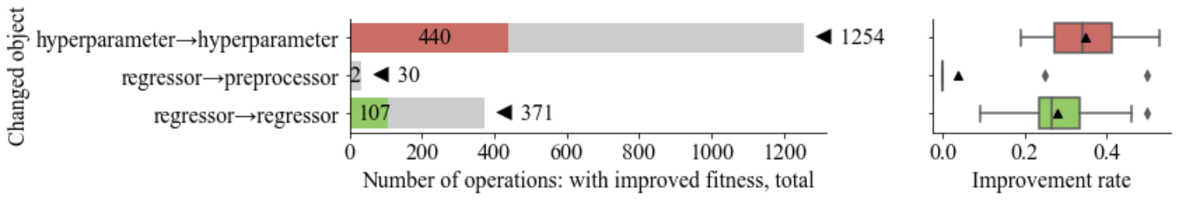


Figure 4.73 Amount and performance of mutation operations by the changed object (short). Concrete.

Removing mutation is the least frequent type of mutation. Its most common objects are the regressor and pre-processor (Figure 4.74), like in the case of additive mutation. Removing extra regressor leads to fitness improvement in 19.42% of cases, which is less than removing the pre-processor (50.42%). Although the difference is significant, both cases might lack observations to make confident conclusions.

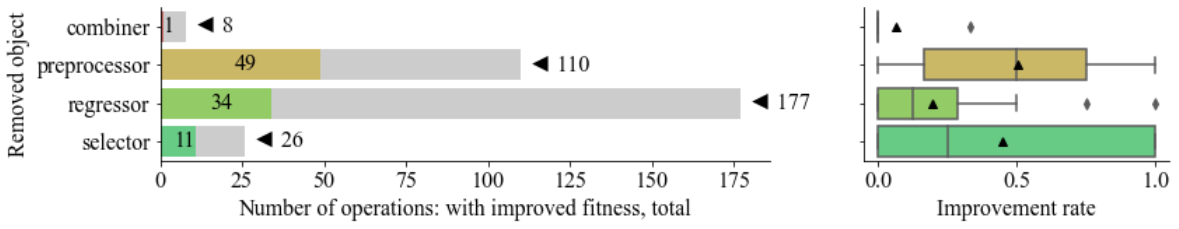


Figure 4.74 Amount and performance of mutation operations by the removed object. Concrete.

Change of the root classifier is the only change possible on level 0. It improves the performance of an individual in 27.62% of cases on average. Although root classifier changes in pipelines of any depth, it is most common in depth 1 (234 of 348 cases) because of the fewer mutation options small pipelines can have. Most of the changes happen on level 1 (Figure 4.75) because that is where root classifier hyperparameters reside, which is known as one of the most frequently changed parts of the pipeline. It is also where all the additions happen in the pipelines of depth 1, as well as all removals and most of the additions and changes in pipelines with depth 2.

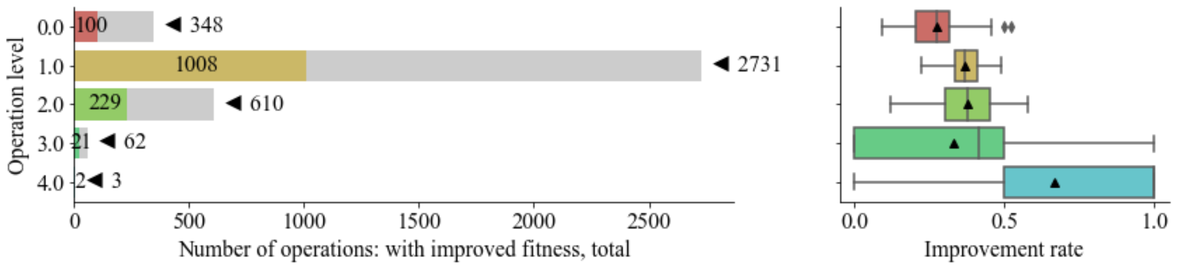


Figure 4.75 Amount and performance of mutation operations by the level of operation. Concrete.

Crossover

On average, the crossover improvement rate is 20.95% in generation 1 (Figure 4.76). It increases during two generations and reaches an average of 24.83% in generation 3. It is followed by a two-generation-long decrease in performance, ending in an average of 21.30%. However, none of those changes is statistically significant. The T-test p-value for the improvement rate distributions in the most diverse generations, i.e., 1 and 3, is 0.103.

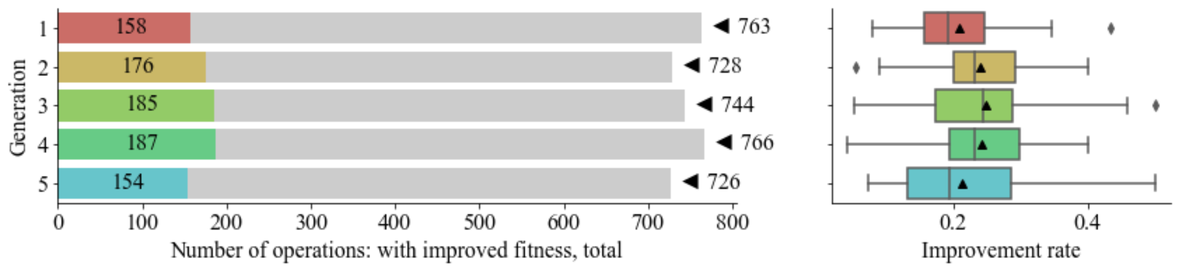


Figure 4.76 Amount and performance of crossover operations by generation. Concrete.

While mutation can result in maximum pipeline change by one unit, crossover impact on the pipeline size reaches up to 3 units difference in either direction (Figure 4.77). Nevertheless, most operations did not change the size of the pipeline. The number of pipeline operators stayed the same in 4315 or 57.66% of total cases and 2535 (68.02%) crossover operations.

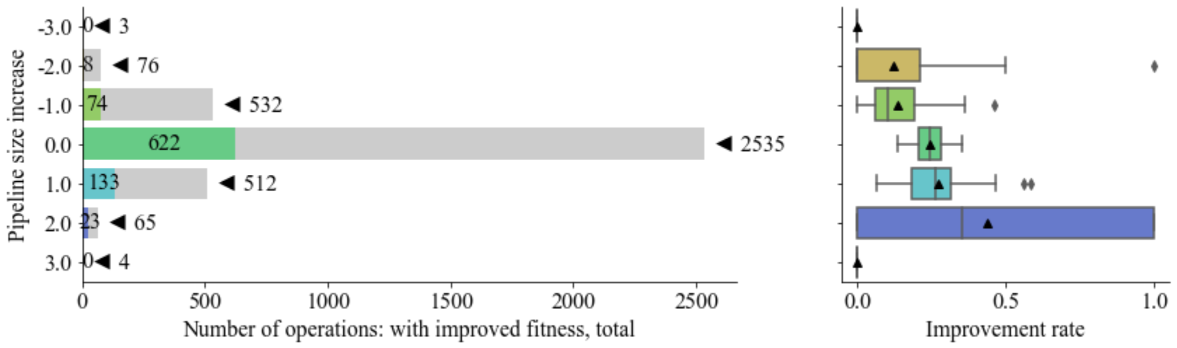


Figure 4.77 Amount and performance of crossover operations by pipeline size increase. Concrete.

Most (71.7%) of the crossovers happened on a leaf node level (Figure 4.78), which is only possible for hyperparameter nodes. It is the only option for small (i.e., of size 1) pipelines, which amount to almost half of the population. Mean improvement rates for branch and leaf crossovers are 25.67% and 24.12%, respectively, and are not significantly different. In 192 cases (or 5.15%), the type of crossovers cannot be identified because either both individuals (before and after a crossover) are equal or have no common pipeline operators.

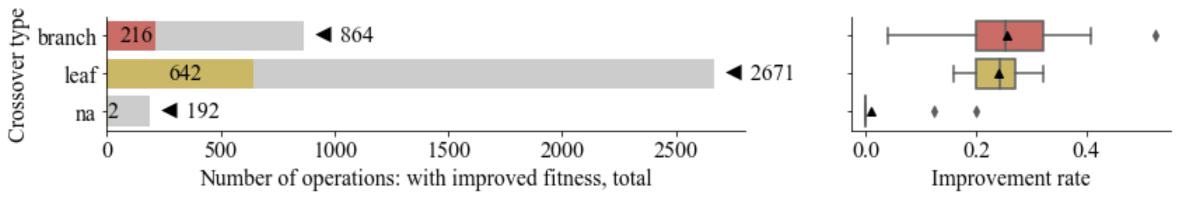


Figure 4.78 Amount and performance of crossover operations by crossover type. Concrete.

As can be observed from Figure 4.79, inheriting the root classifier from the fittest parent (alpha) is more common and leads to fitter offspring than from the other parent (beta). Using the root from the alpha parent amounts to 61.6% of crossovers and leads to a 28.92% average improvement rate, which is significantly higher than the 15.57% of beta.

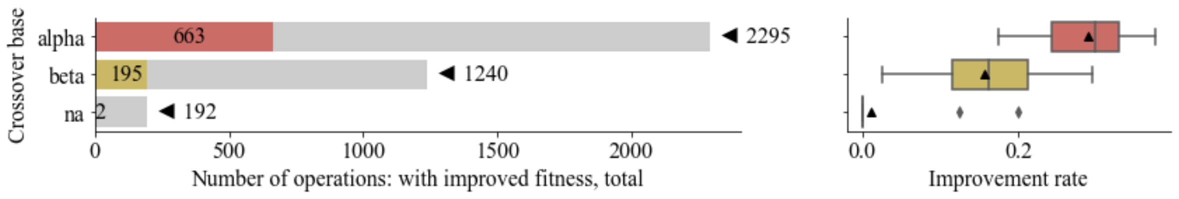


Figure 4.79 Amount and performance of crossover operations by crossover base. Concrete.

Branch crossovers take a bigger share (25.5%) of alpha-based operations compared to beta-based (22.42%). Branch and leaf crossover performances are not significantly different in either alpha- or beta-based operations (Figure 4.80). Crossovers that change only hyperparameters are significantly more effective when the root classifiers are inherited from the fittest parent: 29.62% vs 13.89% mean improvement rates. Both observations correlate with the idea that the higher the share of the fittest parent is observed in the offspring, the better the latter's performance.

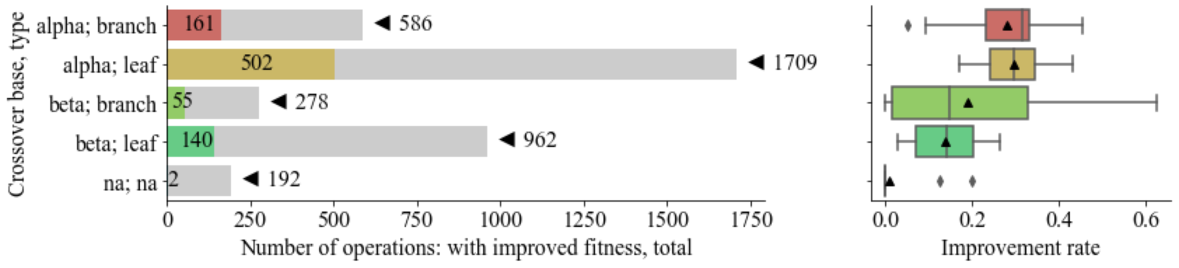


Figure 4.80 Amount and performance of crossover operations by crossover base and type. Concrete.

In the case of crossover, a level of operation means the edge on which the fittest parent was split to produce offspring. Crossovers with zero operational level mean that, presumably full parent tree was copied to the offspring. As expected, most operations happened in level 1 (Figure 4.81), where base classifier hyperparameters reside. Performance in second-level crossovers averages 28.77% and is significantly higher than the performance of the first-level crossovers, which means the improvement rate is 22.77%. Other levels have too few observations to make a confident conclusion.

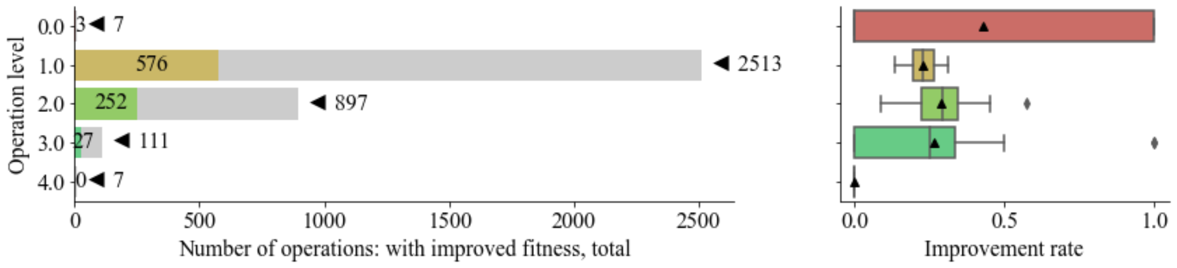


Figure 4.81 Amount and performance of crossover operations by the level of operation. Concrete.

4.4. Results and discussion

The first experiment aims to analyse the performance and construction of resulting pipelines which solve two classification problems (Breast cancer and Waveform) and two regression problems (Bioavailability and Concrete). Each test case is solved using four sets of TPOT settings or scenarios. Under scenario 1, the population of 250 individuals is initialized; under scenarios 2 and 4, the population of 50 individuals is evolved for five generations; under scenario 3, the population of 25 individuals is evolved for ten generations. Scenarios 2 and 4 differ in crossover and mutation rates, which are default (10:90) in scenario 2 and equal (50:50) in scenario 4. All test cases are split into training (90%) and test (10%) sets. Each experiment is run 30 times, so every test case has 120 independent solutions, which analysis is summarized below.

No significant difference exists between test scores of initializations and scores of evolutions in any of the four scenarios. Training scores of initializations, however, are significantly lower in the case of classification problems: Breast cancer and Waveform.

Generation 1 internal CV score (scenarios 2 and 3) results significantly differ from the results of the population initialized in scenario 1. Only in breast cancer and concrete cases is the difference between initialized (scenario 1) and evolved 50 individuals (scenario 2). In cases of Waveform and Bioavailability, the significant difference appears only between scenarios 1 and 3 (25 individuals evolved).

The relationship between internal CV scores of initialized and fully evolved scenarios varies with test cases. Breast cancer case has the most distinct results: all three scenarios have significantly different average CV score distributions. In the case of Concrete, initialized scores are significantly lower than 25 individuals evolved during ten generations (scenario 3) only.

The same is for the Waveform, but with a lower significance level: α=0.1. No significant difference is observed in the final results in the case of Bioavailability. Scenario 2 CV scores of Bioavailability reach the level of scenario 1 in generation 3 and barely grow since then. Scenario 3 scores of Bioavailability do not reach the level of scenario 1 during 10 generations and end up having the lowest average result.

Running TPOT on regression pipelines resulted in all unique pipelines (i.e., 120). However, in the case of classification problems, the output is sometimes repetitive. So, the Breast cancer problem has 118 unique solutions, and the Waveform problem has 106. However, when hyperparameters are disregarded, and only combinations of pipeline operators (e.g., classifiers, selectors) are taken into consideration, the picture is different. Over half of the solutions are operation-wise unique in the case of classification problems: 69 unique operation combinations the Breast cancer and 61 for the Waveform problem. Regression problems, however, have about a third of the solution operation-wise unique: 40 for the Bioavailability and 46 for the Concrete problem. The number of operation-wise unique pipelines is the lowest in scenario 1 and the highest in scenario 3. The number is similar in scenarios 2 and 4; both lay between the values for scenarios 1 and 3. Breast cancer unique solutions vary from 16 to 27, Waveform from 17 to 25, Bioavailability from 11 to 20 and Concrete from 11 to 17.

Regression problem solutions also tend to be shorter than classification problem solutions. Bioavailability and Concrete case solutions consist of just one operator in about half of the cases: 65 in the case of Bioavailability and 59 in the case of Concrete. Another third of the regression pipelines are of a size 2. Classification problem solutions, however, are of the size 2 in over half of the cases: 63 in the case of Breast cancer and 65 in the case of the Waveform. The number of one and three-operator solutions is almost equal for classification problems: 24 and 24 for Breast cancer and 26 and 22 for Waveform.

The bigger size of classification solutions also means a heavier use of preprocessors and selectors. The selector is typically the least frequently used pipeline operator and appears up to 10 out of 120 times in all but the Waveform solutions. Just half of the Waveform dataset attributes are meaningful, which justifies the untypically frequent use of selectors: 39 cases. Preprocessing is common in both classification solutions. While Waveform solutions include preprocessors in a third of the cases, most Breast cancer solutions have at least one preprocessor; in 12 cases, it is included more than once.

The second experiment analyses the statistics and performance of genetic operators (mutation and crossover) used in scenario 4. The experiment was performed on the same four cases: two classification and two regression problems. Mutation and classification are set to be performed with equal probability. The main operator performance indicator is whether the pipeline's fitness after a genetic operation is better than before. In the case of crossover, resulting pipeline fitness is compared with the fittest parent. The results are summarized below.

Initialized population always consist of pipelines of a size 1 or 2. Its share decreases to up to 81.6% by generation 5 (Concrete case). The share decrease is the smallest in the case of Bioavailability. Unlike other test cases, where both size 1 and size 2 shares decrease over generations giving ground to larger pipelines, the number of one-operator solutions for Bioavailability is growing and reaching 57.8% by generation 5 (before operations). While classification solutions grow in size up to 6 in the case of Breast cancer and 7 in the case of the Waveform, selected pipelines for regression problems do not exceed the size of 4.

Mutation and crossover are performed with equal probability, but the mutation pipeline improvement rate is significantly higher in all test cases. Crossover results are always compared with the parent with the highest fitness value, likely contributing to lower improvement rates. In the Breast cancer case, mutation and crossover improvement rates are the lowest compared to other cases: 29.0% and 17.4%. In the case of Concrete, they are the highest: 36.3% and 23.0%. The waveform has rates higher than Breast cancer: 31.2% and 18.2%. Bioavailability has them lower than Concrete: 30.9% and 20.8%.

In all cases, improvement rates of mutation are decreasing over generations. The closer the pipeline result is to the perfect score, the lower are chances of improving the score via mutation. The most common types of mutation are adding a new pipeline element (e.g., classifier or preprocessor) and changing an existing pipeline element (e.g., changing one classifier to another or preprocessor to selector) or hyperparameter value. Mutations that remove pipeline elements account for less than 10% of all mutations, which can be partly explained by the fact that almost half of all selected pipelines have only one pipeline operator, which cannot be removed. The difference between additive and changing mutations can be observed in three test cases: change mutations perform better in the case of Waveform and Bioavailability and worse in the case of Concrete. Mean improvement rates for additive mutations vary from 27.7% (Bioavailability) to 40.3% (Concrete); rates for changing mutations vary between 28.7% (Breast cancer) and 33.1% (Concrete). Preprocessor, classifier/regressor, selector, and combiner are added to the pipeline approximately in the following proportions: 45%, 40%, 10%, and 5%. Although preprocessors are added the most, their improvement rates are just about 21% in the case of classifications and about 26% in the case of regression problems, which are often significantly lower than for other added operators. Only in the case of Bioavailability performance of adding different operators has no significant difference. As expected, adding a selector is the most impactful (42.3%) additive mutation of the Waveform case. Breast cancer solutions benefited the most from adding another classifier (36.8%) and Concrete from adding a regressor (63.1%). Change mutations have 13 to 15 different variations depending on the case, but most of them have less than 20 observations. Hyperparameter switch accounts for most change mutations (about 73%) and a third of all mutations. Another frequently changed operator is classifier/regressor, which amounts to 21% of change mutations. In all cases, change in hyperparameters has a higher improvement rate, which averages vary between 30.8% (Breast cancer) and 35.0% (Concrete). Switch of classifiers benefits 22.2% and 26.1% of pipelines in Breast cancer and Waveform cases, respectively. Switch of regressors improves 25.3% and 27.9% cases of Bioavailability and Concrete. Like in the case of adding, preprocessor and classifier/regressor are the most typical removed operators. The variability of improvement rates of removing mutations is too high due to a small number of observations.

Unlike mutations, crossovers do not have a common pattern in performance change over generations. Crossovers also have a bigger impact on the pipeline size, which can be up to 3 units increase or decrease. Regardless of the test case, the most typical outcome is no change in size, largely due to the popularity of crossovers that change only hyperparameters. Crossover analysis is based on differences between the fittest (or alpha) parent pipeline and the first child pipeline. In most (approx. 62%) cases, a child pipeline inherits the root classifier/regressor from the observed (alpha) parent, in approximately 33% of cases, from the beta parent. In about 5% of cases, a parent cannot be identified (because either both pipelines are equal or have no common pipeline operators. Inheriting root classifier/regressor from the alpha parent benefits offspring more frequently than from beta. Alpha improvement rates vary from 21.7% (Waveform) to 28.9% (Concrete); beta improves the score in 13.0% (Breast cancer) to 16.8% (Bioavailability) of the cases. When the root is taken from one parent, the branch is taken from the other. When a branch consists of one unit, which can be only a hyperparameter, it is called a leaf. Inheriting leaves is the most common type of crossover and takes a share of about 72% in all test cases. When analysed separately, leave crossover does not perform significantly differently from branch crossover. Performances of both vary from about 18% (as in Breast cancer) to 25% (as in Concrete). Leaf crossovers have a significantly higher improvement rate than branch crossovers when the root is inherited from the alpha parent in both classification cases and Bioavailability (but with a lower significance level). Leaf crossovers perform worse than branch crossovers when the root is from the beta parent (with α=0.1) in Waveform and Bioavailability cases. It appears that the bigger share of the fittest parent is inherited, the better the fitness of the offspring.

For both mutations and crossovers, most of the changes happen on level 1 because that is where hyperparameters of the root classifiers and regressors reside, which are known as one of the most frequently changed parts of the pipeline. It is also where all the additions happen in the pipelines of depth 1 and where all removals and most of the additions and changes in pipelines with depth 2 occur.

5. CONCLUSIONS AND RECOMMENDATIONS

AutoML makes ML accessible to non-experts and reduces the time spent on iterative ML tasks like model selection and hyperparameter tuning. Thereby it helps to satisfy the increasing demand for ML specialists. During the last eight years, multiple AutoML tools have been developed, TPOT being one of the first. TPOT is an optimization tool that relies on genetic programming to help users identify the best pipeline. Unlike many other AutoML frameworks, which can find only the best models, TPOT discovers the best pipelines composed of feature selectors, preprocessors, and ML/DL models. TPOT is still under development and lacks a thorough analysis of its operation and effectiveness.

This thesis aims to democratize the knowledge about the TPOT and uncover what is happening under the hood. First, it covers the concept of genetic programming, the foundational concept of the TPOT framework. Then the processes behind TPOT are described in detail. A large part of this thesis is an overview of TPOT-optimized pipeline construction and a description of how pipelines change after crossovers and mutations.

The experiments were run 30 times each on four test cases: two classification datasets and two regression datasets. First, the performance of evolved pipelines was compared with the best of 250 initialized pipelines. Besides judging effectiveness based on test scores, internal scores were analysed by generation. Second, extracted data on each operation enabled studying the effect of mutation and crossover on the pipelines and their performance.

The framework source code reveals that, unlike many other genetic programming algorithms, TPOT selects individuals for mutation and crossover randomly and applies the selection algorithm to create a new population only after a new set of offspring is created and combined with the old population. TPOT uses the multi-objective selection algorithm NSGA-II, a one-point crossover and a mutation approach that uniformly selects between adding, removing and replacing pipeline operators. Moreover, TPOT exists in multiple pre-defined configurations, which can extend the search space with the operators not available in the default mode: neural networks algorithms, GPU-based models, multifactor dimensionality reduction and relief-based feature selection algorithms.

One of the key findings of the research is that evolved pipelines do not perform significantly differently from initialized. Nevertheless, pipelines evolved for 10 generations, somewhat outperformed other scenarios in two of four cases and evolved for 5 generations in one test case. More observations are obtained from the second experiment. Mutations lead to pipeline improvement more often than crossovers. However, comparing crossover offspring with the first parent or the parents’ average performance could lead to a higher improvement rate. Mutation improvement rates decrease over generations, which might imply that mutation effectiveness correlates with initial pipeline performance. The effectiveness of different mutation types is test case sensitive, which might speak in favour of TPOT’s ability to adapt to the given problem. About a third of all mutations is a hyperparameter swap which, by the way, improves pipelines in about a third of the cases. All initialized pipelines consist of a maximum of two operators. So, most of the mutations and crossovers happen on level 1 because after evolving for five generations, about 40% of the pipelines are still of size one. Inheriting root classifier/regressor from the fittest parent benefits offspring more frequently. Inheriting a subtree from the fittest parent is more beneficial than inheriting a terminal value. The opposite is true in the case of heritage from the beta parent. Interestingly, TPOT’s flexibility in pipeline creation can result in nonorthodox combinations of several, sometimes identic, classifiers or regressors within one pipeline.

Current research has been performed using the default configuration, but the results with other predefined and custom configurations are likely to differ. As TPOT operator effectiveness seems to vary with test cases, it is recommended to repeat the experiments with new data. A similar study could be performed on more complex problems using more individuals. Longer pipeline evolution could result in a wider range of pipeline sizes, making it more interesting to compare with relatively shorter initialized pipelines. When repeating the analysis, it is advised to extract data for both parents. That would facilitate the analysis of crossovers.

TPOT developers could consider adjustable mutation and crossover ratios based on their previous performance.

6. BIBLIOGRAPHY

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

Balaji, A., & Allen, A. (2008). Benchmarking Automatic Machine Learning Frameworks. arXiv:1808.06492v1

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1988). Waveform Database Generator (Version 2) Data Set. *UCI Machine Learning Repository*. Retrieved from https://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+(Version+2)

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2002). A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182-197, https://doi.org/10.1109/4235.996017

Fortin, F.-A., De Rainville, F.-M., Gardner M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 2171-2175.

Hutter, F., Kotthoff, L., & Vanschoren, J. (Eds.) (2019). *Automatic machine learning: methods, systems, challenges.* (Challenges in Machine Learning). Springer. https://doi.org/10.1007/978-3-030-05318-5

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

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and Computing, 4*, 87-112. https://doi.org/10.1007/BF00175355

Koza, J. R. (2010). Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines, 11*(3–4), 251-284.

Le, T. T., Fu, W., & Moore, J. H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics, 36*(1), 250–256. https://doi.org/10.1093/bioinformatics/btz470

Mangasarian, O. L., Street, W. N., & Wolberg, W. H. (1995). Breast cancer diagnosis and prognosis via linear programming. *Operations Research, 43*(4), 570-577. https://doi.org/10.1287/opre.43.4.570

Nolet, C. J., Lafargue, V., Raff, E., Nanditale, T., Oates, T., Zedlewski, J., & Patterson, J. (2020). Bringing UMAP Closer to the Speed of Light with GPU Acceleration. https://doi.org/10.48550/arXiv.2008.00325arXiv

Olson R. S., Urbanowicz, R. J., Andrews, P. C., Lavender, N. A., Kidd, L. C., & Moore, J. H. (2016a). Automating biomedical data science through tree-based pipeline optimization. *Applications of Evolutionary Computation, 9597*, 123–137. https://doi.org/10.48550/arXiv.1601.07925

Olson, R. S., Bartley, N., Urbanowicz, R. J., & Moore, J. H. (2016b). Evaluation of a tree-based pipeline optimization tool for automating data science. *In Proceedings of the genetic and evolutionary computation conference 2016*, 485-492. https://doi.org/10.48550/arXiv.1603.06212

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. (2011) Scikit-learn: machine Learning in Python. *Journal of Machine Learning Research 12*, 2825-2830. https://doi.org/10.48550/arXiv.1201.0490

Sohn, A., Olson, R.S., Moore, J. H. (2017). Toward the automated analysis of complex diseases in genome-wide association studies using genetic programming. https://doi.org/10.48550/arXiv.1702.01780

Street, W. N., Wolberg, W. H., & Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis. *Proceedings of the SPIE, 1905*, 861-870. https://doi.org/10.1117/12.148698

Truong, A., Walters, A., Goodsitt, J., Hines, K., Bruss, C. B., Farivar, R. (2019). Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools. *IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI) 2019*, 1471-1479 https://doi.org/10.1109/ICTAI.2019.00209

Wolberg, W. H.. Street, W. N., & Mangasarian, O. L. (1995). Breast Cancer Wisconsin (Diagnostic) Data Set. *UCI Machine Learning Repository*. Retrieved from https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)

Yeh, I.-C. (1998). Modeling of the strength of high-performance concrete using artificial neural networks. *Cement and Concrete Research, 28*(12), 1797–1808 https://doi.org/10.1016/S0008-8846(98)00165-3

Yoshida, F., & Topliss, J. G. (2000). QSAR Model for Drug Human Oral Bioavailability. *Journal of Medicinal Chemistry 2000 43*(13), 2575-2585. https://doi.org/10.1021/jm0000564

7. APPENDICES

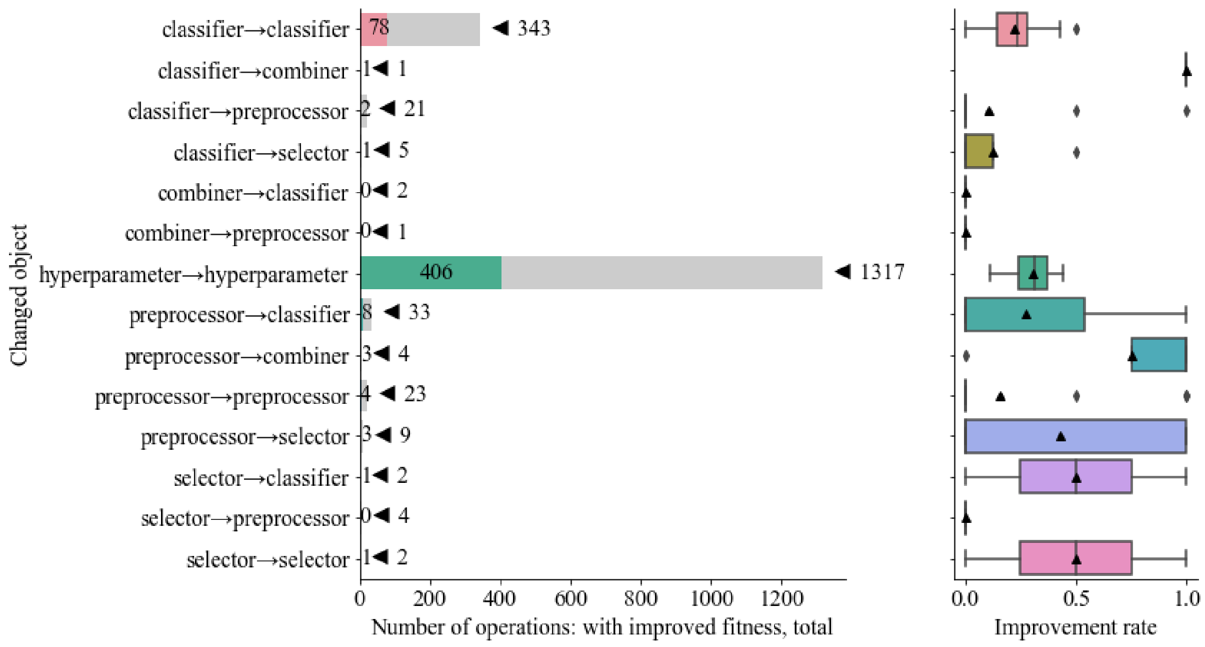
Appendix 1

Example of TPOT generated python code for a classification problem

```python
import numpy as np
import pandas as pd
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MaxAbsScaler
from tpot.export_utils import set_param_recursive

# NOTE: Make sure that the outcome column is labelled 'target' in the data file
tpot_data = pd.read_csv('PATH/TO/DATA/FILE', sep='COLUMN_SEPARATOR',
dtype=np.float64)
features = tpot_data.drop('target', axis=1)
training_features, testing_features, training_target, testing_target = \
            train_test_split(features, tpot_data['target'], random_state=42)

# Average CV score on the training set was: 0.976497948016416
exported_pipeline = make_pipeline(
    SelectPercentile(score_func=f_classif, percentile=78),
    MaxAbsScaler(),
    MLPClassifier(alpha=0.0001, learning_rate_init=0.01)
)
# Fix random state for all the steps in the exported pipeline
set_param_recursive(exported_pipeline.steps, 'random_state', 42)

exported_pipeline.fit(training_features, training_target)
results = exported_pipeline.predict(testing_features)
```

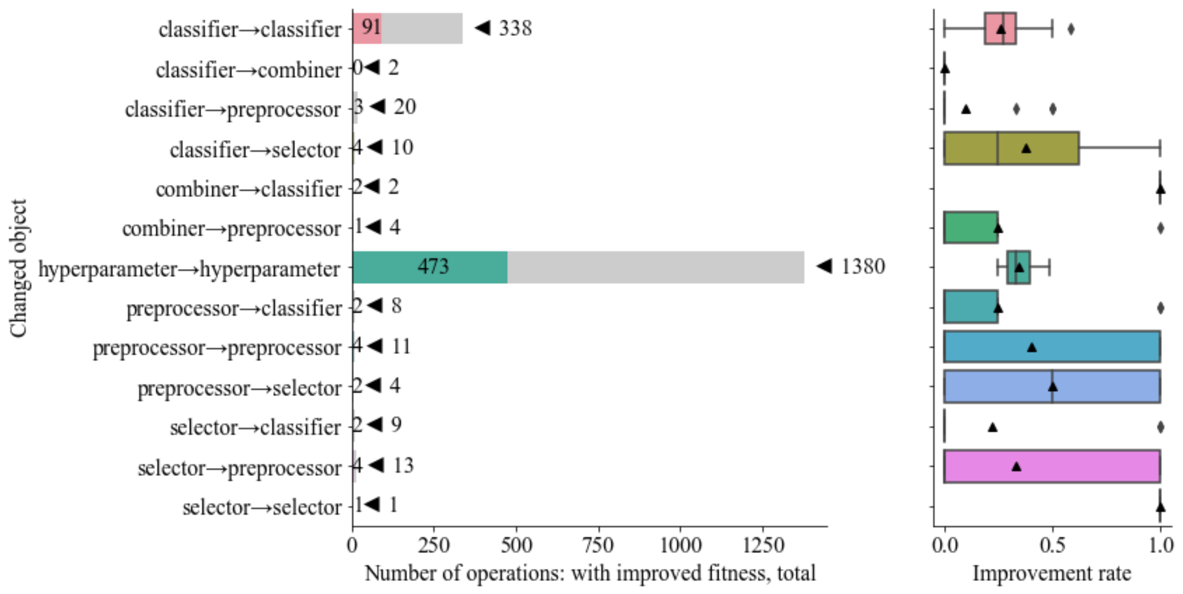
Appendix 2

Amount and performance of mutation operations by the changed object (full version). Breast cancer.



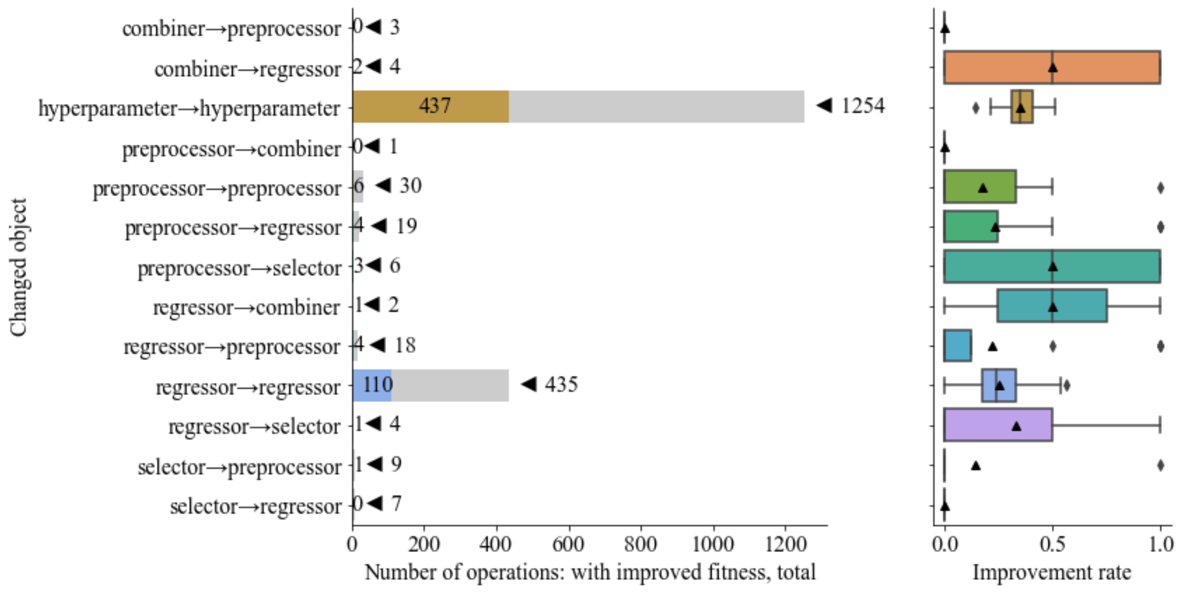
Appendix 3

Amount and performance of mutation operations by the changed object (full version). Waveform.



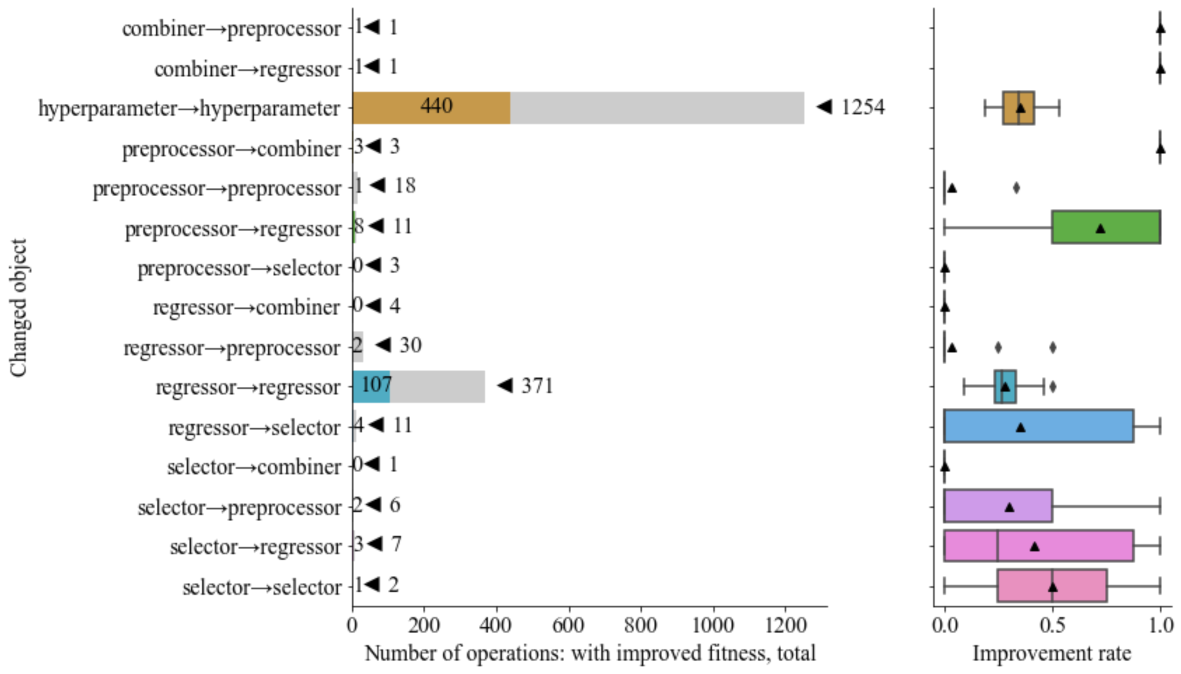
Appendix 4

Amount and performance of mutation operations by the changed object (full version). Bioavailability.



Appendix 5

Amount and performance of mutation operations by the changed object (full version). Concrete.


Changed object
combiner→preprocessor
combiner→regressor
hyperparameter→hyperparameter
preprocessor→combiner
preprocessor→preprocessor
preprocessor→regressor
preprocessor→selector
regressor→combiner
regressor→preprocessor
regressor→regressor
regressor→selector
selector→combiner
selector→preprocessor
selector→regressor
selector→selector
1 ◀ 1
1 ◀ 1
440 ◀ 1254
3 ◀ 3
1 ◀ 18
8 ◀ 11
0 ◀ 3
0 ◀ 4
2 ◀ 30
107 ◀ 371
4 ◀ 11
0 ◀ 1
2 ◀ 6
3 ◀ 7
1 ◀ 2
Number of operations: with improved fitness, total
Improvement rate