NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

ADVANCED GENETIC PROGRAMMING TECHNIQUES FOR MACHINE LEARNING

A comparative analysis with state-of-the-art automated machine learning methods in the context of imbalanced binary classification

Franz Michael Frank

Dissertation

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

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

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

ADVANCED GENETIC PROGRAMMING TECHNIQUES FOR MACHINE LEARNING

A comparative analysis with state-of-the-art automated machine learning methods in the context of imbalanced binary classification

by

Franz Michael Frank

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

Supervisor: Prof. Fernando José Ferreira Lucas Bação

10 2022

STATEMENT OF INTEGRITY

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

Lisbon, 13th of October 2022

ABSTRACT

The number of available machine learning methods and tools is increasing rapidly, with one recent trend being the usage of advanced genetic programming concepts and automated machine learning tools. However, through the rising number of upcoming innovations, it has become a challenge for machine learning applicants to keep up with all the new opportunities and to identify their potentials. While emerging methods are typically compared to conventional standard machine learning algorithms upon their initial introduction, research is still scarce on comparisons of the performances between the new concepts themselves. Therefore, this thesis provides a comparative analysis of two novel genetic programming techniques, differentiable Cartesian genetic programming for artificial neural networks and geometric semantic genetic programming, alongside three state-of-the-art automated machine learning tools, Auto-Keras, Auto-PyTorch and Auto-sklearn, with regard to their relative performances in the machine learning subfield of imbalanced binary classification. In this analysis, the five methods are tested against each other on 20 benchmark datasets, primarily regarding their average and maximum performance, and subsequently the most successful technique is applied to the real-world problem of fraud detection. The purpose of this thesis is not only to familiarize machine learning users with these methods, but above all to determine whether the novel genetic programming techniques can compete with the more established automated machine learning tools, and to identify the overall best performing method.

KEYWORDS

Genetic Programming; Automated Machine Learning; Imbalanced Binary Classification

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ANN	artificial neural network
AUC	area under the receiver operating characteristics curve
AutoML	automated machine learning
CGP	Cartesian genetic programming
CNN	convolutional neural networks
DCGP	differentiable Cartesian genetic programming
DCGPANN	differentiable Cartesian genetic programming of artificial neural networks
DL	deep learning
DNN	deep neural networks
FN	false negatives
FP	false positives
GA	genetic algorithm
G-mean	geometric mean
GP	genetic programming
GSGP	geometric semantic genetic programming
ML	machine learning
NLP	natural language processing
ReLU	rectified linear unit
RNN	recurrent neural network
ROC	receiver operating characteristics
TN	true negatives
TP	true positives

1. INTRODUCTION

A hot topic within machine learning (ML) that has recently been spreading rapidly in both research as well as industry is automated machine learning (AutoML). The partially or fully automated application of ML not only supports the work of domain experts, but by allowing the automation of entire pipelines, it also addresses the problem of the lack of such experts, as it makes ML accessible to non-experts as well (He et al., 2021). The range of tasks to which AutoML can be applied is very broad, ranging from processing simple baseline ML models up to complex artificial neural networks (ANNs) in deep learning (DL), with the number of upcoming AutoML methods and concepts continuing to rise (Wever et al., 2021). Among the latest trends to be observed is the application of genetic programming (GP) as an optimization method in AutoML.

However, while all these novel ideas are a major benefit for the field in general as many of the new opportunities allow to outperform the previous standards, the fast growth has made it challenging to track new techniques and identify their potential. In particular, research is rather scarce in terms of direct comparisons between the different new emerging approaches, which is therefore exactly what this work aims to address. This thesis primarily investigates two novel, advanced techniques from the field of genetic programming (GP), namely differentiable Cartesian genetic programming of artificial neural networks, abbreviated DCGPANN (Märtens & Izzo, 2019), and geometric semantic genetic programming, abbreviated GSGP (Bakurov et al., 2019). Moreover, these two techniques are evaluated alongside three more established tools from the automated machine learning (AutoML) domain, which are Auto-Keras (Jin et al., 2019), Auto-PyTorch (Zimmer et al., 2021) as well as Auto-sklearn (Feurer et al., 2015).

The subfield of supervised learning on which this work focuses is imbalanced binary classification, i.e., learning from a dataset with two classes where the number of instances of one class is considerably higher than the number of instances of the other. This tends to be a frequent and challenging problem when applying ML, and thus GP and AutoML approaches should be capable of overcoming it properly. Conventional ML approaches, in which no action is taken regarding the imbalance of the data, typically return a biased model when dealing with an imbalanced binary classification problem, since the model is trained using far more instances of the majority class, which results in it favoring that class in predictions (Douzas & Bacao, 2019). Therefore, the research community is continuously striving to discover and develop improved solutions to this problem (Douzas & Bacao, 2018; Douzas & Bacao, 2019; He & Garcia, 2009).

Consequently, the central research question of this thesis is whether the two novel GP techniques, DCGPANN and GSGP, are able to compete with the three state-of-the-art AutoML tools, Auto-Keras, Auto-PyTorch, and Auto-sklearn, when applied to imbalanced binary classification, and which of the five methods achieves the best overall performance on this task. To answer this question, primarily the average as well as the maximum achieved scores in terms of three common metrics for imbalanced binary classification, F1-score, geometric mean (G-mean) as well as area under the receiver operating characteristics (ROC) curve (AUC), of the methods across 20 benchmark datasets are evaluated. In addition, the individual techniques are investigated regarding the consistency of their performance as well as their generalization ability. The most promising method is finally applied to a real-world task of fraud detection among credit card transactions. Since the core mission of AutoML is to facilitate the use of ML (Karmaker et al., 2022), the tests are performed under simple setups without requiring

additional knowledge of specific topics such as hyperparameter optimization or sampling of imbalanced data, so that the results depend mainly on the capabilities of the five techniques themselves rather than on external factors.

At the beginning of this thesis, the theoretical foundations are provided in chapter 2. First, AutoML will be introduced, alongside with the three state-of-the-art tools. Next, GP is explained in general before presenting the two techniques GSGP and DCGPANN as special forms of GP. Finally, chapter 2 finishes with the introduction of ANNs, since these are the basis of one of the GP techniques, DCGPANN, as well as one of the AutoML tools, Auto-Keras, and thus represent an important core concept. Subsequently, chapter 3 outlines the research methodology, including a presentation of the benchmark datasets used for the study as well as the dataset concerning credit card transactions, and the setups of the different approaches. In chapter 4, the results of the application of the five techniques to the benchmark datasets are analyzed, followed by the results of the application of the most promising technique to the fraud detection problem. Afterwards, all findings and their meanings are debated in a discussion. Finally, the thesis is wrapped up by the conclusion provided in chapter 5 and a future outlook given in chapter 6.

2. LITERATURE REVIEW

This part introduces the theoretical foundations of the thesis by referring to existing literature. First, a general overview of AutoML is given, and the three state-of-the-art tools under consideration are explained in detail. Then, an introduction into GP is provided and the two special forms investigated in this work, DCGPANN and GSGP, are presented. Finally, ANNs are addressed, since they are relevant for this thesis, for example as an inspiration for DCGPANN or as a basis of the AutoML tool Auto-Keras.

2.1. AUTOMATED MACHINE LEARNING

The term AutoML stands for the automated application of ML. Lately, AutoML has gained a lot of popularity because it allows to automate certain parts of ML, and thus eliminates the need for a human expert for those specific operations. However, human involvement is still required to a certain extent in order to successfully solve real-world tasks using AutoML (Karmaker et al., 2022). As a result, AutoML can be considered to be a useful tool for data scientists, rather than some sort of competitor, while it can also provide access to ML for non-experts.

Put more technically, the majority of AutoML approaches aim to optimize the entire ML pipeline, with the primary focus usually lying on the process of building the model (He et al., 2021; Wever et al., 2021). The various AutoML tools can often be distinguished by two main characteristics, first, the search space used by a particular tool, for example for finding a model and its corresponding best performing hyperparameters, and second, the optimization method used to navigate through this search space (Wever et al., 2021). The search space is closely related to the task of a specific tool, e.g. the search space of an AutoML tool designed for the optimization of ANNs typically consists of all possible architectures of eventual ANNs, usually within certain defined boundaries (Jin et al., 2019). Therefore, there are huge differences in the respective search spaces, depending on the specific purpose of a particular tool. Likewise, there are various different optimization methods used by existing AutoML tools. Essentially any type of optimization can be employed, with the most common being grid and random search, gradient descent, reinforcement learning, surrogate model-based optimization, or evolutionary algorithms, such as genetic algorithms (GAs) or GP (He et al., 2021).

The application areas of AutoML tools in ML are very diverse with, among others, especially the usage of AutoML in DL having seen a rapid increase in interest recently (Zimmer et al., 2021). *Table 1* gives an overview of three of the most popular AutoML tools and their characteristics.

tool	brief description	underlying techniques
AutoKeras (Jin et al., 2019)	ANN optimization tool based on Keras	Bayesian optimization
Auto-Pytorch (Zimmer et al., 2021)	model selection and hyperparameter optimization tool based on Pytorch; also capable of optimizing ANNs	Bayesian optimization, meta-learning and ensemble construction
Auto-sklearn (Feurer et al., 2015)	tool with various features, such as model selection and hyperparameter optimization	Bayesian optimization, meta-learning and ensemble construction

Table 1 – AutoML tools and their characteristics

2.1.1. Auto-Keras

Auto-Keras (Jin et al., 2019) focuses entirely on ANNs. It is using the open-source ML library Keras (Chollet et al., 2015) to build the neural networks. The main goal of Auto-Keras is to make the task of creating and optimizing ANNs accessible to everyone, especially including those without sophisticated ML expertise (Jin et al., 2019). For this purpose, it allows to generate optimized ANNs for image data, text data as well as conventional tabular structured data, in each case for classification as well as for regression tasks. Additionally, it can also be used for time series forecasting. However, Auto-Keras does not perform any kind of data preprocessing steps. For the optimization process, Auto-Keras "utilizes Bayesian optimization to guide through the search space by selecting the most promising operations each time" (Jin et al., 2018, p.1). The following definition provides a formal explanation of Bayesian optimization.

Bayesian optimization (Theckel Joy et al., 2019, p. 658)

"Bayesian optimization is an efficient method for the global optimization of unknown objective functions $f : X \rightarrow \mathbb{R}$*, where* $X \subset \mathbb{R}^d$*. Formally, it solves*

$$x^* = \arg\max_{x \in X} f(x),$$

where X is a compact and convex set."

In simple terms, Auto-Keras is a technique that optimizes all hyperparameters of an ANN by means of a Bayesian technique. This involves repeatedly creating and training new ANNs until a predefined limit is reached, whereafter the best network is returned and available to be used for predictions.

2.1.2. Auto-PyTorch

Auto-PyTorch (Zimmer et al., 2021) is an AutoML toolkit based on the open-source ML framework PyTorch (Paszke et al., 2019). The primary objective of Auto-PyTorch is to automate both model selection and hyperparameter tuning for its users (Zimmer et al., 2021). It is mainly designed to handle tabular data and supports both classifaction and regression tasks. For the purpose of providing a well-fitting model, Auto-PyTorch first initializes various baseline ML algorithms, such as support vector machines or light gradient boosting machines, in certain hyperparameter configurations. The initialization process includes a data-driven meta-learning effort to already start with appropriate setups. Subsequently, Auto-PyTorch optimizes the hyperparameters of these models until a predefined time limit is reached using a Bayesian optimization-based tool called sequential model algorithm configuration, abbreviated SMAC (Lindauer et al., 2022). Data preprocessing techniques such as for example holdout or cross validation are also used during this process. Finally, from the resulting models, the best ensemble is built and provided as the final model. Additionally, the latest version of Auto-PyTorch is mainly dedicated to support DL in order to, for example, be able to optimize CNNs for image classification tasks (Zimmer et al., 2021). In summary, Auto-PyTorch primarily exploits the benefits of meta-learning, Bayesian optimization, and ensemble construction.

2.1.3. Auto-sklearn

Auto-sklearn (Feurer et al., 2015; Feurer et al., 2020), which is built on top of the open-source ML library scikit-learn (Pedregosa et al., 2011), was developed by the same research group as Auto-

PyTorch. Therefore, there are many similarities between these two AutoML tools. For instance, the initialization using meta-learning techniques and the subsequent optimization using a Bayesian optimization approach are almost identical. Thereafter, also with Auto-sklearn, the best ensemble of the resulting models is selected. As with Auto-PyTorch, a variety of ML algorithms are initially considered. In this case, these are the following 15: AdaBoost, Bernoulli naïve Bayes, decision tree, extremely randomized trees, Gaussian naïve Bayes, gradient boosting, k-nearest neighbors, linear discriminant analysis, linear support vector machine, kernel support vector machine, multinominal naïve Bayes, passive aggressive, quadratic discriminant analysis, random forest and stochastic gradient descent (Feurer et al., 2015). Also here, data preprocessing techniques are automatically used. The main difference, however, is that unlike Auto-PyTorch, Auto-sklearn does not operate with ANNs of any kind and, therefore, cannot be applied to DL tasks.

2.2. GENETIC PROGRAMMING

GP belongs to the family of evolutionary methods. Furthermore, GP is closely related to GAs and shares several important characteristics with them. For this reason, in the following, GAs are introduced first in order to subsequently refer to them when elaborating on GP in general. Finally, the two advanced types of GP, namely GSGP and differentiable Cartesian genetic programming (DCGP) in the form of its subtype DCGPANN, which are being tested in this work, are presented.

2.2.1. Genetic algorithms

Although GAs can be used for a wide range of problems, they are generally considered to be a technique for optimizing functions (Castelli et al., 2013). They take their inspiration from Charles Darwin's theory of evolution (Ezugwu et al., 2021; S. Wang et al., 2019). Thus, GAs can be described as “a class of computational models that mimic the process of natural evolution” (Vanneschi et al., 2017, p. 39). The four main features that unite most GAs are a whole population of chromosomes instead of just a single one, a selection process based on a certain fitness function, crossover to generate new chromosomes from existing ones, as well as the mutation of them (Castelli et al., 2013). The population of chromosomes that gets evolved typically represents a set of possible solutions within the search space of a given problem (Jiacheng & Lei, 2020).

Mitchell (1995), Castelli et al. (2013) and Jiacheng & Lei (2020) generalize the high-level procedure of a GA as follows: First, the initial population is created, typically randomly at this stage. Next, the genetic process starts with the selection of the further considered chromosomes, also called individuals, based on a fitness function that has to be defined in advance. Although this function also gives individuals with a weaker fitness score the chance to survive, individuals with a better score always have a better survival probability. The selected chromosomes then, with a certain probability, get crossed over, which results in new chromosomes. In this process of crossover, the loci structures within the initial chromosomes are typically divided and reassembled in a manner that produces admissible new individuals. Subsequently, the process of mutation follows. Thereby, the loci within the individual chromosomes are altered with a certain probability, which occurs independently of the other chromosomes. All individuals resulting from this procedure then replace the old population and the whole process of evolution starts all over again from this new population in the next iteration, also called generation. The parameters and operators that need to be specified in a simple GA are the fitness function, the respective genetic operators for selection, crossover and mutation, the size of the population, a criterion for stopping the formation of new generations, and the probabilities for

crossover and mutation respectively. A formal definition of the GA algorithm is given below as *Algorithm 1.*

I.	Initialize a random population of chromosomes.
II.	Calculate the fitness of all chromosomes within the population.
III.	Repeat steps i. – ii. until a termination criterion is reached.
i.	Repeat steps 1. – 4. until n offspring have been created.
	1. Select two parent chromosomes with replacement from the population.
	2. With a predefined crossover probability P_c, perform a crossover of the pair of selected chromosomes to form two new chromosomes. With probability 1 - P_c, continue with the two parent chromosomes as unmodified offspring.
	3. With probability P_m, mutate both offspring. With probability 1 - P_m, continue with the two unmodified offspring.
	4. Place the two offspring into the new population.
ii.	Replace the old population with the new population.
IV.	Return the best solution from the current population.

Algorithm 1 – GA (adapted from Castelli et al., 2013, p. 645)

2.2.2. Standard genetic programming

Genetic Programming was first introduced by Koza (1994). He describes GP as "a way to find a computer program of unspecified size and shape to solve, or approximately solve, a problem" (Koza, 1994, p. 88). Similar to GAs, GP is an evolutionary method and its basic idea is to evolve a set of solutions based on Darwin's theory of evolution (Tran et al., 2019). However, one of the most important differences of GP compared to GAs is the representation of the solutions. While with the application of GAs usually strings with a fixed length are evolved, in GP these solutions are dynamic computer programs, typically represented as LISP-like trees (Vanneschi et al., 2019). To enable the GP algorithm to develop these solutions, a set of functions and a set of terminals have to be defined in advance, on the basis of which the individual solution trees can be expressed (Koza, 1994). The set of functions can for example include, among others, mathematical functions, arithmetic operations or conditional logical operations, while examples for the terminal symbols would be a variety of numerical constants (Koza, 1994; Vanneschi et al., 2019). However, both the predefined set of functions as well as the set of terminals are highly dependent on the underlying problem. The solution space in GP consists of every possible composition of the possible functions and terminals (Koza, 1994).

Less than 20 years after the first introduction of GP, a large number of use cases for GP had already been found in which GP delivered results competitive with human-produced results. These use cases come from wildly varying domains including, among others, robotics, finite algebras, symbolic regression, analog electrical circuits, image recognition, bioinformatics or software repair (Koza, 2010). The successful application of GP in this broad range of scenarios is partly achieved because one of the advantages of GP is that only marginal knowledge about the underlying problem is required (Vanneschi et al., 2019).

Besides the strengths and versatile application possibilities, however, traditional GP also has certain limitations. According to Vanneschi et al. (2019), two of the main issues that come with GP are its heavy time consumption and the processes of mutation and crossover. The first problem of time consumption arises because each solution within the population must be evaluated which, depending on the problem at hand, often requires a highly extensive computation procedure. In this context, time consumption often correlates with certain parameters, such as tree size in particular, whereby an increasing value does often not lead to the corresponding desired better fitness (Vanneschi et al., 2019). The second problem addresses the fact that by the application of the standard genetic operators for GP only virtually blind syntax transformations take place in the respective solutions. This means that no knowledge about the resulting behavior of a solution after a transformation is taken into account (Moraglio et al., 2012; Vanneschi et al., 2019). Thus, Traditional GP entirely ignores the meanings of the computer programs being evolved, the so-called semantics (Moraglio et al., 2012).

2.2.3. Geometric semantic genetic programming

Addressing the aforementioned issue of the ignorance of semantics in traditional GP, Moraglio et al. (2012) introduced an advanced version of GP called geometric semantic genetic programming (GSGP). They refer to the fact that the success of a computer program, and thus of a solution in the population of a GP algorithm, is dependent on the semantics of the solution, which is why these semantics should not be disregarded. There are numerous ways to formally define the semantics of a solution. They can be defined as simply the fitness of the solution, the canonical representation, the mathematical function of the program or as a logical formalism which describes the programs behavior (Moraglio et al., 2012). Moreover, semantics in GSGP can in this context be defined as the vector of the output values of a possible solution within the population computed during the training phase (Vanneschi et al., 2014). According to this particular definition, a specific solution within the solution space associated to a GP algorithm is “a point in a multidimensional space called semantic space, where the dimensionality is equal to the number of observations in the training set” (Bakurov et al., 2021, p. 2). In accordance with this, *Figure 1* shows an example of a representation of solutions, on the left side in the genotypic space and on the right side with the corresponding points in a simple exemplary two-dimensional semantic space.

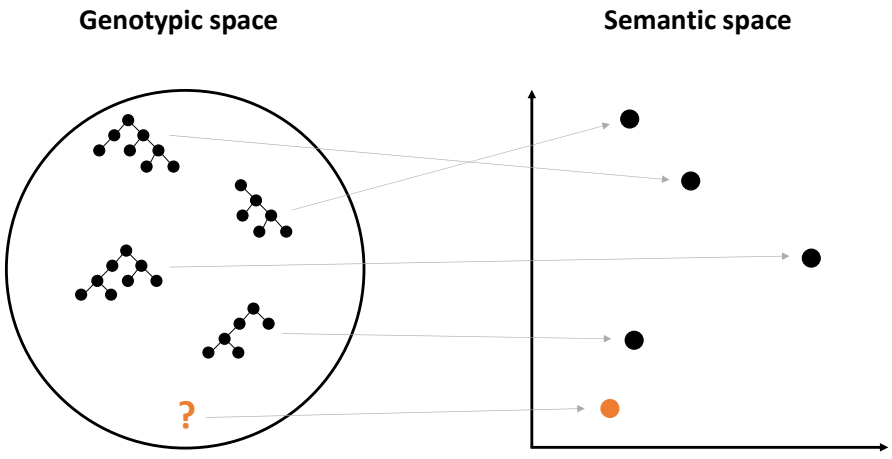


Figure 1 – Representation of solutions in the genotypic as well as the semantic space

The introduction of the semantic space enables the global optimum to be approached in a stepwise manner during the execution of a GP algorithm while, contrary to traditional GP, taking the meaning of the individual solutions into account (Moraglio et al., 2012; Vanneschi et al., 2013). For this purpose, however, new methods for crossover and mutation are necessary, the so-called geometric semantic operators (Moraglio et al., 2012), which are presented in the following.

2.2.3.1. Geometric semantic crossover

The aim of the introduction of the geometric semantic crossover is to perform a crossover of the genotypes of two solutions in such a way that it is possible to know the effect of the crossover on the offspring in the semantic space, and thus on the phenotype, in order to approach the global optimum (Krawiec & Pawlak, 2013; Moraglio et al., 2012). More precisely, the introduced geometric semantic crossover is performed in a way that any possible resulting offspring must be located between its parents in the semantic space, thus, the semantics of the offspring are a linear combination of the semantic vectors of its parents (Moraglio et al., 2012; Vanneschi et al., 2013). The consequence of this is that the fitness of the offspring must be at least as good as the worst fitness of its parents, which means that there is no possibility for the crossover to produce worse solutions in terms of fitness, which in turn implies that the movement in the semantic space can never be away from the global optimum (Moraglio et al., 2012). A formal definition of geometric semantic crossover can be given as follows.

Geometric semantic crossover (Moraglio et al., 2012; Vanneschi et al., 2013, p. 208)

"Given two parent functions T_1, T_2: $\mathbb{R}^n \rightarrow \mathbb{R}$, the geometric semantic crossover returns the real function $T_{XO} = (T_1 \cdot T_R) + [(1 - T_R) \cdot T_2]$, where T_R is a random real function whose output values range in the interval [0, 1]."

Moraglio et al. (2012) formally prove that a crossover based on the given definition always returns an offspring as a linear combination of its parents in the semantic space. In the actual process of creating the new offspring, the effect of the described crossover operator on a particular entry of the semantic vector will first be observed, and then used to describe the associated genotypic change required to produce that particular semantic modification. This description will then be used as guidance for the production of the offspring, whereby it must be noted that this means that the syntax of the offspring must also contain both of its parents, which means that the size of the solutions in this form of GSGP increases exponentially with each generation (Moraglio et al., 2012).

2.2.3.2. Geometric semantic mutation

In addition to geometric semantic crossover, a geometric semantic mutation method is also necessary for GSGP. Also in this respect, the effect of the genetic operator in the semantic space is of essential importance. In this context, the aim of geometric semantic mutation is to create a unimodal fitness landscape (Moraglio et al., 2012; Vanneschi et al., 2013). This is achieved by means of a box mutation, whereby the solution emerging after the mutation is only able to range within certain boundaries around the initial solution in the semantic space (Vanneschi et al., 2013). The following definition provides a formal explanation of geometric semantic mutation.

Geometric semantic mutation (Moraglio et al., 2012; Vanneschi et al., 2013, p. 208)

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

The fact that a mutation method based on this definition results in the desired box mutation within the semantic space is proven by Moraglio et al. (2012). Vanneschi et al. (2013) state that every element of the resulting semantic vector in this form of mutation represents a rather weak perturbation of the associated element of the semantic vector of the parent. While the magnitude of the mutation's effect can be changed by modifying the mutation step ms, the perturbation can be described as weak as the random expression it results from is centered around zero (Vanneschi et al., 2013).

2.2.3.3. Implementation of geometric semantic genetic programming

While GSGP in theory introduces groundbreaking advantages such as a steady convergence to the optimum with crossover as well as a unimodal fitness landscape with mutation, the exponential growth of programs over generations renders the operators in this form basically infeasible in practice (Vanneschi et al., 2013). The solution Moraglio et al. (2012) suggest for this problem is a simplification of the syntax of the offspring, whereby the actual program that the offspring represents must always remain unchanged. However, since this approach is only a rather small improvement in practice that doesn't resolve the issue, because even in this case the size of the individuals continues to grow and the design of this simplification in practice is another very challenging task, Vanneschi et al. (2013) introduced a new implementation, which is an efficient version of GSGP in practice without any simplifications needed.

The fundamental idea of this implementation is that it is possible to fully describe an individual by simply using its semantics in GSGP, which means that in comparison to standard GP, the syntactic element loses importance in GSGP (Vanneschi et al., 2013). Vanneschi et al. (2013) do not construct each individual in each generation anew, but merely keep an instruction on how to possibly construct the tree, as well as the semantics of it. One possible specification of the algorithm of their implementation can be obtained in detail below as *Algorithm 2*.

I.	Create a random population of n individuals and store them in a table P.
II.	Create a table M.
III.	Evaluate the individuals of the population stored in P and store the evaluations in table V. For this evaluation, every individual gets assessed for each possible fitness case, meaning that V consists of n rows and k columns, where k is the number of fitness cases of the training set.
IV.	Repeat steps i. – ii. until the maximum number of generations or another specified termination criterion is reached.
i.	Create a new table V'.
ii.	Repeat steps 1. – 4. until the new population of this generation is of the same size as the initial population.
	1. Perform crossover with a probability P_C.

a. Select two individuals from the population.

b. Apply geometric semantic crossover on the two individuals and store the resulting individual T represented as a triplet, $T = \langle ID(T_1), ID(T_2), ID(R)\rangle$, where R is a random tree and $ID(T_n)$ is a reference to T_n, together with the information that the corresponding genetic operator is crossover in M.

c. Obtain the semantics of the new individual T for all fitness cases as $(T_1 \cdot R) + ((1 - R) \cdot T_2)$ and store them in V'.

d. Store the random tree R in P and evaluate it over all fitness cases to get its semantics as a result and store them in V'.

2. Perform mutation with a probability $1 - P_C$.

   a. Select one individual from the population.

   b. Apply geometric semantic mutation on the individual and store the resulting individual T represented as a triplet, $T = \langle ID(T_1), ID(R_1), ID(R_2)\rangle$, where R_1 and R_2 are newly created random trees, together with the information that the corresponding genetic operator is mutation in M.

   c. Obtain the semantics of the new individual T for all fitness cases as $T_1 + ms \cdot (R_1 - R_2)$ and store them in V'.

   d. Store the random trees R_1 and R_2 in P and evaluate them over all fitness cases to get their semantics as a result and store them in V'.

3. Copy V' into V and erase V'.

4. Erase all the rows of P and M that do not refer to ancestors of individuals of the new population.

V. Reconstruct the best individual of the population to obtain its syntax.

Algorithm 2 – Efficient implementation of GSGP (adapted from Vanneschi et al., 2013, p. 209-211)

In *Algorithm 2*, two assumptions were made. Firstly, only crossover and mutation were considered, while in most cases a simple duplication of an individual from one generation to the next one is also possible. Secondly, in step V it is assumed that only the syntax of the best individual is needed, while there might as well be cases in which several of the individuals or even none at all, which would represent a black box case, are necessary. This implementation of GSGP solves the previously mentioned problems and allows for GSGP to be used in practice, as the tables P and M are still growing from generation to generation but in a linear manner and V stays the same size and can be updated in a time efficient way (Vanneschi et al., 2013; Vanneschi et al., 2014). Vanneschi et al. (2014) show that with this implementation, successful predictions in supervised learning tasks can be achieved.

However, this initial design of GSGP can only be applied to regression tasks. Therefore, Bakurov et al. (2019) have recently developed it further so that it can also be used for classification problems. The general concept is essentially the same, with the key modification being fairly straightforward: the classification task is treated as a regression problem, with the only possible target values being 0 and 1, which represent the two classes of the underlying problem (Bakurov et al., 2019). To accomplish this, a logistic activation function is used to transform the actual output of the GSGP solution into one of the two binary numbers and the root mean square error is utilized as the fitness function in this case (Bakurov et al., 2019; Bakurov et al., 2022).

2.2.4. Cartesian genetic programming

While in both standard GP as well as GSGP computer programs are evolved as tree-like structures, the programs in Cartesian genetic programming (CGP) are encoded as graph-based structures (Turner & Miller, 2015). Here, the name Cartesian GP comes from the fact that these structures are indexed typically by their respective Cartesian coordinates (Turner & Miller, 2017). In this context, a chromosome in CGP is represented as an "encoding of a graph as a string of integers that represent the functions and connections between graph nodes and program inputs and outputs" (Miller & Smith, 2006, p. 167). Turner & Miller (2015) thereby distinguish between three different types of chromosome genes: First, there are function genes, which define the functionality of a particular node within the graph. Then, there are genes that specify where the input for a particular node originates from, the so-called connection genes. Finally, the output genes indicate what is used as program output, which can be internal nodes as well as any input. The formal definition of the chromosome composed of these different types of genes is as follows.

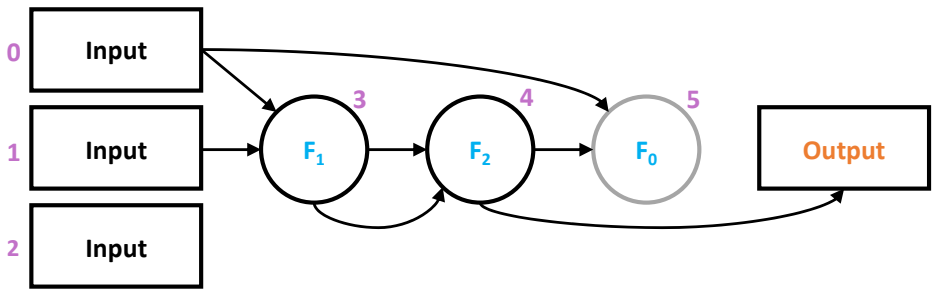
Cartesian genotype (Turner & Miller, 2015, p. 85)

"A generic (one row) CGP chromosome is… [defined as]

$$F_0\, C_{0,0} \ldots C_{0,\alpha}\, F_1\, C_{1,0} \ldots C_{1,\alpha} \ldots F_n\, C_{n,0} \ldots C_{n,\alpha}\, O_0 \ldots O_m$$

…where α is the arity of each node, n is the number of nodes and m is the number of program outputs."

For illustration, *Figure 2* provides an example of a single CGP program as both the graph and the corresponding numerical chromosome. It can be obtained that all of the three nodes of the DCGP individual are connected to either inputs or previous nodes, or both. Furthermore, it can be seen that one input is not used at all while another one is used multiple times, which can be the case in this form of GP. Additionally, in this example only two of the three nodes are actually contributing to the output, which is also possible to occur in DCGP.



corresponding chromosome: 101 233 004 4

Figure 2 – Example of a CGP program, represented as a graph as well as a chromosome (adapted from Turner & Miller, 2015, p. 85)

Märtens & Izzo (2019) proposed the concept of encoding and training ANNs using a differentiable form of CGP, which they refer to as DCGPANN. The term differentiable here implies that the information

about the derivatives of the program output with regard to the respective input as well as the weights is used for learning weights and biases while training the network. This is equivalent to the backpropagation procedure in ANNs. During the training phase of DCGPANN, ANNs are modified by means of a mutation of the activation functions and of the neural connections (Märtens & Izzo, 2019). The DCGPANN module developed by Izzo et al. (2016) can be applied to both classification as well as regression tasks.

2.2.5. Use cases of genetic programming

From the initial introduction of GP (Koza, 1994) to the present day, GP, along with its subtypes, has been successfully applied to a large number of widely varying task domains, of which some are listed hereafter. Among the very early application areas was the optimization of electrical circuits, wherein GP was capable of effectively automating parts of the circuit design process (Koza et al., 1999). In addition, another early use of GP was found in game playing, in which GP-based automated programs could already compete with human players in games such as for example chess or backgammon (Sipper et al., 2005). Furthermore, a remarkable use case was the successful GP-based automated design of an antenna that was deployed on a NASA spacecraft in 2006 (Lohn et al., 2008). Koza (2010) presents a detailed overview of the human-competitive results produced by GP from 1994 to 2009, to which is referred here for further examples.

Then, with ML's sharp rise in popularity and usage, especially concerning predictions in supervised learning, GP naturally has become of increased interest in this field as well. An example of GSGP's application to a regression problem is its implementation in the field of pharmacokinetics, where GSGP has shown to be capable of successfully predicting human oral bioavailability as well as protein-plasma binding levels of medical drugs (Vanneschi et al., 2013). Also, the successful use of GSGP for classification has been proven by, for example, producing reasonably good results compared to other ML models on several ML benchmark binary classification datasets (Bakurov et al., 2022). In addition, GP has also been used to optimize certain subtasks of ML, such as for example feature construction (Batista et al., 2021; Tran et al., 2019) or stacked generalization (Bakurov et al., 2021). Furthermore, a major application area of GP in recent years has been AutoML (Gijsbers & Vanschoren, 2019; Suchoparova & Neruda, 2020).

2.3. ARTIFICIAL NEURAL NETWORKS

ANNs, in the context of ML, derive their name from the fact that their design and functionality are inspired by the biological neural networks of the human brain (Bacanin et al., 2021). Typically, they are applied when the relationships of a problem are either very complex or completely unknown, whereby they define a non-linear connection between the given input data and the resulting output in order to provide a solution (Manngård et al., 2018; Yu et al., 2016). More specifically, ANNs conduct a transformation of the input data through a set of interconnected nodes, the so-called neurons, that results in the output data (Bacanin et al., 2021). Principally, provided that the amount of these neurons is large enough, an ANN is capable of estimating any desired continuous function (Manngård et al., 2018).

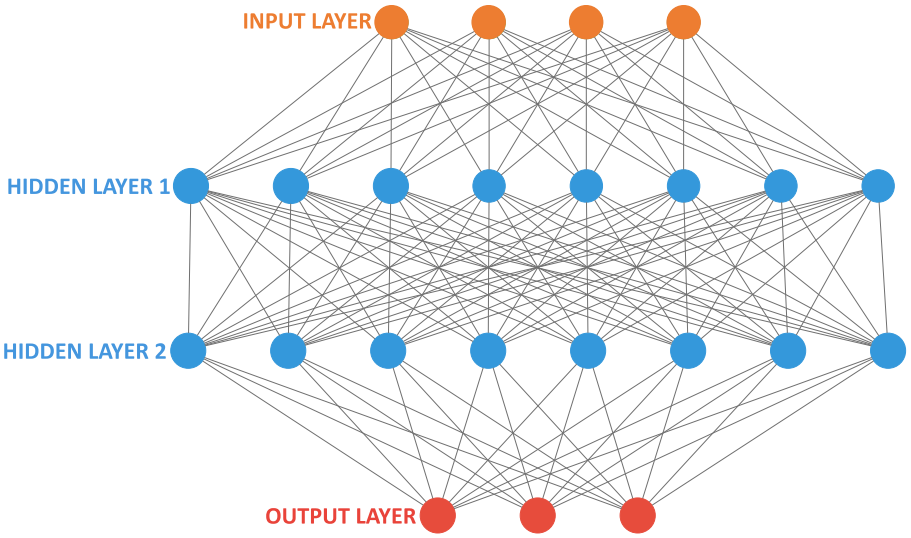


Figure 3 – Visualization of a simple ANN with two hidden layers

Generally phrased, an ANN is created by generating a certain number of neurons and connecting them in a specified way. Typically, the neurons are arranged in different layers, which can be divided into three different categories: input layers, hidden layers, also called processing layers, and output layers (Szymczyk, 2015). A graphical representation of a simple ANN with a four-dimensional input layer, two hidden layers and an output layer can be seen in *Figure 3*. In general, ANNs have exactly one input and one output layer each, while the number of hidden layers can be extended from a minimum of one up to an arbitrarily large number in theory (Berg & Nyström, 2018). In the scientific community the term deep neural networks (DNNs) is also commonly used when referring to ANNs with numerous layers, i.e. a large depth (Bacanin et al., 2021; Berg & Nyström, 2018; Manngård et al., 2018), but for the sake of clarity, in this thesis the term ANN is used consistently.

The individual connections between two neurons are assigned an initial weight (Manngård et al., 2018). Each neuron in both the hidden and output layers typically has a so-called activation function, which transforms the weighted input through mathematical calculation into the neuron's output (Berg & Nyström, 2018; Szymczyk, 2015). There are different types of activation functions, with the most common being rectified linear unit (ReLU), sigmoid, hyperbolic tangent, linear activation and softmax (Badem et al., 2017; Berg & Nyström, 2018). The latter two are commonly used as activation functions of output layer neurons, with linear activation generally used for regression problems (Berg & Nyström, 2018), and softmax for classification problems (Badem et al., 2017).

Once the basic structure of an ANN is built, the process of training can be initiated, during which, among others, the weights of the individual connections as well as the biases, which are constants that are added to the weighted inputs of a respective neuron, are adjusted, or in other words learned (Bacanin et al., 2021). The training phase involves passing the data through the ANN, whereby passing it one single time is called an epoch. In practice, ANNs are usually trained for hundreds or thousands of epochs until adequate levels of performance are achieved (Komaris et al., 2019; Pasa et al., 2015). Training ANNs can be considered a supervised learning task, since it has to be conducted on the basis of labeled data in order to either typically maximize accuracy in the case of a classification problem or to minimize the error in the case of a regression problem (Bacanin et al., 2021). When training ANNs, a distinction is made between two principal types, namely ANNs with a one-way connection and ANNs

with feedback (Szymczyk, 2015). Put into technical terms, the former are called feed-forward neural networks while the latter are called neural networks with backpropagation. Described in a simplified way, the feed-forward approach involves the adjusting calculation of the weights from the input layer to the output layer, while the backpropagation approach carries out the whole process backwards, i.e. from the output to the input layer (Jeon et al., 2021).

ANNs have many beneficial properties, including their ability to learn and their ability to self-organize as well as to adapt (Szymczyk, 2015). Furthermore, they tend to be particularly superior to conventional ML models in cases where the underlying problem cannot be explained by a mathematical model or formula (Yu et al., 2016). Consequently, ANNs and their subtypes are being increasingly used in practice, achieving good results in a wide range of areas such as self-driving cars, pattern recognition, natural language processing (NLP), image analysis, as well as object recognition, to name just a few (Berg & Nyström, 2018). However, a major problem with the application of ANNs in DL is overfitting (Bacanin et al., 2021; Badem et al., 2017; Manngård et al., 2018). ANNs can easily be tailored to accurately predict all training instances, and thus achieve virtually perfect training performance, while they might at the same time perform significantly worse when unseen data is provided to the input layer which would ultimately result in a poor test performance (Bacanin et al., 2021). The more complex an ANN is built, the higher the risk of overfitting (Berg & Nyström, 2018), thus, a simple representation, characterized for example by fewer layers or numerous weights being zero, is often desirable (Manngård et al., 2018). Besides the here presented standard ANN, the two most popular and most widely used advanced types are the convolutional neural network (CNN) and the recurrent neural network (RNN), both of which have already been successfully applied in science as well as in practice (Mao & Sejdic, 2022; Mecheter et al., 2022; Xie et al., 2022; Yin et al., 2022; Zhao et al., 2018).

The biggest challenge in the application of any type of ANNs is to find the optimal network architecture as well as the corresponding set of hyperparameters (Bacanin et al., 2021; Yoo, 2019; Zhang et al., 2021). In practice, this problem leads above all to great challenges in terms of computational time, since the search space, which grows exponentially with the number of parameters, typically is of enormous size (Li et al., 2022). Besides the large number of possible hyperparameter combinations, however, their evaluation is also challenging, since ANNs tend to be very complex, and therefore training a single one often requires a substantial amount of time as well (Zhang et al., 2021). Hence, providing approaches to address the problem is very helpful for the practical application of ANNs as it facilitates their usage.

Essentially, hyperparameter search in ANNs can be described as a global optimization problem with the objective of finding the combination of hyperparameters that minimizes the validation error, or respectively maximizes the accuracy, of the network (Zhang et al., 2021). Moreover, it is characteristically a black box optimization problem, since the underlying function to be optimized is not a conventional linear mathematical function, but in fact rather the opposite, as it includes highly non-linear operators (Yoo, 2019).

parameter	brief description	type
learning rate	controls speed of learning of the ANN	training

loss function	evaluates how well the ANN predicts the target variable	training
mini-batch size	quantity of instances passed through at the same time	training
number of epochs	number of times the data gets passed through	training
number of hidden layers	number of layers between input and output layer	structure
layer size	number of neurons within the layer	structure
weight decay	responsible for regularization	structure
activation functions	transforms weighted sum of inputs	structure
weight initialization	initial values of weights of the connections	structure
bias initialization	initial values of biases of the nodes	structure

Table 2 – Most important hyperparameters of standard ANNs (Yoo, 2019)

When optimizing the hyperparameters of ANNs, a distinction is made between two different subsets, one defining the structure of the network and the other determining the process of training the network (Yoo, 2019). An overview of the most important hyperparameters to be optimized in standard ANNs is presented in *Table 2*, with the right column indicating to which subset the respective parameter belongs to. However, this is not the complete set of all possible parameters, as, depending also on the type of the ANN, there may be additional ones to consider.

3. METHODOLOGY

The evaluation of the performance of the two chosen GP approaches and the three state-of-the-art AutoML modules is carried out in a comparative analysis of the five techniques on 20 different ML benchmark datasets for binary classification. Subsequently, the best performing tool is applied to the real-world use case of fraud detection, whereby the assessment is conducted based on a dataset containing credit card transactions that are either fraudulent or not. In order to put the results into perspective, the mentioned tasks are also solved using a simple logistic regression, a typical baseline model for classification. Similar to other studies in the ML field that compare the performances of different models or tools (Bakurov et al., 2019; Bakurov et al., 2022), each technique is examined on 30 independent executions per dataset in order to produce statistically representative results.

3.1. APPLICATION TO BENCHMARK DATASETS

As usual in these kinds of comparisons of various techniques, a variety of different datasets are needed in order to obtain meaningful results. Therefore, for this thesis, 20 different benchmark datasets are used, which is a number similar to other related papers (Bakurov et al., 2022; Douzas & Bacao, 2018). All chosen datasets and their characteristics are presented in *Table 3*. They differ mainly in the categories number of instances, number of features and imbalance ratio, which is calculated by dividing the size of the majority class by the size of the minority class.

name	number of features	number of instances	majority instances	minority instances	imbalance ratio
arcene	1998	200	112	88	1.273
audit	25	775	470	305	1.541
banknote authentication	4	1372	762	610	1.249
breast cancer	30	569	357	212	1.684
cleveland	13	297	243	54	4.500
dermatology	33	366	346	20	17.300
ecoli	7	336	284	52	5.462
eucalyptus	8	642	544	98	5.551
haberman	3	306	225	81	2.778
ionosphere	32	351	225	126	1.786
led	7	500	455	45	10.111
libras	90	360	336	24	14.000
liver	6	345	200	145	1.380
page blocks	10	5473	4913	560	8.773
parkinsons	22	195	147	48	3.063
pima	8	768	500	268	1.866
spambase	57	4601	2788	1813	1.538

vehicle	18	846	647	199	3.251
vowel	12	9961	8865	1096	8.089
yeast	8	1484	1240	244	5.082

Table 3 – Benchmark datasets and their characteristics

The datasets are downloaded using ML-Research (Fonseca et al., 2021), an open-source library for machine learning research, and they represent several of the best-known binary classification benchmark datasets of the ML field. All datasets remain unmodified in terms of their number of features as well as their number of instances. Furthermore, no missing values exists across all 20 datasets, thus, there is no need to apply any techniques for missing value replacement. However, one step that needs to be performed is the normalization of the variables, since they often vary widely in their respective magnitudes. The normalization method chosen for this purpose is Z-score normalization, which is defined, for example by Bakurov et al. (2019, p. 43), as follows.

$$z_{ij} = \frac{x_{ij} - \mu_i}{\sigma_i}$$

Here, x_{ij} represents the i-th feature of the j-th instance, μ_i and σ_i describe the mean and the standard deviation respectively, calculated for every feature f_i. Finally, the resulting z_{ij} is the desired normalized value, and therefore replaces the corresponding real value x_{ij} in the dataset. After the normalization is completed, the datasets are ready to be processed by all classifiers in exactly this same format.

After the respective 30 executions per dataset, the techniques are compared across four different categories, each time considering the metrics F1-score, G-mean and AUC. The first two categories are the average value and the maximum value of the 30 resulting individual scores, making them the most appropriate factors to assess the actual success of a technique. The third category is the standard deviation of the scores, which therefore is intended to provide information about the consistency of the performance. The last category is the generalization ability, which is supposed to reveal the performance gap between training and test data. After the techniques are ranked all together in each of these four categories, the statistical significance of these rankings is tested using the Friedman test (Guyon, 2003), in order to determine if there is a significant difference among the ranks. Following this overall comparison, the techniques are additionally examined pairwise in order to be able to draw conclusions about differences between each possible pair of tools. In the latter case, statistical significance is checked using the Wilcoxon rank-sum test (Haynes, 2013). Finally, in addition to the four categories described, it will be tested whether there is a relationship between the performance and the characteristics of the datasets. Both the Pearson and Spearman correlations between the performances in terms of the respective metrics and the individual dataset properties are examined.

3.2. APPLICATION TO FRAUD DETECTION

The real-world scenario chosen for this thesis addresses fraud detection, since this task is an ideal example of a binary classification problem, where one class is vastly outnumbered by another in terms of the number of instances. For identifying fraud among financial transfers, it is typically the case that, while information about a large number of non-fraudulent transactions is available, only a very small amount of fraudulent ones are accessible, as these occur far less frequently in practice (Pozzolo et al., 2015). The dataset used includes information on 284,807 real credit card transactions and was

originally provided to another research group (Pozzolo et al., 2015) by their industrial partner. It has 28 numeric features in addition to one target variable which indicates a non-fraudulent transaction with 0 and a fraudulent one with 1, further information can be found in *Table 4*.

name	number of features	number of instances	majority instances	minority instances	imbalance ratio
credit card transactions	28	284807	284315	492	577.876

Table 4 – Credit card transactions dataset and its characteristics

The technique with the best performance on the benchmark datasets is applied to the real-world problem. Again, logistic regression is used as a reference. In this case, the respective average and maximum scores after 30 executions are used for the assessment, since in practice these are the most important, because the model with the best score on the test data would be used for the predictions on unseen data.

3.3. EVALUATION METRICS

The basis of several evaluation metrics for binary classification is the so-called confusion matrix (He & Garcia, 2009), which is illustrated in *Table 5*. It divides the predictions given by a model into four different categories. True positives (TP) are the correctly identified instances of one class, e. g. class 1, false positives (FP) are instances that are incorrectly assigned to that class, and true negatives (TN) are the correctly identified instances of the other class, e. g. class 0, while false negatives (FN) are instances that are incorrectly assigned to this other class.

		true class	
		positive (1)	negative (0)
predicted class	positive (1)	TP	FP
	negative (0)	FN	TN

Table 5 – Confusion matrix

Typically, the most widely used metric for measuring a classifier's success in a binary classification problem is the accuracy measure defined below, which simply calculates the percentage of correctly identified instances in total, without examining the classes separately (He & Garcia, 2009). However, when dealing with imbalanced data, i.e. when one class is considerably larger than the other in terms of the number of respective instances, accuracy is generally not suitable to be used as the metric. For example, given a problem where 97% of the instances belong to class 0 and only 3% to class 1, an approach that simply assigns all cases to class 0 would achieve a very good accuracy of 97%, despite completely failing to identify any instances of class 1.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Since the majority of the benchmark datasets used are imbalanced, and especially because fraud detection is naturally characterized by a particularly high imbalance, using a more appropriate metric than accuracy is essential for this analysis. For this reason, in the course of this work, as in various other studies on imbalanced learning (Bakurov et al., 2022; Douzas & Bacao, 2018; Douzas & Bacao, 2019), the so-called F1-score, which is defined below, is used as a metric for assessing the success of the tools. The F1-score is the harmonic mean of precision and recall, with precision being the share of all correctly predicted positives among the entire number of predicted positives and recall representing the percentage of correctly identified positives out of the total actual positives. Thus, in contrast to accuracy, the F1-score is more informative when dealing with an imbalanced classification problem, as it would, for example, be 0, which is the lowest possible value, in the previously mentioned example where a classifier cannot identify any correct positive instances at all, no matter how many negatives are accurately predicted.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

In order to consolidate the results of the comparison instead of making them dependent on just one single metric, in addition to the F1-score, the individual techniques are also compared using two further measures. These are the G-mean, which is defined hereafter, as well as the AUC (He & Garcia, 2009). The G-mean is defined as the square root of the product of sensitivity and specificity. The sensitivity is equivalent to the recall described above, while the specificity describes the fraction of correctly predicted negatives out of the entire actual negatives. Therefore, the G-mean is a measure for evaluating "the degree of inductive bias in terms of a ratio of positive accuracy and negative accuracy" (He & Garcia, 2009, p. 1277).

$$Sensitivity = Recall = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$G\text{-}mean = \sqrt{sensitivity \cdot specificity}$$

For calculating the AUC metric, it is necessary to first obtain the ROC curve. The ROC graph is composed of both TP rate and FP rate axes, with the TP rate being identical to sensitivity or recall, and the FP rate representing the ratio between incorrectly predicted positives and the actual negative instances. Typically, the TP rate represents the y-axis of the ROC graph while the FP rate corresponds to the x-axis. It is common that the ROC curve of a classifier is composed of several points, which usually are characterized by different threshold values, i.e. the cutoff point of the predicted probability from which an instance is assigned to one class instead of the other. However, it is also possible that a classifier

produces only a single ROC point, so only a single pair of TP rate and FP rate, which is the case with models that only output hard class labels instead of the corresponding probabilities (He & Garcia, 2009). The latter is the case for this study, since it is not possible to directly obtain the probabilities with certain tools. Therefore, for reasons of comparability, the AUC score is calculated solely based on the predicted class labels throughout all techniques. While this makes a single classifier's ROC curve itself hardly informative, the comparison of the respective AUC value is nevertheless a useful measure to evaluate the tools' performances against each other.

$$TP\,rate = Sensitivity = Recall = \frac{TP}{TP + FN}$$

$$FP\,rate = \frac{FP}{TN + FP}$$

For the FP rate, 0 is the best value, since it implies that no FP exist among the predicted class labels, and 1 is the worst. Regarding all the other metrics, a value of 1 is the optimal score, while a classifier evaluated with a 0 is considered to be not useful at all. The resulting metric values are consistently rounded to three decimal places in the course of this work. In order to be able to assess the performance of the various techniques on unseen data, they are evaluated using the relevant metrics on the test partitions of the respective datasets, i.e. the train partitions are irrelevant in this context and are only used for training the models. The train and test partitions are generated for each dataset with a fully random division into 75% train and 25% test set, with this split being constantly redone for each of the 30 independent executions, so that the results do not depend on the specificities of a single partition. Each execution is strictly limited to a time of 300 seconds to ensure equivalent conditions.

3.4. SETUPS OF THE TECHNIQUES UNDER ANALYSIS

Since the objective of the AutoML tools is to simplify and automate the application of ML models, the most appropriate for this study is to use them mainly with their respective default setups. Furthermore, in order for the comparison with the established AutoML tools to be meaningful, the GP approaches are also not undergoing a separate hyperparameter optimization, but instead are applied in the same setup to all datasets. Likewise, the logistic regression is applied in its default setup, as given by the ML library scikit-learn (Pedregosa et al., 2011). The versions of the respective Python libraries used for the six different approaches can be found in *Appendix 1*.

3.4.1. Setups of the state-of-the-art automated machine learning tools

The implementation of Auto-Keras, Auto-PyTorch and Auto-sklearn is very straightforward. Obtaining a functioning and optimized model requires only two steps, one is initializing the classifier and the other one is its training, with both corresponding to a simple line of code each. A major benefit of these three tools is that they allow the user to choose the metric according to which the models should be optimized. In order to take advantage of this feature, the default optimization metric of all three of them is replaced with the F1-score, since the examined datasets are all imbalanced binary classification problems, and the techniques are subsequently evaluated mainly based on the achieved F1-score. Apart from this modification of the optimization measure, however, the default setups of Auto-Keras, Auto-PyTorch and Auto-sklearn remain unchanged for this study. This means for Auto-PyTorch that it does not use ANNs but rather makes use of its pool of baseline ML models.

3.4.2. Setups of the genetic programming approaches

In comparison to the established Auto-ML tools, the DCGPANN as well as the GSGP approaches are more laborious to implement, as several parameters need to be specified first, and it also requires more lines of code. However, since for the other tools simply the default setups are used, for reasons of fairness and comparability there is no individual hyperparameter tuning conducted for these approaches, instead the standard parameters provided by the developers are used.

The setup used for DCGPANN, which is the one suggested by Izzo et al. (2016), can be obtained from *Table 6*. In comparison to the AutoML tools, a drawback is that DCGPANN cannot be optimized according to the F1-score. Hence, DCGPANN is optimized according to its default measure for classification, the cross-entropy loss.

parameter	value
number of hidden layers	2
size of hidden layer 1	50
size of hidden layer 2	20
activation function of hidden layers	sigmoid
activation function of output layer	softmax
epochs	no limit
initial network weights	normally distributed numbers with $\mu = 0$ and $\sigma = 1$
initial network biases	normally distributed numbers with $\mu = 0$ and $\sigma = 1$
learning rate	1.0
number of levels-back in the Cartesian program	5

Table 6 – Parameter setup of DCGPANN

GSGP, like the state-of-the-art AutoML tools, is optimized using the F1-score in the course of this study. All other elements of the parameter setup are adapted from the introducers of GSGP for classification (Bakurov, 2021; Bakurov et al., 2019; Bakurov et al., 2022). They are presented in *Table 7*.

parameter	brief description	value
population size	number of individuals	500
generations	maximum number of iterations	2000
crossover rate	probability of applying crossover	0.0
mutation rate	probability of applying mutation	1.0
function set	all functions allowed in an individual	+ , - , · , /
terminal set	all terminals allowed in an individual	input variables, random constants
elitism	best individual always survives if true	true
initialization	initialization algorithm of the population	ramped half and half, max. depth = 6

selection	algorithm that selects the individuals	tournament, pressure = 0.1
crossover	algorithm for crossing two individuals	geometric semantic crossover
mutation	algorithm for mutating an individual	geometric semantic crossover
α	steepness of the logistic function	1.0
R	limits of the range of random constants	10

Table 7 – Parameter setup of GSGP

4. RESULTS AND DISCUSSION

This section presents the findings of the study. First, the results regarding the benchmark datasets are presented, followed by the credit card transactions dataset results. In the concluding discussion, the results and their meanings are debated.

4.1. RESULTS ON BENCHMARK DATASETS

4.1.1. Average performances

First, the different classification methods are examined based on their average performance on the 20 benchmark datasets. Thus, the average values of the metrics achieved by each of the approaches after 30 repetitions per dataset are considered. All actual mean values can be found in *Appendix 2 – 4*. Each technique is given a rank per dataset, which expresses its relative performance to the other five techniques, with the best average metric value corresponding to rank one, and the worst consequently to rank six. The average ranks across all datasets achieved by each of the classification methods per metric are shown in *Figure 4*.

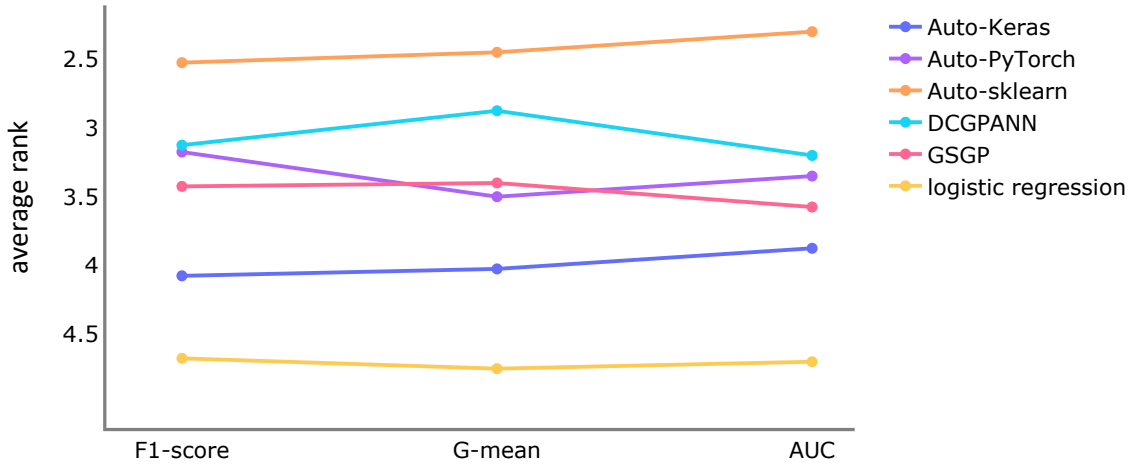


Figure 4 – Average ranks achieved by the different approaches across all benchmark datasets in terms of the respective average achieved metric values per dataset after 30 repetitions

In order to determine whether there is a significant difference between the average ranks of the methods in this case, the Friedman test is applied. For each of the three metrics, a separate Friedman test is performed, independent of the rankings associated with the respective other metrics. Thereby, the null hypothesis is the same for each test, namely that there is no significant difference in the mean ranks of the classification methods with regard to the specific metric. From a significance level of 0.05 and lower, the null hypothesis is rejected and the alternative hypothesis, which states that there are statistically significant differences, is accepted. The p-values resulting from the three Friedman tests are presented in *Table 8*. Since all three p-values are smaller than the significance level, the null hypothesis is rejected in all three cases. Thus, it can be concluded that there are significant differences in the average performances of the six methods, with respect to all three metrics, F1-score, G-mean and AUC.

metric	p-value	significance
F1-score	0.005	true
G-mean	0.002	true
AUC	0.003	true

Table 8 – Results of the Friedman tests for the average performances on the benchmark datasets

In the following, the general differences between the individual techniques, as confirmed by the Friedman test, are investigated in more detail. For this purpose, the average metric values achieved by the individual methods are compared in a pairwise manner, in order to draw conclusions about the performance differences between two methods. For each metric, 15 separate and independent Wilcoxon tests are performed, covering all 15 possible pairs of techniques. The null hypothesis throughout all the tests is that there is no statistically significant difference in the mean ranks of the two respective classification methods under consideration. At a significance level of 0.05 and lower, the null hypothesis is rejected and the alternative hypothesis, which indicates that there is a statistically significant difference between two methods, is accepted. The p-values resulting from the pairwise Wilcoxon tests per metric are presented in *Table 9,* significant p-values are highlighted as bold.

technique	metric	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP
Auto-PyTorch	F1-score	**0.042**	-	-	-	-
	G-mean	**0.050**	-	-	-	-
	AUC	0.107	-	-	-	-
Auto-sklearn	F1-score	**0.003**	0.126	-	-	-
	G-mean	**0.002**	**0.022**	-	-	-
	AUC	**0.003**	**0.027**	-	-	-
DCGPANN	F1-score	**0.048**	0.911	0.126	-	-
	G-mean	**0.013**	0.235	0.173	-	-
	AUC	0.156	0.881	0.117	-	-
GSGP	F1-score	0.126	0.654	0.204	0.970	-
	G-mean	**0.037**	0.455	0.263	0.526	-
	AUC	0.191	0.867	0.083	0.823	-
logistic regression	F1-score	0.681	**0.030**	**0.005**	**0.021**	0.191
	G-mean	0.658	**0.037**	**0.007**	**0.011**	0.076
	AUC	0.641	0.057	**0.008**	**0.024**	0.135

Table 9 – P-values resulting from pairwise Wilcoxon tests for the average performances on the benchmark datasets

The Wilcoxon tests show that there are eight pairs of techniques for which the null hypothesis has to be rejected with respect to at least one metric, meaning that those pairs have statistically significant performance differences on one or more metrics. These eight pairs are shown in *Table 10*. The metrics columns show the exact pairwise results if there is a significant difference for a specific metric. The first number represents the amount of datasets in which the first-mentioned technique achieved a higher average value than the second one, the number in the middle stands for the number of ties, and the last number indicates the number of datasets in which the latter method achieved a higher value. Thereby, the superior numbers, and consequently statistically significantly superior techniques with respect to at least one metric, are highlighted in bold.

pair of techniques	**metrics**		
	F1-score	**G-mean**	**AUC**
Auto-Keras & **Auto-PyTorch**	7-0-**13**	7-2-**11**	not significant
Auto-Keras & **Auto-sklearn**	4-1-**15**	4-2-**14**	4-2-**14**
Auto-Keras & **DCGPANN**	7-0-**13**	6-0-**14**	not significant
Auto-Keras & **GSGP**	not significant	8-0-**12**	not significant
Auto-PyTorch & **Auto-sklearn**	not significant	4-1-**15**	4-1-**15**
Auto-PyTorch & logistic regression	**16**-0-4	**16**-0-4	not significant
Auto-sklearn & logistic regression	**17**-0-3	**16**-1-3	**17**-0-3
DCGPANN & logistic regression	**16**-0-4	**17**-0-3	**17**-0-3

Table 10 – Pairs of techniques with significant differences concerning their average performance

Several observations can be made from these results. Auto-sklearn is the only method that outperforms three others with respect to at least one metric, followed by Auto-PyTorch and DCGPANN, which outperform two others each, and GSGP, which is superior to one other method. In contrast, the most frequently significantly outperformed method on at least one metric is Auto-Keras, with a total of four times, followed by logistic regression with three times and Auto-PyTorch with one time. The only two methods that statistically significantly dominate another across all three metrics are Auto-sklearn and DCGPANN, with the former even managing to do so twice. Only Auto-Keras and logistic regression are not able to significantly outperform other methods, while only Auto-sklearn, DCGPANN and GSGP are not significantly outperformed by any other technique.

4.1.2. Maximum performances

In comparison to chapter 4.1.1, in this section the different classification methods are examined based on their best performance on the 20 benchmark datasets, instead of the average performance. Thus, the maximum values of the metrics achieved by each of the techniques after 30 repetitions per dataset are considered. However, this is the only difference between this part and chapter 4.1.1, meaning that the procedures are identical, and therefore are not explained in detail again here, instead it is referred to the previous chapter for more information about the formal process. All actual best scores can be found in *Appendix 5 – 7*. The average ranks across all datasets achieved by each of the methods per metric, regarding their respective best score per dataset, are shown in *Figure 5*.

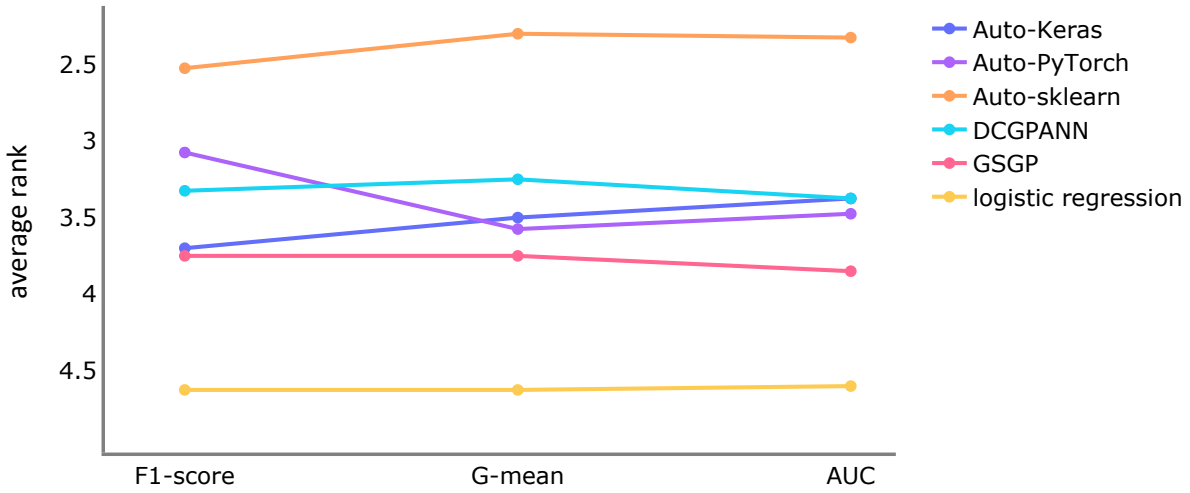


Figure 5 – Average ranks achieved by the different approaches across all benchmark datasets in terms of the respective maximum achieved metric values per dataset after 30 repetitions

Also here, Friedman tests are applied to test for statistically significant differences. *Table 11* presents the results of these tests, which show that for all three metrics the null hypothesis is rejected, meaning that there are statistically significant differences between the average ranks of the methods in terms of the maximum score achieved throughout all metrics.

metric	p-value	significance
F1-score	0.013	true
G-mean	0.006	true
AUC	0.008	true

Table 11 – Results of the Friedman tests for the maximum performances on the benchmark datasets

As in chapter 4.1.1, Wilcoxon tests are conducted next. These compare the techniques in pairs and test the null hypothesis that there are no significant differences in the average ranks between two methods, meaning that their maximum reached metric scores across all datasets do not differ significantly. The p-values resulting from these Wilcoxon tests per metric are presented in *Table 12,* whereby p-values that indicate significance are highlighted as bold.

technique	metric	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP
Auto-PyTorch	F1-score	0.494	-	-	-	-
	G-mean	0.872	-	-	-	-
	AUC	0.952	-	-	-	-
Auto-sklearn	F1-score	**0.015**	0.085	-	-	-
	G-mean	**0.008**	**0.013**	-	-	-
	AUC	**0.013**	**0.011**	-	-	-

DCGPANN	F1-score	0.523	0.523	0.084	-	-
	G-mean	0.570	0.828	**0.015**	-	-
	AUC	0.619	0.586	**0.015**	-	-
GSGP	F1-score	0.936	0.276	**0.014**	0.616	-
	G-mean	0.601	0.811	**0.008**	0.845	-
	AUC	0.809	0.948	**0.007**	0.845	-
logistic regression	F1-score	0.314	**0.013**	**0.001**	**0.023**	0.121
	G-mean	0.295	**0.010**	**0.002**	**0.021**	0.131
	AUC	0.305	**0.010**	**0.002**	**0.033**	0.136

Table 12 – P-values resulting from pairwise Wilcoxon tests for the best performances on the benchmark datasets

This time, the Wilcoxon tests show that there are seven pairs of techniques for which the null hypothesis is rejected with respect to at least one metric, indicating that those pairs have statistically significant performance differences on one or more metrics with regard to their maximum score achieved. All these seven pairs are shown in *Table 13*, in the same fashion as in chapter 4.1.1.

pair of techniques	**metrics**		
	F1-score	**G-mean**	**AUC**
Auto-Keras & **Auto-sklearn**	5-2-**13**	4-2-**14**	5-2-**13**
Auto-PyTorch & **Auto-sklearn**	not significant	6-3-**11**	6-3-**11**
Auto-PyTorch & logistic regression	**14**-1-5	**15**-1-4	**15**-1-4
Auto-sklearn & DCGPANN	not significant	**12**-3-5	**12**-3-5
Auto-sklearn & GSGP	**14**-2-4	**15**-2-3	**16**-2-2
Auto-sklearn & logistic regression	**16**-2-2	**16**-2-2	**16**-2-2
DCGPANN & logistic regression	**13**-4-3	**14**-2-4	**13**-4-3

Table 13 – Pairs of techniques with significant differences concerning their best performance

It is noticeable that the comparison in terms of the maximum achieved scores is more consistent across the different metrics than in the case of the average achieved scores. Five of the seven significantly different pairs here are significantly different throughout all three metrics, whereas in the previous chapter this was the case for only three out of eight. While Auto-sklearn is already the dominant technique in terms of average scores, it is even more so in terms of maximum scores. All five other techniques are statistically significantly outperformed by Auto-sklearn on at least two metrics. The logistic regression, in terms of average scores outperformed by three other methods, is again outperformed by the same three, namely Auto-PyTorch, Auto-sklearn, and DCGPANN. The method that arguably represents the one with the greatest difference between the analysis of chapter 4.1.1 and the one of this chapter is Auto-Keras. While Auto-Keras was significantly outperformed by four other techniques in at least one metric for the average results, it is only outperformed once, by Auto-sklearn, in terms of maximum results.

4.1.3. Standard deviation

After analyzing the techniques based on their average scores as well as their maximum scores in the previous sections, the analysis in this chapter is performed based on the standard deviation. For this purpose, per technique and per dataset, the standard deviation of the scores is calculated after the 30 executions, respectively for the three metrics, F1-score, G-mean and AUC, based on the test partitions of the datasets. Thus, the analysis aims to provide information on how consistently the individual methods perform. The standard deviation scores can be seen in *Appendix 8 – 10*. The average ranks across the 20 datasets in terms of standard deviation are shown in *Figure 6*, with rank one indicating the highest standard deviation and rank six the lowest. Therefore, in this case, a lower rank is desirable as it implies a more stable performance throughout 30 repetitions.

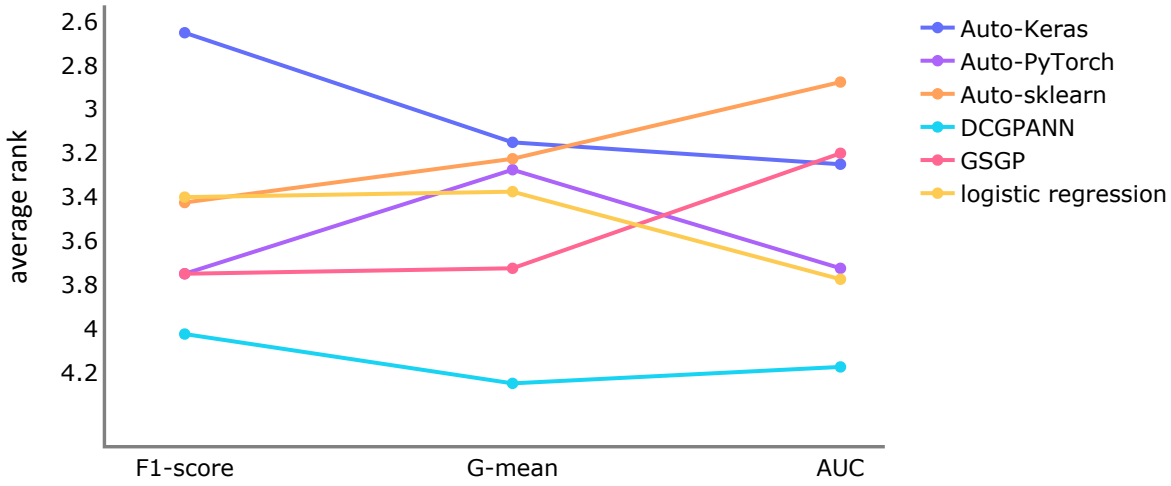


Figure 6 – Average ranks achieved by the different approaches across all benchmark datasets in terms of standard deviation of the metric values per dataset after 30 repetitions

In this analysis, however, the null hypothesis of the Friedman tests could not be rejected for any of the three metrics, contrary to the two preceding analyses. The corresponding p-values can be found in *Appendix 11*. This means that according to the Friedman tests, there is no statistically significant difference in the average rankings in terms of standard deviation of the six techniques under investigation. However, in order to test whether there are any differences in the pairwise comparisons, in contrast to the non-existent differences across all six methods together, the Wilcoxon tests are conducted nevertheless. The detailed results of these are given in *Appendix 12*. Only three pairs that differ statistically significantly can be identified, with none of them being significantly different across all three metrics (see *Appendix 13*). However, it is noteworthy that in each case the three pairs are formed by one of the three state-of-the-art AutoML tools as well as consistently DCGPANN as a counterpart, with the latter always being the technique with the statistically significantly lower standard deviation with respect to at least one metric. Therefore, it can be concluded that DCGPANN provides more consistent results compared to the three AutoML tools.

4.1.4. Generalization ability

Ultimately, the analysis already familiar from chapters 4.1.1 to 4.1.3 is performed a final time, specifically this time regarding the generalization ability of the classification methods. This ability is

measured by the percentage of the training metrics that the methods are able to reach on the test partition. Thus, for example, if a technique achieves an F1-score of 0.8 at training and an F1-score of 0.7 on the test partition of an execution on one dataset, the associated generalization percentage is 0.7/0.8 = 0.875. For the following analysis, the average of these calculated percentage values after 30 executions is used in each case (see *Appendix 14 – 16*). The corresponding average ranks per metric are shown in *Appendix 17*, and the resulting p-values of the Friedman tests, which indicate that there are statistically significant differences in the mean ranks across all metrics, are shown in *Appendix 18*. The detailed p-values of the subsequent Wilcoxon tests are presented in *Appendix 19*. They reveal that the simple logistic regression outperforms all other approaches across all metrics in terms of generalization ability. A possible reason for this is that, since the test partition scores of the logistic regression are already rather poor in comparison, it is also comparatively worse at modeling the relationships during training. Therefore, this property is not necessarily a benefit of the logistic regression, and it is disregarded for the pairwise analysis in this case. *Appendix 20* shows the pairs with statistically significant differences among the three state-of-the-art AutoML tools tested and the two GP approaches under examination. It is noticeable that in all cases methods based on ANNs are outperformed, more precisely Auto-Keras once and DCGPANN three times. The tool that outperforms both in terms of generalization ability is Auto-sklearn, while Auto-PyTorch as well as GSGP are also able to outperform DCGPANN.

4.1.5. Correlations of metric scores and dataset characteristics

Lastly, in this section it is tested whether there are relationships between the average F1, G-mean and AUC scores achieved by the classification methods and the characteristics of the datasets, namely their number of instances, number of features as well as their imbalance ratio. For this purpose, on the one hand, the correlations of these three characteristics with the respective actual average achieved scores are investigated in order to identify relationships for individual techniques themselves. On the other hand, it is examined whether there is a correlation with regard to the relative ranks achieved concerning the average results.

To ensure that the correlations are not biased by datasets with exceptional characteristics, the outliers of the individual properties are removed for this analysis by using the interquartile range method (see *Appendix 21*). Both the visual illustrations as well as the calculated Pearson and Spearman correlation coefficients are reviewed in order to assess whether there are any meaningful relationships. In *Appendix 22,* a visual display of the associations between the characteristics of the datasets as well as the corresponding average F1-scores achieved is presented. *Figure 7* depicts the relationships between the characteristics and the corresponding ranks in terms of average F1-scores. The corresponding correlation coefficients, both Pearson and Spearman, are never above 0.5 or below -0.5 for neither the AutoML tools nor the GP approaches. The detailed coefficients can be found in *Appendix 23* and *Appendix 24*. Therefore, no strong correlations can be detected between these three specific characteristics of a dataset and the associated performances, neither in terms of absolute F1-scores achieved by a single technique nor regarding the relative F1-scores in comparison to the other techniques. The same analyses, but with respect to G-mean and AUC instead of F1-scores, likewise fail to identify any strong correlations (see *Appendix 25 – 28*).

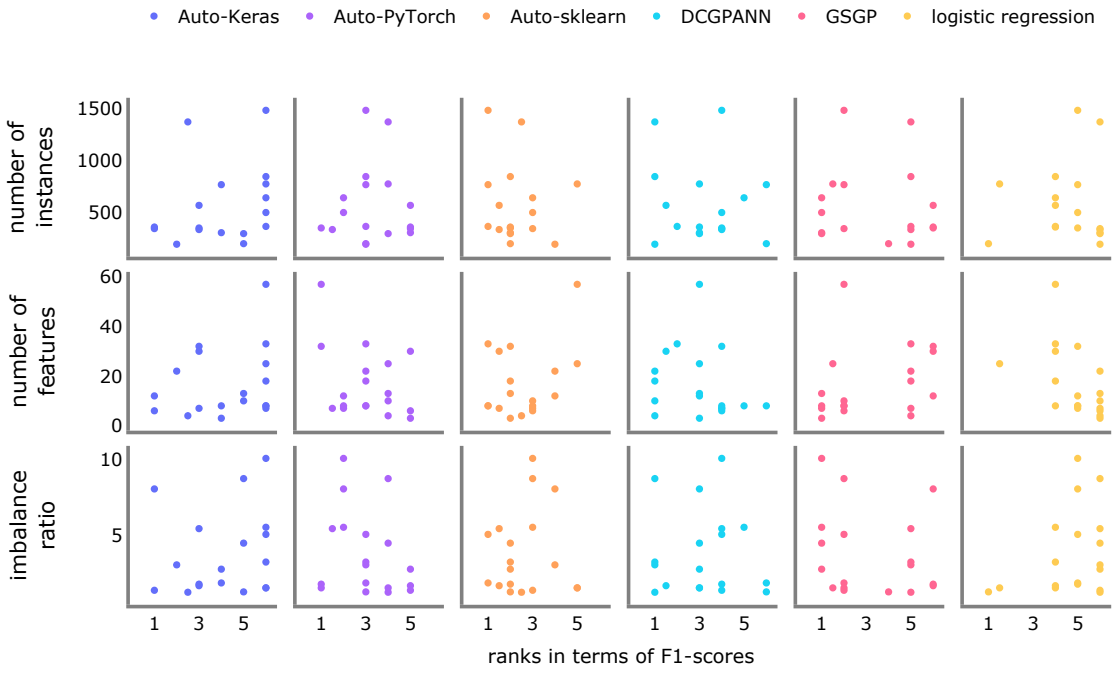


Figure 7 – Scatter plots showing the relationships between achieved ranks in terms of average F1-scores and dataset characteristics

4.2. RESULTS ON CREDIT CARD FRAUD DETECTION

Having outperformed the other techniques on benchmark datasets, Auto-sklearn is finally applied to the real-world problem. This is, as described in chapter 3.3, the task of identifying the fraudulent transactions among a number of credit card transfers. Auto-sklearn is used here in the same default setup as for the benchmark datasets. In order to be able to put the obtained results in perspective, also in this case a simple logistic regression is applied as a reference. *Table 14* shows the F1 scores, G-mean scores, and AUC scores achieved after 30 runs, respectively, as their mean and their best value. The respective best values are marked as bold.

techniques	F1-score		G-mean		AUC	
	mean	best	mean	best	mean	best
Auto-sklearn	**0.862**	**0.909**	**0.904**	**0.945**	**0.909**	**0.947**
logistic regression	0.725	0.786	0.785	0.850	0.809	0.861

Table 14 – Achieved metric values on the credit card transactions dataset

		true class	
		positive (1)	negative (0)
predicted class	positive (1)	99.90	9.57
	negative (0)	22.33	71070.20

		true class	
		positive (1)	negative (0)
predicted class	positive (1)	100	5
	negative (0)	15	71082

Table 15 – Average (left) and, according to the F1-score, best (right) confusion matrices achieved by Auto-sklearn

		true class	
		positive (1)	negative (0)
predicted class	positive (1)	76.13	10.30
	negative (0)	47.30	71068.27

		true class	
		positive (1)	negative (0)
predicted class	positive (1)	77	12
	negative (0)	30	71083

Table 16 – Average (left) and, according to the F1-score, best (right) confusion matrices achieved by logistic regression

It can be clearly observed that Auto-sklearn outperforms the logistic regression across all criteria. Since in practice the best model would be chosen, Auto-sklearn is capable of providing a model with an F1 score of 0.909, a G-mean score of 0.945, or an AUC score of 0.947 for the given task of credit card fraud detection. *Table 15* and *Table 16* give an overview of the corresponding confusion matrices for Auto-sklearn and logistic regression respectively, in each case showing the average matrix after 30 trials as well as the matrix of the trial with the best F1-score. The confusion matrices demonstrate yet again that Auto-sklearn outperforms the logistic regression. The best Auto-sklearn model in terms of F1-score misclassifies only 20 of the total 71,202 test instances, detecting 100 of 115 fraudulent transactions.

4.3. DISCUSSION

During the analysis based on 20 ML benchmark datasets (chapter 4.1), five techniques for imbalanced binary classification are examined and a default logistic regression is used as a reference. The five techniques include two approaches that are based on GP, DCGPANN and GSGP. The other three are the established AutoML tools Auto-Keras, Auto-PyTorch and Auto-sklearn. Both GP methods accomplish the given classification tasks with only one attempt, i.e. only one initialization is performed per execution per dataset followed by the execution of the algorithm until the time limit of 300 seconds is reached. In GSGP, a population of mathematical functions is evolved to fit the dataset as accurately as possible, whereas in DCGPANN, an ANN is trained over numerous epochs for this purpose. Unlike the two GP techinques, the three state-of-the-art AutoML tools handle the task with multiple attempts. Auto-Keras repeatedly initializes new ANNs with varying network architectures and trains

them. Auto-PyTorch and Auto-sklearn try models from a pool of baseline ML algorithms, optimize them, and then build ensembles. Only Auto-PyTorch and Auto-sklearn also include data preprocessing steps. All three of these AutoML tools return the respective best model found after the time has elapsed. All five techniques are used as default. This means that all the results obtained in this work were achieved without having to apply steps such as model selection or hyperparameter optimization. In addition, also the datasets are not modified by any resampling techniques for adjusting the imbalance of them, since it is intended to test how good the respective techniques can deal with the problem of imbalanced data. Therefore, the results are of particular interest for non-ML experts, as no profound knowledge is required for the application of the methods as used here, but only the data and the prebuilt modules, which take care of the whole process of creating a suitable model, are needed.

For this discussion, we assume that one tool only outperforms another if it is superior to it in at least one of the two categories average or maximum performance, since these two are the crucial ones in practice, while the categories standard deviation and generalization ability are rather intended to provide further insights into the techniques' performances. Auto-sklearn, a tool which is based on various baseline ML algorithms, emerged clearly as the best technique from the comparison, as it outperformed three out of five others in terms of average results, and all five in terms of best results. Therefore, it can be concluded that Auto-sklearn is the strongest performing technique, with a great chance of achieving good results, and its use for imbalanced binary classification can be recommended. Auto-PyTorch, again an AutoML tool using multiple baseline ML models, outperformed two other techniques in terms of average performance and one other concerning maximum performance, while in both cases only being outperformed by Auto-sklearn. Thus, Auto-PyTorch is a solid tool for imbalanced binary classification. The third state-of-the-art AutoML tool, Auto-Keras, was less successful than the other two during this study. It is noticeable that Auto-Keras particularly falls behind in terms of average performance, while it scores fairly average in terms of maximum performance, therefore it can be assumed that Auto-Keras is strongly dependent on the respective initialization. Hence, applying Auto-Keras only once bears the risk of getting a poor model, so if applied, it should be tried several times, as it seems to vary considerably from attempt to attempt.

DCGPANN arguably performed at a similar level as Auto-PyTorch, as it also outperformed two others at average scores and one other at maximum scores. It is remarkable that DCGPANN is superior to Auto-Keras, i.e. the approach of DCGPANN seems to be more effective than the one of Auto-Keras, even though the latter generates multiple ANNs while DCGPANN only generates a single one per attempt. Furthermore, it is interesting that DCGPANN achieves significantly more consistent results than all three established AutoML tools. The other GP method, GSGP could only outperform one technique, namely Auto-Keras in terms of average performance. However, it has also been outperformed itself only once, and that was by Auto-sklearn in terms of maximum scores. Therefore, it can be concluded that both DCGPANN and GSGP are also solid techniques to be used for imbalanced binary classification, being even able to keep up with certain state-of-the-art AutoML tools.

Additionally, it is noted that both techniques based on ANNs, namely Auto-Keras and DCGPANN are outperformed by others in terms of generalization ability. Therefore, it can be argued that both are not capable of solving the overfitting problem of ANNs sufficiently. Finally, it is worth noting that throughout the analysis, no strong correlation between performance and the three dataset characteristics number of instances, number of features, and imbalance ratio could be identified.

The application to the real-world problem of fraud detection (chapter 4.2) has once again proven that Auto-sklearn is capable of providing a good model, with maximum scores across all three evaluation metrics higher than 0.9. It clearly outperformed the logistic regression with regard to every metric. This demonstrates that AutoML, in this case in the form of the tool Auto-sklearn, can be used to solve a highly imbalanced real-world classification problem sufficiently well, without any profound knowledge in areas such as sampling techniques, model selection, or hyperparameter optimization needed.

5. CONCLUSION

In this thesis, the two advanced GP methods DCGPANN and GSGP were compared alongside the three state-of-the-art AutoML tools Auto-Keras, Auto-PyTorch and Auto-sklearn in terms of their performances in the domain of imbalanced binary classification. This was accomplished by testing them across 20 benchmark datasets, focusing on the respective scores achieved for the three evaluation metrics, F1-score, G-mean score, and AUC score. In addition to the respective average as well as maximum scores, which were crucial for determining success, the consistency of the performance as well as the generalization ability of the techniques were examined in order to gain further insights into the techniques' individual capabilities.

The analysis showed that the two GP methods were not statistically significant outperformed by any of the three established AutoML techniques in terms of average performance, and that they were both outperformed only by Auto-sklearn in terms of best performance. Therefore, to answer the first part of the research question stated in the introduction, it can be concluded that the GP concepts DCGPANN and GSGP are indeed competitive compared to the three state-of-the-art AutoML tools when applied to imbalanced binary classification, making their use in practice highly recommended. Nevertheless, Auto-sklearn emerged as the strongest performer out of all of them, thus, the answer to the second part of the research question, which of the five methods is the most successful one overall, is conclusively Auto-sklearn.

Furthermore, it was found that the performances of the individual techniques do not strongly correlate with neither the number of instances, nor the number of features or the imbalance ratio of the datasets. Finally, it was demonstrated that the best performing technique, namely the AutoML tool Auto-sklearn, is successfully applicable to the real-world task of fraud detection, wherein it was able to provide a model with a score higher than 0.9 across all three metrics F1-score, G-mean and AUC, completely without the need of applying any additional steps such as resampling techniques for balancing or hyperparameter optimization.

To conclude, it is worth noting that each of the five techniques outperformed all others approaches for at least one dataset, so it is advisable in practice to apply not just one, but multiple techniques in order to maximize the chance of obtaining the best possible final model. Therefore, all the concepts under analysis from the GP and AutoML fields can be considered to be highly beneficial for ML users that apply them.

6. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

Although this thesis contributes valuable insights about the five advanced ML techniques under analysis for literature and practice, it also has its limitations which in turn provide opportunities for further research. For instance, this comparative analysis is limited to imbalanced binary classification problems. However, it would be of interest to investigate how the techniques perform in other subdomains of supervised learning, i.e., multiclass classification and regression, thus, future research on the five techniques under consideration should address these areas as well. Moreover, this comparative analysis is based on 20 benchmark datasets, which are relatively small regarding their number of instances. It would, therefore, also be insightful to compare the methods on multiple, much larger and eventually more complex datasets in order to be able to identify further strengths or eventual shortcomings of them. Furthermore, the five techniques do not cover the whole range of possible GP and AutoML approaches, with especially the latter offering numerous further possibilities that could be used to extend a future analysis. Considering this and taking into account that the relative performance of each technique varies from dataset to dataset, developing an all-in-one AutoML tool that automatically applies a variety of GP approaches and AutoML tools in order to provide the user with the best possible model could lead to the creation of a powerful instrument for ML applicants, and thus should be addressed in future research.

7. BIBLIOGRAPHY

Bacanin, N., Bezdan, T., Venkatachalam, K., Zivkovic, M., Strumberger, I., Abouhawwash, M., & Ahmed, A. B. (2021). Artificial Neural Networks Hidden Unit and Weight Connection Optimization by Quasi-Refection-Based Learning Artificial Bee Colony Algorithm. *IEEE Access*, *9*, 169135–169155. https://doi.org/10.1109/ACCESS.2021.3135201

Badem, H., Basturk, A., Caliskan, A., & Yuksel, M. E. (2017). A new efficient training strategy for deep neural networks by hybridization of artificial bee colony and limited–memory BFGS optimization algorithms. *Neurocomputing*, *266*, 506–526. https://doi.org/10.1016/j.neucom.2017.05.061

Bakurov, I. (2021). *GPOL - General Purpose Optimization Library*. Gitlab. https://gitlab.com/ibakurov/general-purpose-optimization-library

Bakurov, I., Castelli, M., Fontanella, F., Scotto di Freca, A., & Vanneschi, L. (2022). A novel binary classification approach based on geometric semantic genetic programming. *Swarm and Evolutionary Computation*, *69*, 101028. https://doi.org/10.1016/j.swevo.2021.101028

Bakurov, I., Castelli, M., Fontanella, F., & Vanneschi, L. (2019). A Regression-like Classification System for Geometric Semantic Genetic Programming. *Proceedings of the 11th International Joint Conference on Computational Intelligence*, 40–48. https://doi.org/10.5220/0008052900400048

Bakurov, I., Castelli, M., Gau, O., Fontanella, F., & Vanneschi, L. (2021). Genetic programming for stacked generalization. *Swarm and Evolutionary Computation*, *65*, 100913. https://doi.org/10.1016/j.swevo.2021.100913

Batista, J. E., Cabral, A. I. R., Vasconcelos, M. J. P., Vanneschi, L., & Silva, S. (2021). Improving Land Cover Classification Using Genetic Programming for Feature Construction. *Remote Sensing*, *13*(9), 1623. https://doi.org/10.3390/rs13091623

Berg, J., & Nyström, K. (2018). A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, *317*, 28–41. https://doi.org/10.1016/j.neucom.2018.06.056

Castelli, M., Beretta, S., & Vanneschi, L. (2013). A hybrid genetic algorithm for the repetition free longest common subsequence problem. *Operations Research Letters*, *41*(6), 644–649. https://doi.org/10.1016/j.orl.2013.09.002

Chollet, F., et al. (2015). *Keras*. Github. https://github.com/fchollet/keras

Douzas, G., & Bacao, F. (2018). Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Systems with Applications*, *91*, 464–471. https://doi.org/10.1016/j.eswa.2017.09.030

Douzas, G., & Bacao, F. (2019). Geometric SMOTE a geometrically enhanced drop-in replacement for SMOTE. *Information Sciences*, *501*, 118–135. https://doi.org/10.1016/j.ins.2019.06.007

Ezugwu, A. E., Shukla, A. K., Nath, R., Akinyelu, A. A., Agushaka, J. O., Chiroma, H., & Muhuri, P. K. (2021). Metaheuristics: a comprehensive overview and classification along with bibliometric analysis. *Artificial Intelligence Review*, *54*(6), 4237–4316. https://doi.org/10.1007/s10462-020-09952-0

Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., & Hutter, F. (2020). Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *ArXiv E-Prints*. https://doi.org/10.48550/arXiv.2007.04074

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., & Hutter, F. (2015). Efficient and Robust Automated Machine Learning. *Advances in Neural Information Processing Systems*, *28*. https://doi.org/10.1007/978-3-030-05318-5_6

Fonseca, J., Douzas, G., & Bacao, F. (2021). Increasing the Effectiveness of Active Learning: Introducing Artificial Data Generation in Active Learning for Land Use/Land Cover Classification. *Remote Sensing*, *13*, 2619. https://doi.org/10.3390/rs13132619

Gijsbers, P., & Vanschoren, J. (2019). GAMA: Genetic Automated Machine learning Assistant. *Journal of Open Source Software*, *4*(33), 1132. https://doi.org/10.21105/joss.01132

Guyon, I. (2003). Design of experiments of the NIPS 2003 variable selection benchmark. *NIPS 2003 Workshop on Feature Extraction*.

Haynes, W. (2013). Wilcoxon Rank Sum Test. *Encyclopedia of Systems Biology*, 2354–2355. https://doi.org/10.1007/978-1-4419-9863-7_1185

He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, *21*(9), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

He, X., Zhao, K. & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems, 212*, 106622. https://doi.org/10.1016/j.knosys.2020.106622

Izzo, D., Biscani, F., & Mereta, A. (2017). Differentiable Genetic Programming. *European Conference on Genetic Programming*, 35–51. https://doi.org/10.48550/arxiv.1611.04766

Jeon, Y., Lee, M., & Choi, J. Y. (2021). Differentiable Forward and Backward Fixed-Point Iteration Layers. *IEEE Access*, *9*, 18383–18392. https://doi.org/10.1109/ACCESS.2021.3053764

Jiacheng, L., & Lei, L. (2020). A Hybrid Genetic Algorithm Based on Information Entropy and Game Theory. *IEEE Access*, *8*, 36602–36611. https://doi.org/10.1109/ACCESS.2020.2971060

Jin, H., Song, Q., & Hu, X. (2019). Auto-Keras: An Efficient Neural Architecture Search System. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1946–1956. https://doi.org/10.48550/arXiv.1806.10282

Karmaker, S. K., Hassan, Md. M., Smith, M. J., Xu, L., Zhai, C., & Veeramachaneni, K. (2022). AutoML to Date and Beyond: Challenges and Opportunities. *ACM Computing Surveys*, *54*(8), 1–36. https://doi.org/10.1145/3470918

Komaris, D.-S., Perez-Valero, E., Jordan, L., Barton, J., Hennessy, L., O'Flynn, B., & Tedesco, S. (2019). Predicting Three-Dimensional Ground Reaction Forces in Running by Using Artificial Neural Networks and Lower Body Kinematics. *IEEE Access*, *7*, 156779–156786. https://doi.org/10.1109/ACCESS.2019.2949699

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, *4*(2), 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), 251–284. https://doi.org/10.1007/s10710-010-9112-3

Koza, J. R., Keane, M. A., Bennett III, F. H., Yu, J., Mydlowec, W., & Stiffelman, O. (1999). Searching for the Impossible using Genetic Programming. *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*, *2*, 1083–1091. https://doi.org/10.5555/2934046.2934067

Krawiec, K., & Pawlak, T. (2013). Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. *Genetic Programming and Evolvable Machines*, *14*(1), 31–63. https://doi.org/10.1007/s10710-012-9172-7

Li, Q., Luo, Z., Chen, H., & Li, C. (2022). An Overview of Deeply Optimized Convolutional Neural Networks and Research in Surface Defect Classification of Workpieces. *IEEE Access*, *10*, 26443–26462. https://doi.org/10.1109/ACCESS.2022.3157293

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhopf, T., Sass, R., & Hutter, F. (2022). SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research*, *23*, 1–9. https://doi.org/10.48550/arXiv.2109.09831

Lohn, J. D., Hornby, G. S., & Linden, D. S. (2008). Human-competitive evolved antennas. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *22*(3), 235–247. https://doi.org/10.1017/S0890060408000164

Manngård, M., Kronqvist, J., & Böling, J. M. (2018). Structural learning in artificial neural networks using sparse optimization. *Neurocomputing*, *272*, 660–667. https://doi.org/10.1016/j.neucom.2017.07.028

Mao, S., & Sejdic, E. (2022). A Review of Recurrent Neural Network-Based Methods in Computational Physiology. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21. https://doi.org/10.1109/TNNLS.2022.3145365

Märtens, M., & Izzo, D. (2019). Neural network architecture search with differentiable cartesian genetic programming for regression. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 181–182. https://doi.org/10.1145/3319619.3322003

Mecheter, I., Abbod, M., Zaidi, H., & Amira, A. (2022). Brain MR images segmentation using 3D CNN with features recalibration mechanism for segmented CT generation. *Neurocomputing*, *491*, 232–243. https://doi.org/10.1016/j.neucom.2022.03.039

Miller, J. F., & Smith, S. L. (2006). Redundancy and computational efficiency in Cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, *10*(2), 167–174. https://doi.org/10.1109/TEVC.2006.871253

Mitchell, M. (1995). Genetic Algorithms: An Overview. *Complexity*, *1*(1), 31–39. https://doi.org/10.1002/cplx.6130010108

Moraglio, A., Krawiec, K., & Johnson, C. G. (2012). Geometric Semantic Genetic Programming. *International Conference on Parallel Problem Solving from Nature*, 22–31. https://doi.org/10.1007/978-3-642-32937-1_3

Pasa, L., Testolin, A., & Sperduti, A. (2015). Neural Networks for Sequential Data: a Pre-training Approach based on Hidden Markov Models. *Neurocomputing*, *169*, 323–333. https://doi.org/10.1016/j.neucom.2014.11.081

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, *32*, 8024–8035. https://doi.org/10.48550/arXiv.1912.01703

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., 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

Pozzolo, A. D., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating Probability with Undersampling for Unbalanced Classification. *2015 IEEE Symposium Series on Computational Intelligence*, 159–166. https://doi.org/10.1109/SSCI.2015.33

Sipper, M., Azaria, Y., Hauptman, A., & Shichel, Y. (2005). Attaining Human-Competitive Game Playing with Genetic Programming. *IEEE Transactions on Systems, Man and Cybernetics*. https://doi.org/10.1007/11861201_4

Suchoparova, G., & Neruda, R. (2020). Genens: An AutoML System for Ensemble Optimization Based on Developmental Genetic Programming. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 631–638. https://doi.org/10.1109/SSCI47803.2020.9308582

Szymczyk, P. (2015). Z-transform artificial neural networks. *Neurocomputing*, *168*, 1207–1210. https://doi.org/10.1016/j.neucom.2015.05.001

Theckel Joy, T., Rana, S., Gupta, S., & Venkatesh, S. (2019). A flexible transfer learning framework for Bayesian optimization with convergence guarantee. *Expert Systems with Applications*, *115*, 656–672. https://doi.org/10.1016/j.eswa.2018.08.023

Tran, B., Xue, B., & Zhang, M. (2019). Genetic programming for multiple-feature construction on high-dimensional classification. *Pattern Recognition*, *93*, 404–417. https://doi.org/10.1016/j.patcog.2019.05.006

Turner, A. J., & Miller, J. F. (2015). Introducing a cross platform open source Cartesian Genetic Programming library. *Genetic Programming and Evolvable Machines*, *16*(1), 83–91. https://doi.org/10.1007/s10710-014-9233-1

Turner, A. J., & Miller, J. F. (2017). Recurrent Cartesian Genetic Programming of Artificial Neural Networks. *Genetic Programming and Evolvable Machines*, *18*(2), 185–212. https://doi.org/10.1007/s10710-016-9276-6

Vanneschi, L., Castelli, M., Manzoni, L., & Silva, S. (2013). A New Implementation of Geometric Semantic GP and Its Application to Problems in Pharmacokinetics. *European Conference on Genetic Programming*, 205–216. https://doi.org/10.1007/978-3-642-37207-0_18

Vanneschi, L., Castelli, M., Scott, K., & Trujillo, L. (2019). Alignment-based genetic programming for real life applications. *Swarm and Evolutionary Computation*, *44*, 840–851. https://doi.org/10.1016/j.swevo.2018.09.006

Vanneschi, L., Castelli, M., & Silva, S. (2014). A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines*, *15*(2), 195–214. https://doi.org/10.1007/s10710-013-9210-0

Vanneschi, L., Henriques, R., & Castelli, M. (2017). Multi-objective genetic algorithm with variable neighbourhood search for the electoral redistricting problem. *Swarm and Evolutionary Computation*, *36*, 37–51. https://doi.org/10.1016/j.swevo.2017.04.003

Wang, S., Roger, M., Sarrazin, J., & Lelandais-Perrault, C. (2019). Hyperparameter Optimization of Two-Hidden-Layer Neural Networks for Power Amplifiers Behavioral Modeling Using Genetic Algorithms. *IEEE Microwave and Wireless Components Letters*, *29*(12), 802–805. https://doi.org/10.1109/LMWC.2019.2950801

Wever, M., Tornede, A., Mohr, F., & Hüllermeier, E. (2021). AutoML for Multi-Label Classification: Overview and Empirical Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 43*(9), 3037-3054. https://doi.org/10.1109/TPAMI.2021.3051276

Xie, Y., Chen, H., Ma, Y., & Xu, Y. (2022). Automated design of CNN architecture based on efficient evolutionary search. *Neurocomputing*, *491*, 160–171. https://doi.org/10.1016/j.neucom.2022.03.046

Yin, W., Li, L., & Wu, F.-X. (2022). Deep learning for brain disorder diagnosis based on fMRI images. *Neurocomputing*, *469*, 332–345. https://doi.org/10.1016/j.neucom.2020.05.113

Yoo, Y. (2019). Hyperparameter optimization of deep neural network using univariate dynamic encoding algorithm for searches. *Knowledge-Based Systems*, *178*, 74–83. https://doi.org/10.1016/j.knosys.2019.04.019

Yu, X., Ye, C., & Xiang, L. (2016). Application of artificial neural network in the diagnostic system of osteoporosis. *Neurocomputing*, *214*, 376–381. https://doi.org/10.1016/j.neucom.2016.06.023

Zhang, M., Li, H., Pan, S., Lyu, J., Ling, S., & Su, S. (2021). Convolutional Neural Networks-Based Lung Nodule Classification: A Surrogate-Assisted Evolutionary Algorithm for Hyperparameter Optimization. *IEEE Transactions on Evolutionary Computation*, *25*(5), 869–882. https://doi.org/10.1109/TEVC.2021.3060833

Zhao, B., Li, X., Lu, X., & Wang, Z. (2018). A CNN–RNN architecture for multi-label weather recognition. *Neurocomputing*, *322*, 47–57. https://doi.org/10.1016/j.neucom.2018.09.048

Zimmer, L., Lindauer, M., & Hutter, F. (2021). Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(9), 3079–3090. https://doi.org/10.1109/TPAMI.2021.3067763

APPENDIX

APPENDIX 1: Versions of the Python libraries used to apply the five advanced techniques under examination and the logistic regression

technique	library	version
Auto-Keras	autokeras	1.0.18
Auto-PyTorch	autoPyTorch	0.1.1
Auto-sklearn	auto-sklearn	0.14.7
DCGPANN	dcgpy	1.4.2
GSGP	gpol	as of 1st of April 2022 (no version specified)
logistic regression	scikit-learn	1.0.2

APPENDIX 2: Average achieved F1-scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.565 (5)	0.721 (3)	0.761 (2)	0.563 (6)	0.604 (4)	0.800 (1)
audit	0.608 (6)	0.881 (4)	0.800 (5)	0.935 (3)	0.962 (1.5)	0.962 (1.5)
banknote authentication	0.999 (2.5)	0.998 (4)	0.999 (2.5)	1.000 (1)	0.986 (5)	0.980 (6)
breast cancer	0.955 (3)	0.948 (5)	0.957 (1.5)	0.957 (1.5)	0.946 (6)	0.952 (4)
cleveland	0.078 (5)	0.087 (4)	0.243 (2)	0.178 (3)	0.267 (1)	0.008 (6)
dermatology	0.000 (6)	0.979 (3)	0.991 (1)	0.988 (2)	0.895 (5)	0.959 (4)
ecoli	0.811 (3)	0.819 (1.5)	0.819 (1.5)	0.740 (4)	0.725 (5)	0.654 (6)
eucalyptus	0.453 (6)	0.493 (2)	0.480 (3)	0.460 (5)	0.504 (1)	0.472 (4)
haberman	0.321 (4)	0.293 (5)	0.446 (2)	0.338 (3)	0.458 (1)	0.219 (6)
ionosphere	0.853 (3)	0.896 (1)	0.893 (2)	0.826 (4)	0.739 (6)	0.752 (5)
led	0.005 (6)	0.671 (2)	0.659 (3)	0.640 (4)	0.710 (1)	0.622 (5)
libras	0.844 (1)	0.678 (5)	0.809 (2)	0.803 (3)	0.654 (6)	0.771 (4)
liver	0.649 (1)	0.581 (5)	0.614 (3)	0.586 (4)	0.620 (2)	0.561 (6)
page blocks	0.751 (5)	0.779 (4)	0.781 (3)	0.819 (1)	0.812 (2)	0.717 (6)
parkinsons	0.747 (2)	0.731 (3)	0.726 (4)	0.824 (1)	0.642 (5)	0.639 (6)
pima	0.620 (4)	0.621 (3)	0.661 (1)	0.560 (6)	0.642 (2)	0.613 (5)
spambase	0.824 (6)	0.921 (1)	0.885 (5)	0.903 (3)	0.909 (2)	0.895 (4)
vehicle	0.707 (6)	0.917 (3)	0.937 (2)	0.944 (1)	0.890 (5)	0.912 (4)
vowel	0.981 (1)	0.980 (2)	0.978 (4)	0.979 (3)	0.913 (6)	0.936 (5)
yeast	0.069 (6)	0.577 (3)	0.596 (1)	0.511 (4)	0.587 (2)	0.467 (5)
average rank	**4.075**	**3.175**	**2.525**	**3.125**	**3.425**	**4.675**

APPENDIX 3: Average G-mean scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.461 (6)	0.756 (3)	0.780 (2)	0.615 (5)	0.647 (4)	0.828 (1)
audit	0.201 (6)	0.851 (4)	0.741 (5)	0.942 (3)	0.969 (1)	0.968 (2)
banknote authentication	0.999 (3)	0.999 (3)	0.999 (3)	1.000 (1)	0.988 (5)	0.983 (6)
breast cancer	0.964 (3)	0.958 (5)	0.964 (3)	0.966 (1)	0.956 (6)	0.964 (3)
cleveland	0.119 (5)	0.173 (4)	0.458 (2)	0.349 (3)	0.502 (1)	0.018 (6)
dermatology	0.000 (6)	0.980 (3)	0.999 (1)	0.993 (2)	0.966 (5)	0.971 (4)
ecoli	0.888 (2)	0.886 (3)	0.889 (1)	0.842 (5)	0.859 (4)	0.755 (6)
eucalyptus	0.631 (5)	0.635 (4)	0.674 (2)	0.656 (3)	0.687 (1)	0.609 (6)
haberman	0.458 (4)	0.446 (5)	0.611 (2)	0.506 (3)	0.618 (1)	0.363 (6)
ionosphere	0.878 (3)	0.915 (1)	0.910 (2)	0.853 (4)	0.788 (6)	0.799 (5)
led	0.000 (6)	0.778 (2)	0.759 (4)	0.770 (3)	0.833 (1)	0.732 (5)
libras	0.875 (1)	0.721 (6)	0.845 (2)	0.832 (3)	0.803 (5)	0.818 (4)
liver	0.701 (1)	0.644 (4)	0.669 (2)	0.647 (3)	0.632 (5)	0.631 (6)
page blocks	0.819 (5)	0.863 (4)	0.883 (3)	0.892 (1)	0.884 (2)	0.794 (6)
parkinsons	0.809 (2)	0.797 (4)	0.806 (3)	0.886 (1)	0.762 (5)	0.740 (6)
pima	0.701 (3.5)	0.701 (3.5)	0.738 (1)	0.656 (6)	0.722 (2)	0.691 (5)
spambase	0.847 (6)	0.931 (1)	0.897 (5)	0.920 (3)	0.924 (2)	0.911 (4)
vehicle	0.744 (6)	0.944 (3)	0.963 (2)	0.970 (1)	0.933 (5)	0.940 (4)
vowel	0.990 (1)	0.986 (4)	0.988 (2)	0.987 (3)	0.956 (6)	0.963 (5)
yeast	0.093 (6)	0.676 (3.5)	0.721 (2)	0.676 (3.5)	0.744 (1)	0.587 (5)
average rank	**4.025**	**3.500**	**2.450**	**2.875**	**3.400**	**4.750**

APPENDIX 4: Average AUC scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.665 (4)	0.765 (3)	0.784 (2)	0.623 (6)	0.656 (5)	0.830 (1)
audit	0.578 (6)	0.889 (4)	0.811 (5)	0.944 (3)	0.969 (1)	0.968 (2)
banknote authentication	0.999 (3)	0.999 (3)	0.999 (3)	1.000 (1)	0.988 (5)	0.983 (6)
breast cancer	0.965 (2.5)	0.959 (5)	0.965 (2.5)	0.966 (1)	0.956 (6)	0.964 (4)
cleveland	0.507 (4)	0.501 (5)	0.545 (1)	0.522 (3)	0.531 (2)	0.494 (6)
dermatology	0.500 (6)	0.981 (3)	0.999 (1)	0.993 (2)	0.969 (5)	0.973 (4)
ecoli	0.892 (2)	0.891 (3)	0.894 (1)	0.851 (5)	0.864 (4)	0.781 (6)
eucalyptus	0.687 (5)	0.691 (3)	0.706 (2)	0.690 (4)	0.717 (1)	0.676 (6)
haberman	0.582 (3)	0.558 (4)	0.628 (2)	0.557 (5)	0.632 (1)	0.547 (6)
ionosphere	0.882 (3)	0.917 (1)	0.914 (2)	0.860 (4)	0.797 (6)	0.809 (5)
led	0.500 (6)	0.802 (2)	0.797 (3)	0.795 (4)	0.844 (1)	0.769 (5)
libras	0.887 (1)	0.777 (6)	0.861 (2)	0.851 (3)	0.826 (5)	0.840 (4)
liver	0.707 (1)	0.660 (3)	0.676 (2)	0.657 (4)	0.644 (6)	0.652 (5)
page blocks	0.833 (5)	0.871 (4)	0.889 (2.5)	0.897 (1)	0.889 (2.5)	0.813 (6)
parkinsons	0.825 (2)	0.815 (4)	0.820 (3)	0.892 (1)	0.773 (5)	0.76 (6)
pima	0.711 (5)	0.716 (3)	0.740 (1)	0.667 (6)	0.723 (2)	0.712 (4)
spambase	0.855 (6)	0.932 (1)	0.902 (5)	0.920 (3)	0.925 (2)	0.912 (4)
vehicle	0.790 (6)	0.945 (3)	0.963 (2)	0.970 (1)	0.933 (5)	0.941 (4)
vowel	0.990 (1)	0.986 (4)	0.988 (2)	0.987 (3)	0.956 (6)	0.963 (5)
yeast	0.523 (6)	0.721 (3)	0.748 (2)	0.706 (4)	0.760 (1)	0.663 (5)
average rank	**3.875**	**3.350**	**2.300**	**3.200**	**3.575**	**4.700**

APPENDIX 5: Best achieved F1-scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.868 (4)	0.872 (3)	0.913 (2)	0.727 (6)	0.762 (5)	0.933 (1)
audit	0.879 (6)	0.981 (4)	0.965 (5)	0.985 (2)	0.982 (3)	0.987 (1)
banknote authentication	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	0.997 (6)
breast cancer	0.991 (2.5)	0.982 (5)	1.000 (1)	0.991 (2.5)	0.981 (6)	0.99 (4)
cleveland	0.370 (4)	0.261 (5)	0.474 (1)	0.467 (2)	0.378 (3)	0.118 (6)
dermatology	0.000 (6)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)
ecoli	0.957 (2)	0.963 (1)	0.923 (3)	0.870 (5)	0.900 (4)	0.833 (6)
eucalyptus	0.681 (1)	0.629 (5)	0.655 (3)	0.679 (2)	0.634 (4)	0.585 (6)
haberman	0.609 (2)	0.526 (4)	0.588 (3)	0.478 (5)	0.615 (1)	0.370 (6)
ionosphere	0.971 (2)	0.967 (3)	1.000 (1)	0.933 (4)	0.862 (6)	0.897 (5)
led	0.148 (6)	0.833 (2)	0.818 (3)	0.800 (4.5)	0.846 (1)	0.800 (4.5)
libras	1.000 (2.5)	0.941 (5)	1.000 (2.5)	1.000 (2.5)	0.889 (6)	1.000 (2.5)
liver	0.763 (1)	0.720 (3)	0.725 (2)	0.703 (4)	0.702 (5)	0.687 (6)
page blocks	0.835 (5)	0.867 (2)	0.844 (4)	0.865 (3)	0.869 (1)	0.780 (6)
parkinsons	0.880 (4)	0.923 (2)	0.909 (3)	1.000 (1)	0.870 (5)	0.783 (6)
pima	0.714 (3)	0.671 (5)	0.740 (1)	0.643 (6)	0.728 (2)	0.683 (4)
spambase	0.896 (6)	0.949 (1)	0.948 (2)	0.920 (4.5)	0.929 (3)	0.920 (4.5)
vehicle	0.926 (6)	0.979 (2)	0.977 (3)	0.990 (1)	0.944 (5)	0.968 (4)
vowel	0.993 (3)	0.994 (1.5)	0.990 (4)	0.994 (1.5)	0.950 (6)	0.952 (5)
yeast	0.598 (5)	0.674 (2)	0.707 (1)	0.609 (4)	0.667 (3)	0.564 (6)
average rank	**3.700**	**3.075**	**2.525**	**3.325**	**3.750**	**4.625**

APPENDIX 6: Best achieved G-mean scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.901 (3)	0.899 (4)	0.919 (2)	0.760 (6)	0.790 (5)	0.941 (1)
audit	0.893 (6)	0.981 (4)	0.965 (5)	0.985 (2)	0.984 (3)	0.989 (1)
banknote authentication	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	0.997 (6)
breast cancer	0.991 (2.5)	0.982 (6)	1.000 (1)	0.991 (2.5)	0.984 (5)	0.990 (4)
cleveland	0.616 (4)	0.424 (5)	0.740 (1)	0.678 (2)	0.660 (3)	0.263 (6)
dermatology	0.000 (6)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)
ecoli	0.986 (2)	0.993 (1)	0.980 (3)	0.959 (5)	0.967 (4)	0.901 (6)
eucalyptus	0.825 (2)	0.768 (5)	0.828 (1)	0.781 (4)	0.817 (3)	0.745 (6)
haberman	0.696 (3)	0.614 (5)	0.706 (2)	0.621 (4)	0.708 (1)	0.509 (6)
ionosphere	0.976 (2)	0.975 (3)	1.000 (1)	0.942 (4)	0.886 (6)	0.901 (5)
led	0.000 (6)	0.915 (2)	0.897 (4)	0.901 (3)	0.943 (1)	0.891 (5)
libras	1.000 (2.5)	0.943 (6)	1.000 (2.5)	1.000 (2.5)	0.994 (5)	1.000 (2.5)
liver	0.795 (1)	0.747 (3)	0.760 (2)	0.743 (4)	0.709 (6)	0.741 (5)
page blocks	0.860 (5)	0.931 (3)	0.964 (1)	0.937 (2)	0.922 (4)	0.843 (6)
parkinsons	0.964 (2)	0.947 (3)	0.941 (4)	1.000 (1)	0.935 (5)	0.848 (6)
pima	0.778 (3)	0.763 (4)	0.792 (1)	0.724 (6)	0.780 (2)	0.748 (5)
spambase	0.918 (6)	0.960 (1)	0.958 (2)	0.933 (5)	0.939 (3)	0.934 (4)
vehicle	0.945 (6)	0.994 (2.5)	0.994 (2.5)	0.997 (1)	0.973 (5)	0.983 (4)
vowel	0.998 (2)	0.996 (3)	0.995 (4)	0.999 (1)	0.976 (6)	0.977 (5)
yeast	0.782 (3)	0.742 (5)	0.803 (1)	0.752 (4)	0.795 (2)	0.694 (6)
average rank	**3.500**	**3.575**	**2.300**	**3.250**	**3.750**	**4.625**

APPENDIX 7: Best achieved AUC scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.906 (3)	0.899 (4)	0.919 (2)	0.760 (6)	0.792 (5)	0.942 (1)
audit	0.896 (6)	0.981 (4)	0.966 (5)	0.985 (2)	0.984 (3)	0.989 (1)
banknote authentication	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	0.997 (6)
breast cancer	0.991 (2.5)	0.982 (6)	1.000 (1)	0.991 (2.5)	0.984 (5)	0.990 (4)
cleveland	0.617 (4)	0.567 (5)	0.740 (1)	0.695 (2)	0.670 (3)	0.519 (6)
dermatology	0.500 (6)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)	1.000 (3)
ecoli	0.986 (2)	0.993 (1)	0.981 (3)	0.959 (5)	0.968 (4)	0.903 (6)
eucalyptus	0.831 (1.5)	0.774 (5)	0.831 (1.5)	0.788 (4)	0.820 (3)	0.761 (6)
haberman	0.734 (1)	0.671 (4)	0.727 (2)	0.636 (5)	0.712 (3)	0.605 (6)
ionosphere	0.976 (2)	0.975 (3)	1.000 (1)	0.943 (4)	0.887 (6)	0.906 (5)
led	0.500 (6)	0.915 (2)	0.899 (4)	0.904 (3)	0.944 (1)	0.896 (5)
libras	1.000 (2.5)	0.944 (6)	1.000 (2.5)	1.000 (2.5)	0.994 (5)	1.000 (2.5)
liver	0.795 (1)	0.753 (3)	0.768 (2)	0.743 (5)	0.712 (6)	0.744 (4)
page blocks	0.869 (5)	0.932 (3)	0.964 (1)	0.937 (2)	0.924 (4)	0.853 (6)
parkinsons	0.964 (2)	0.948 (3)	0.941 (4)	1.000 (1)	0.938 (5)	0.851 (6)
pima	0.778 (3)	0.766 (4)	0.792 (1)	0.726 (6)	0.781 (2)	0.756 (5)
spambase	0.918 (6)	0.960 (1)	0.958 (2)	0.934 (4.5)	0.940 (3)	0.934 (4.5)
vehicle	0.945 (6)	0.994 (2.5)	0.994 (2.5)	0.997 (1)	0.973 (5)	0.983 (4)
vowel	0.998 (2)	0.996 (3)	0.995 (4)	0.999 (1)	0.976 (6)	0.977 (5)
yeast	0.792 (3)	0.771 (4)	0.809 (1)	0.764 (5)	0.806 (2)	0.732 (6)
average rank	**3.375**	**3.475**	**2.325**	**3.375**	**3.850**	**4.600**

APPENDIX 8: Standard deviation of F1-scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.321 (1)	0.077 (5)	0.087 (2)	0.081 (4)	0.084 (3)	0.067 (6)
audit	0.111 (1)	0.103 (2.5)	0.103 (2.5)	0.032 (4)	0.021 (5)	0.016 (6)
banknote authentication	0.004 (3)	0.002 (4.5)	0.002 (4.5)	0.000 (6)	0.012 (1)	0.008 (2)
breast cancer	0.026 (2.5)	0.025 (4.5)	0.026 (2.5)	0.019 (6)	0.025 (4.5)	0.031 (1)
cleveland	0.126 (1)	0.095 (4)	0.113 (2.5)	0.113 (2.5)	0.060 (5)	0.030 (6)
dermatology	0.000 (6)	0.044 (3)	0.028 (5)	0.038 (4)	0.084 (1)	0.071 (2)
ecoli	0.068 (5)	0.082 (4)	0.059 (6)	0.101 (2)	0.094 (3)	0.118 (1)
eucalyptus	0.152 (1)	0.080 (6)	0.095 (3)	0.085 (4)	0.081 (5)	0.098 (2)
haberman	0.147 (1)	0.112 (2)	0.081 (6)	0.085 (3)	0.082 (5)	0.083 (4)
ionosphere	0.056 (3)	0.038 (6)	0.055 (4)	0.051 (5)	0.074 (1)	0.062 (2)
led	0.027 (6)	0.106 (4)	0.163 (1)	0.115 (2)	0.091 (5)	0.112 (3)
libras	0.125 (4.5)	0.203 (1)	0.125 (4.5)	0.119 (6)	0.171 (2)	0.142 (3)
liver	0.063 (4)	0.069 (2.5)	0.069 (2.5)	0.060 (5)	0.059 (6)	0.070 (1)
page blocks	0.041 (3)	0.071 (1)	0.058 (2)	0.031 (4)	0.028 (6)	0.030 (5)
parkinsons	0.098 (4)	0.117 (3)	0.126 (1.5)	0.093 (5)	0.126 (1.5)	0.075 (6)
pima	0.057 (1)	0.037 (5)	0.048 (2.5)	0.048 (2.5)	0.033 (6)	0.046 (4)
spambase	0.057 (1)	0.017 (3)	0.056 (2)	0.013 (4)	0.007 (6)	0.010 (5)
vehicle	0.180 (1)	0.032 (3)	0.023 (6)	0.024 (5)	0.035 (2)	0.028 (4)
vowel	0.008 (3)	0.006 (6)	0.007 (5)	0.008 (3)	0.014 (1)	0.008 (3)
yeast	0.162 (1)	0.042 (5)	0.050 (3.5)	0.050 (3.5)	0.035 (6)	0.053 (2)
average rank	**2.650**	**3.750**	**3.425**	**4.025**	**3.750**	**3.400**

APPENDIX 9: Standard deviation of G-mean scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.378 (1)	0.063 (4)	0.071 (2)	0.068 (3)	0.062 (5)	0.056 (6)
audit	0.347 (1)	0.236 (3)	0.275 (2)	0.027 (4)	0.018 (5)	0.013 (6)
banknote authentication	0.003 (3)	0.002 (4)	0.001 (5)	0.000 (6)	0.010 (1)	0.007 (2)
breast cancer	0.015 (5)	0.019 (3)	0.021 (1)	0.013 (6)	0.020 (2)	0.016 (4)
cleveland	0.195 (1)	0.172 (2)	0.163 (3)	0.149 (4)	0.081 (5)	0.067 (6)
dermatology	0.000 (6)	0.042 (3)	0.002 (5)	0.028 (4)	0.065 (1)	0.043 (2)
ecoli	0.049 (5.5)	0.055 (4)	0.049 (5.5)	0.072 (2)	0.069 (3)	0.083 (1)
eucalyptus	0.172 (1)	0.079 (3)	0.077 (4)	0.063 (6)	0.072 (5)	0.087 (2)
haberman	0.150 (1)	0.110 (2)	0.062 (6)	0.075 (4)	0.064 (5)	0.087 (3)
ionosphere	0.045 (3.5)	0.029 (6)	0.050 (2)	0.043 (5)	0.060 (1)	0.045 (3.5)
led	0.000 (6)	0.080 (4)	0.160 (1)	0.084 (3)	0.062 (5)	0.103 (2)
libras	0.093 (6)	0.188 (1)	0.099 (5)	0.101 (4)	0.122 (2)	0.116 (3)
liver	0.051 (5.5)	0.055 (3)	0.057 (2)	0.051 (5.5)	0.054 (4)	0.060 (1)
page blocks	0.027 (4)	0.054 (1)	0.051 (2)	0.032 (3)	0.021 (6)	0.023 (5)
parkinsons	0.088 (4)	0.092 (3)	0.097 (1.5)	0.074 (5)	0.097 (1.5)	0.052 (6)
pima	0.045 (1)	0.033 (4)	0.030 (5)	0.041 (2)	0.029 (6)	0.035 (3)
spambase	0.052 (2)	0.016 (3)	0.053 (1)	0.010 (4)	0.006 (6)	0.009 (5)
vehicle	0.174 (1)	0.027 (3)	0.021 (5)	0.014 (6)	0.022 (4)	0.029 (2)
vowel	0.005 (4.5)	0.005 (4.5)	0.005 (4.5)	0.005 (4.5)	0.010 (1)	0.007 (2)
yeast	0.211 (1)	0.035 (5)	0.052 (2)	0.039 (4)	0.027 (6)	0.046 (3)
average rank	**3.150**	**3.275**	**3.225**	**4.250**	**3.725**	**3.375**

APPENDIX 10: Standard deviation of AUC scores on the test partition of the techniques per dataset

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.168 (1)	0.060 (4)	0.070 (2)	0.065 (3)	0.059 (5)	0.055 (6)
audit	0.139 (1)	0.113 (3)	0.129 (2)	0.025 (4)	0.018 (5)	0.012 (6)
banknote authentication	0.003 (3)	0.002 (4)	0.001 (5)	0.000 (6)	0.010 (1)	0.007 (2)
breast cancer	0.015 (4.5)	0.018 (3)	0.021 (1)	0.013 (6)	0.020 (2)	0.015 (4.5)
cleveland	0.033 (5)	0.036 (4)	0.070 (1)	0.058 (3)	0.065 (2)	0.011 (6)
dermatology	0.000 (6)	0.040 (3)	0.002 (5)	0.026 (4)	0.058 (1)	0.041 (2)
ecoli	0.044 (5.5)	0.050 (4)	0.044 (5.5)	0.065 (1.5)	0.063 (3)	0.065 (1.5)
eucalyptus	0.099 (1)	0.049 (5)	0.057 (2)	0.048 (6)	0.050 (4)	0.052 (3)
haberman	0.066 (1)	0.051 (3.5)	0.048 (5)	0.051 (3.5)	0.053 (2)	0.029 (6)
ionosphere	0.042 (3)	0.028 (6)	0.045 (2)	0.039 (5)	0.051 (1)	0.041 (4)
led	0.000 (6)	0.062 (4)	0.079 (1)	0.063 (3)	0.053 (5)	0.073 (2)
libras	0.078 (6)	0.106 (1)	0.084 (4)	0.083 (5)	0.098 (2)	0.092 (3)
liver	0.048 (4.5)	0.048 (4.5)	0.053 (2)	0.046 (6)	0.049 (3)	0.054 (1)
page blocks	0.022 (4)	0.044 (1.5)	0.044 (1.5)	0.028 (3)	0.019 (5)	0.018 (6)
parkinsons	0.074 (4)	0.075 (3)	0.081 (2)	0.065 (5)	0.084 (1)	0.045 (6)
pima	0.040 (1)	0.028 (6)	0.029 (4.5)	0.034 (2)	0.029 (4.5)	0.030 (3)
spambase	0.043 (2)	0.015 (3)	0.046 (1)	0.010 (4)	0.006 (6)	0.009 (5)
vehicle	0.096 (1)	0.026 (2.5)	0.021 (4.5)	0.014 (6)	0.021 (4.5)	0.026 (2.5)
vowel	0.005 (4.5)	0.005 (4.5)	0.005 (4.5)	0.005 (4.5)	0.009 (1)	0.007 (2)
yeast	0.065 (1)	0.024 (5)	0.036 (2)	0.029 (3)	0.021 (6)	0.028 (4)
average rank	**3.250**	**3.725**	**2.875**	**4.175**	**3.200**	**3.775**

APPENDIX 11: P-values of the Friedman tests for the standard deviation ranks on the benchmark datasets

metric	p-value	significance
F1-score	0.260	false
G-mean	0.414	false
AUC	0.267	false

APPENDIX 12: P-values resulting from pairwise Wilcoxon tests for the standard deviation of the metric scores on the benchmark datasets

technique	metric	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP
Auto-PyTorch	F1-score	0.263	-	-	-	-
	G-mean	0.198	-	-	-	-
	AUC	0.338	-	-	-	-
Auto-sklearn	F1-score	0.170	0.653	-	-	-
	G-mean	0.276	1.000	-	-	-
	AUC	0.728	0.122	-	-	-
DCGPANN	F1-score	**0.020**	0.422	**0.037**	-	-
	G-mean	0.053	**0.031**	0.067	-	-
	AUC	0.171	0.408	**0.027**	-	-
GSGP	F1-score	0.167	0.376	0.376	0.837	-
	G-mean	0.167	0.126	0.305	1.000	-
	AUC	0.627	0.614	0.446	0.313	-
logistic regression	F1-score	0.099	0.502	0.695	0.520	0.668
	G-mean	0.077	0.433	0.422	0.305	0.779
	AUC	0.121	0.314	0.093	0.717	0.135

APPENDIX 13: Pairs of techniques with significant differences concerning their standard deviation

pair of techniques	metrics		
	F1-score	**G-mean**	**AUC**
Auto-Keras & DCGPANN	**16**-1-3	not significant	not significant
Auto-PyTorch & DCGPANN	not significant	**13**-1-6	not significant
Auto-sklearn & DCGPANN	**12**-3-5	not significant	**15**-1-4

APPENDIX 14: Percentage of F1-scores reached on the test partition compared to the train partition

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.565 (5)	0.775 (3)	0.794 (2)	0.563 (6)	0.641 (4)	0.800 (1)
audit	0.612 (6)	0.881 (4)	0.801 (5)	0.935 (3)	0.979 (2)	0.984 (1)
banknote authentication	0.999 (3.5)	0.998 (5)	0.999 (3.5)	1.000 (2)	0.994 (6)	1.001 (1)
breast cancer	0.959 (4)	0.961 (3)	0.965 (2)	0.957 (5)	0.951 (6)	0.967 (1)
cleveland	0.000 (5.5)	0.184 (3)	0.421 (1)	0.178 (4)	0.418 (2)	0.000 (5.5)
dermatology	0.000 (6)	0.983 (3)	0.991 (1)	0.988 (2)	0.895 (5)	0.959 (4)
ecoli	0.908 (2)	0.877 (4)	0.902 (3)	0.740 (6)	0.746 (5)	0.923 (1)
eucalyptus	0.659 (3)	0.602 (5)	0.653 (4)	0.460 (6)	0.672 (2)	0.803 (1)
haberman	0.823 (2)	0.433 (5)	0.730 (3)	0.348 (6)	0.692 (4)	0.862 (1)
ionosphere	0.858 (4)	0.921 (1)	0.910 (2)	0.826 (5)	0.798 (6)	0.861 (3)
led	0.007 (6)	0.908 (1)	0.870 (3)	0.818 (5)	0.902 (2)	0.835 (4)
libras	0.848 (1)	0.745 (5)	0.829 (2)	0.803 (4)	0.683 (6)	0.816 (3)
liver	0.803 (2)	0.656 (5)	0.745 (4)	0.586 (6)	0.792 (3)	0.935 (1)
page blocks	0.843 (4)	0.819 (6)	0.820 (5)	0.894 (3)	0.953 (2)	0.983 (1)
parkinsons	0.790 (3)	0.771 (4)	0.748 (5)	0.824 (2)	0.668 (6)	0.885 (1)
pima	0.757 (4)	0.731 (5)	0.858 (2)	0.560 (6)	0.837 (3)	0.945 (1)
spambase	0.836 (6)	0.939 (3)	0.902 (5)	0.908 (4)	0.986 (1)	0.984 (2)
vehicle	0.734 (6)	0.934 (5)	0.946 (2)	0.944 (3)	0.943 (4)	0.977 (1)
vowel	0.983 (4.5)	0.987 (2)	0.983 (4.5)	0.981 (6)	0.985 (3)	0.996 (1)
yeast	0.102 (6)	0.777 (4)	0.809 (3)	0.511 (5)	0.907 (2)	0.951 (1)
average rank	**4.175**	**3.800**	**3.100**	**4.450**	**3.700**	**1.775**

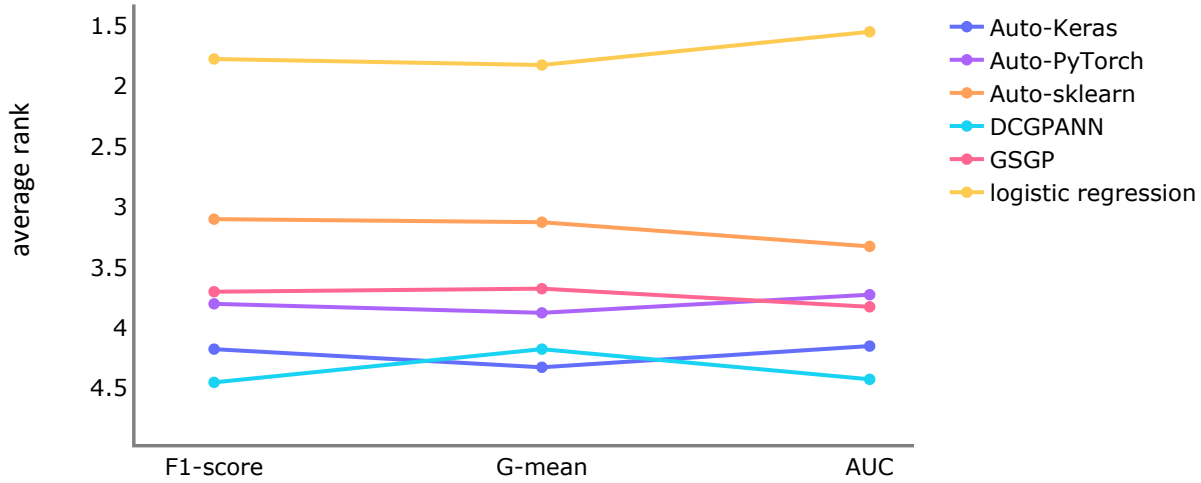
APPENDIX 15: Percentage of G-mean scores reached on the test partition compared to the train partition

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.461 (6)	0.806 (3)	0.809 (2)	0.615 (5)	0.683 (4)	0.828 (1)
audit	0.203 (6)	0.851 (4)	0.741 (5)	0.942 (3)	0.982 (2)	0.987 (1)
banknote authentication	0.999 (4)	0.999 (4)	0.999 (4)	1.000 (1.5)	0.995 (6)	1.000 (1.5)
breast cancer	0.968 (4)	0.970 (3)	0.972 (2)	0.966 (5)	0.961 (6)	0.979 (1)
cleveland	0.000 (5.5)	0.284 (4)	0.587 (2)	0.349 (3)	0.602 (1)	0.000 (5.5)
dermatology	0.000 (6)	0.984 (3)	0.999 (1)	0.993 (2)	0.966 (5)	0.971 (4)
ecoli	0.958 (2)	0.928 (4)	0.940 (3)	0.842 (6)	0.875 (5)	0.964 (1)
eucalyptus	0.820 (2)	0.743 (5)	0.807 (3)	0.656 (6)	0.806 (4)	0.888 (1)
haberman	0.843 (2)	0.595 (5)	0.831 (3)	0.516 (6)	0.795 (4)	0.916 (1)
ionosphere	0.882 (4)	0.937 (1)	0.926 (2)	0.853 (5)	0.840 (6)	0.894 (3)
led	0.000 (6)	0.948 (1)	0.897 (4)	0.905 (3)	0.942 (2)	0.895 (5)
libras	0.879 (1)	0.789 (6)	0.864 (2)	0.832 (4)	0.821 (5)	0.863 (3)
liver	0.840 (2)	0.716 (5)	0.789 (4)	0.647 (6)	0.794 (3)	0.952 (1)
page blocks	0.882 (6)	0.889 (5)	0.903 (4)	0.943 (3)	0.977 (2)	0.991 (1)
parkinsons	0.844 (3)	0.834 (4)	0.823 (5)	0.886 (2)	0.782 (6)	0.942 (1)
pima	0.819 (4)	0.799 (5)	0.891 (2)	0.656 (6)	0.875 (3)	0.962 (1)
spambase	0.858 (6)	0.946 (3)	0.912 (5)	0.925 (4)	0.988 (1.5)	0.988 (1.5)
vehicle	0.771 (6)	0.954 (5)	0.967 (3)	0.970 (2)	0.961 (4)	0.980 (1)
vowel	0.990 (5)	0.991 (3.5)	0.991 (3.5)	0.988 (6)	0.996 (2)	0.999 (1)
yeast	0.123 (6)	0.845 (4)	0.873 (3)	0.676 (5)	0.957 (2)	0.974 (1)
average rank	**4.325**	**3.875**	**3.125**	**4.175**	**3.675**	**1.825**

APPENDIX 16: Percentage of AUC scores reached on the test partition compared to the train partition

	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP	logistic regression
arcene	0.665 (5)	0.815 (2)	0.813 (3)	0.623 (6)	0.692 (4)	0.830 (1)
audit	0.581 (6)	0.889 (4)	0.811 (5)	0.944 (3)	0.982 (2)	0.988 (1)
banknote authentication	0.999 (4)	0.999 (4)	0.999 (4)	1.000 (1.5)	0.996 (6)	1.000 (1.5)
breast cancer	0.968 (4)	0.970 (3)	0.973 (2)	0.966 (5)	0.961 (6)	0.979 (1)
cleveland	0.692 (3)	0.708 (2)	0.687 (4)	0.522 (6)	0.635 (5)	0.975 (1)
dermatology	0.501 (6)	0.985 (3)	0.999 (1)	0.993 (2)	0.969 (5)	0.973 (4)
ecoli	0.961 (2)	0.932 (4)	0.944 (3)	0.851 (6)	0.881 (5)	0.973 (1)
eucalyptus	0.855 (2)	0.797 (5)	0.834 (4)	0.690 (6)	0.836 (3)	0.926 (1)
haberman	0.910 (2)	0.715 (5)	0.847 (3)	0.568 (6)	0.811 (4)	0.970 (1)
ionosphere	0.887 (4)	0.939 (1)	0.929 (2)	0.860 (5)	0.849 (6)	0.902 (3)
led	0.577 (6)	0.959 (1)	0.930 (3)	0.924 (4)	0.950 (2)	0.923 (5)
libras	0.890 (1)	0.835 (6)	0.880 (3)	0.851 (4)	0.842 (5)	0.885 (2)
liver	0.845 (2)	0.733 (5)	0.796 (4)	0.657 (6)	0.801 (3)	0.959 (1)
page blocks	0.895 (6)	0.897 (5)	0.909 (4)	0.947 (3)	0.979 (2)	0.993 (1)
parkinsons	0.853 (3)	0.852 (4)	0.837 (5)	0.892 (2)	0.793 (6)	0.949 (1)
pima	0.827 (4)	0.813 (5)	0.893 (2)	0.667 (6)	0.876 (3)	0.968 (1)
spambase	0.866 (6)	0.947 (3)	0.917 (5)	0.925 (4)	0.988 (1.5)	0.988 (1.5)
vehicle	0.809 (6)	0.955 (5)	0.967 (3)	0.970 (2)	0.962 (4)	0.981 (1)
vowel	0.990 (5)	0.991 (3.5)	0.991 (3.5)	0.988 (6)	0.996 (2)	0.999 (1)
yeast	0.657 (6)	0.877 (4)	0.895 (3)	0.706 (5)	0.962 (2)	0.985 (1)
average rank	**4.150**	**3.725**	**3.325**	**4.425**	**3.825**	**1.550**

APPENDIX 17: Average ranks achieved by the different approaches across all benchmark datasets in terms of average percentage of the metrics reached on the test partition compared to the train partition per dataset after 30 repetitions



APPENDIX 18: P-values of the Friedman tests for the generalization ability ranks on the benchmark datasets

metric	p-value	significance
F1-score	< 0.001	true
G-mean	< 0.001	true
AUC	< 0.001	true

APPENDIX 19: P-values resulting from pairwise Wilcoxon tests for the generalization ability on the benchmark datasets

technique	metric	Auto-Keras	Auto-PyTorch	Auto-sklearn	DCGPANN	GSGP
Auto-PyTorch	F1-score	0.198	-	-	-	-
	G-mean	0.126	-	-	-	-
	AUC	0.153	-	-	-	-
Auto-sklearn	F1-score	**0.035**	0.083	-	-	-
	G-mean	**0.017**	0.071	-	-	-
	AUC	0.064	0.286	-	-	-
DCGPANN	F1-score	0.550	**0.037**	**0.011**	-	-
	G-mean	0.263	0.131	**0.050**	-	-
	AUC	0.955	**0.028**	**0.020**	-	-
GSGP	F1-score	0.117	0.401	0.502	0.057	-
	G-mean	0.067	0.145	0.709	**0.014**	-
	AUC	0.279	0.601	0.455	**0.013**	-
logistic regression	F1-score	**< 0.001**	**0.015**	**0.046**	**0.001**	**0.006**
	G-mean	**< 0.001**	**0.019**	**0.022**	**0.005**	**0.008**
	AUC	**< 0.001**	**0.001**	**0.002**	**< 0.001**	**0.001**

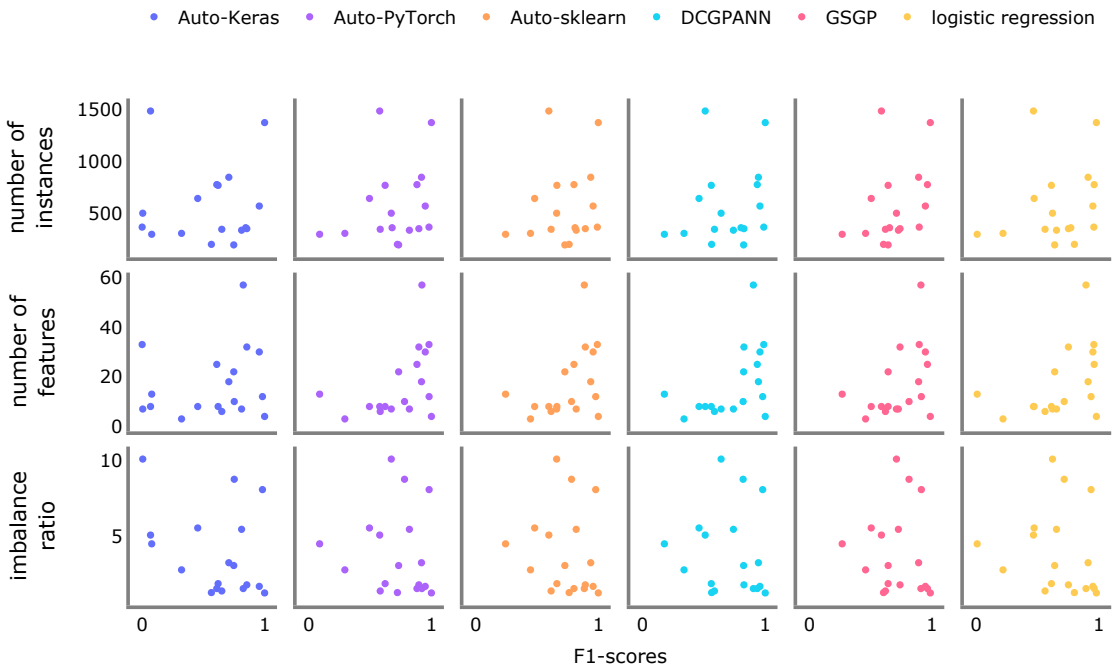
APPENDIX 20: Pairs of techniques with significant differences concerning their generalization ability, without logistic regression

pair of techniques	metrics		
	F1-score	**G-mean**	**AUC**
Auto-Keras & **Auto-sklearn**	7-2-**11**	6-1-**13**	not significant
Auto-PyTorch & DCGPANN	**13**-0-7	not significant	**13**-0-7
Auto-sklearn & DCGPANN	**15**-0-5	**13**-0-7	**14**-0-6
DCGPANN & **GSGP**	not significant	7-0-**13**	7-0-**13**

APPENDIX 21: Removed outliers for correlation analysis

characteristic	dataset	value
number of features	arcene	1998
number of features	libras	90
number of instances	page blocks	5473
number of instances	spambase	4601
number of instances	vowel	9961
imbalance ratio	dermatology	17.300
imbalance ratio	libras	14.000

APPENDIX 22: Scatter plots showing the relationships between achieved average F1-scores and dataset characteristics



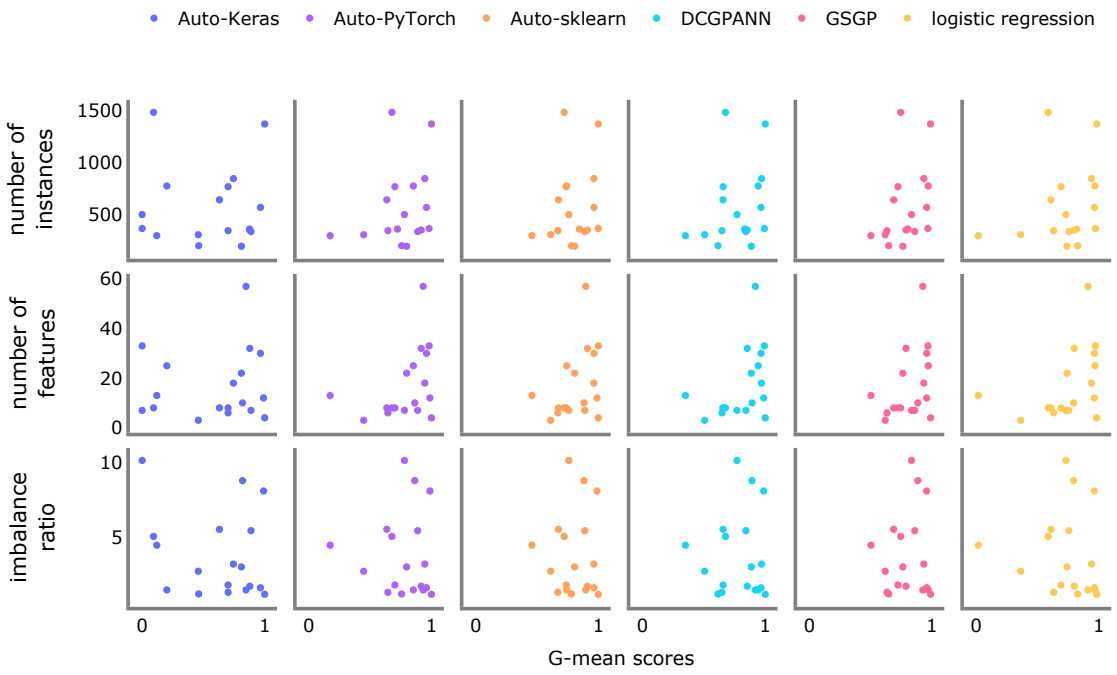
APPENDIX 23: Correlations of actual mean metric scores and dataset characteristics

technique	metric	number of instances		number of features		imbalance ratio	
		pearson	spearman	pearson	spearman	pearson	spearman
Auto-Keras	F1-score	0.015	0.042	0.191	0.128	-0.341	-0.294
	G-mean	-0.030	-0.006	0.091	0.048	-0.235	-0.209
	AUC	0.056	0.048	0.162	0.048	-0.149	-0.197
Auto-PyTorch	F1-score	0.235	0.277	0.458	0.441	-0.109	-0.263
	G-mean	0.249	0.306	0.397	0.407	-0.027	-0.152
	AUC	0.254	0.346	0.460	0.425	-0.003	-0.141
Auto-sklearn	F1-score	0.185	0.282	0.442	0.424	-0.146	-0.261
	G-mean	0.204	0.225	0.378	0.435	-0.005	-0.123
	AUC	0.226	0.294	0.404	0.454	0.024	-0.100
DCGPANN	F1-score	0.198	0.319	0.482	0.457	-0.093	-0.243
	G-mean	0.255	0.463	0.439	0.447	0.014	-0.063
	AUC	0.255	0.461	0.464	0.447	0.052	-0.049
GSGP	F1-score	0.353	0.431	0.435	0.418	-0.094	-0.232
	G-mean	0.385	0.534	0.405	0.391	0.079	-0.077
	AUC	0.387	0.522	0.402	0.397	0.099	-0.077
logistic regression	F1-score	0.218	0.309	0.462	0.454	-0.185	-0.381
	G-mean	0.230	0.297	0.403	0.460	-0.123	-0.381
	AUC	0.238	0.375	0.478	0.445	-0.123	-0.298

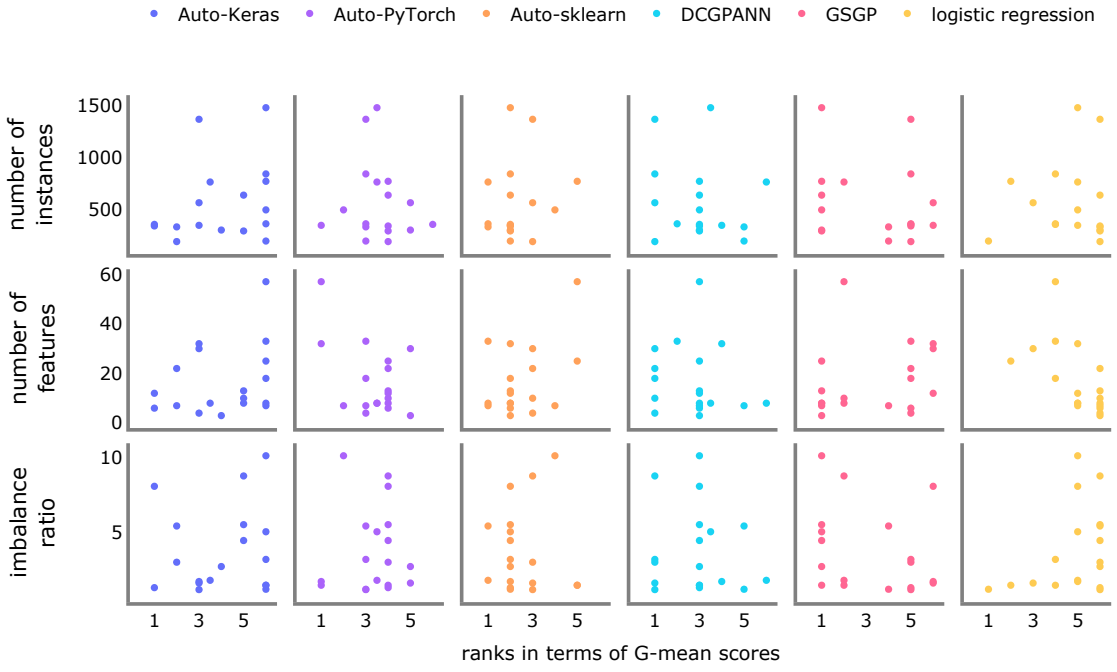
APPENDIX 24: Correlations of average ranks regarding mean metric scores and dataset characteristics

technique	metric	number of instances		number of features		imbalance ratio	
		pearson	spearman	pearson	spearman	pearson	spearman
Auto-Keras	F1-score	0.275	0.409	0.236	0.269	0.162	0.229
	G-mean	0.278	0.347	0.276	0.297	0.087	0.048
	AUC	0.427	0.546	0.266	0.297	0.131	0.138
Auto-PyTorch	F1-score	0.015	-0.025	-0.354	-0.295	-0.266	-0.309
	G-mean	-0.100	-0.150	-0.491	-0.203	0.032	0.118
	AUC	-0.156	-0.226	-0.354	-0.002	0.085	0.135
Auto-sklearn	F1-score	-0.115	-0.145	0.337	0.116	0.058	-0.006
	G-mean	0.178	0.157	0.441	0.204	-0.026	-0.158
	AUC	0.245	0.229	0.488	0.161	-0.181	-0.231
DCGPANN	F1-score	-0.145	-0.098	-0.219	-0.290	-0.055	0.002
	G-mean	-0.175	-0.161	-0.151	-0.190	-0.096	-0.034
	AUC	-0.218	-0.249	-0.297	-0.457	-0.066	-0.018
GSGP	F1-score	-0.103	0.009	0.261	0.391	-0.193	-0.219
	G-mean	-0.203	-0.082	0.217	0.334	-0.280	-0.302
	AUC	-0.270	-0.233	0.181	0.275	-0.295	-0.328
logistic regression	F1-score	0.029	-0.301	-0.503	-0.614	0.307	0.211
	G-mean	0.055	-0.247	-0.599	-0.668	0.367	0.314
	AUC	0.061	-0.236	-0.517	-0.561	0.418	0.435

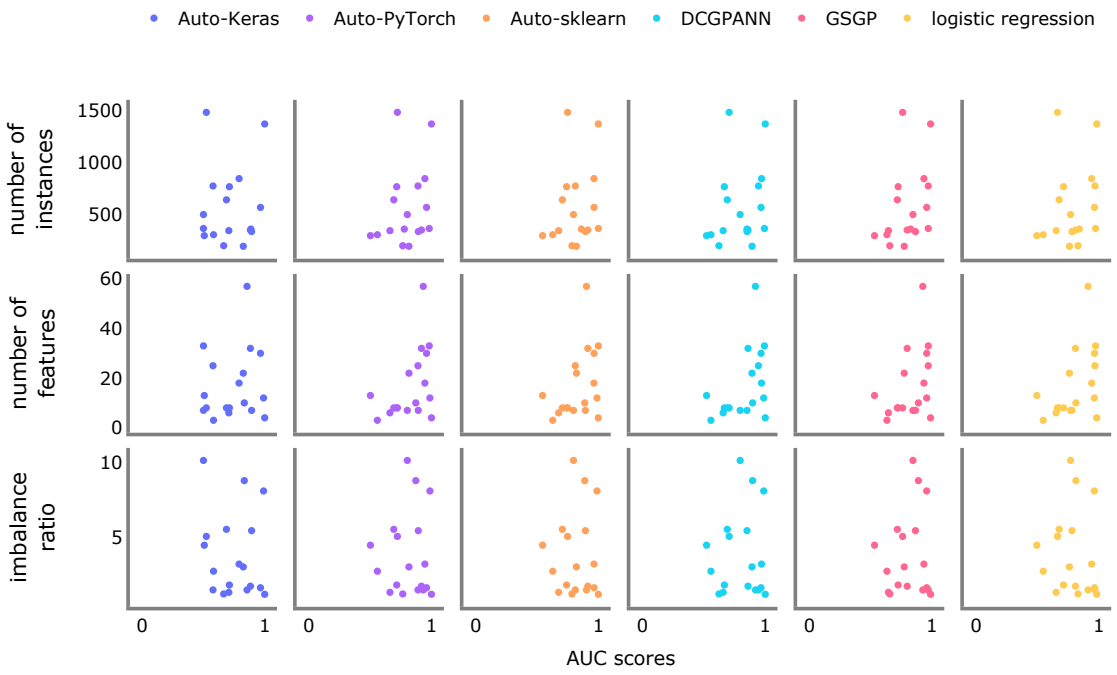
APPENDIX 25: Scatter plots showing the relationships between achieved average G-mean scores and dataset characteristics



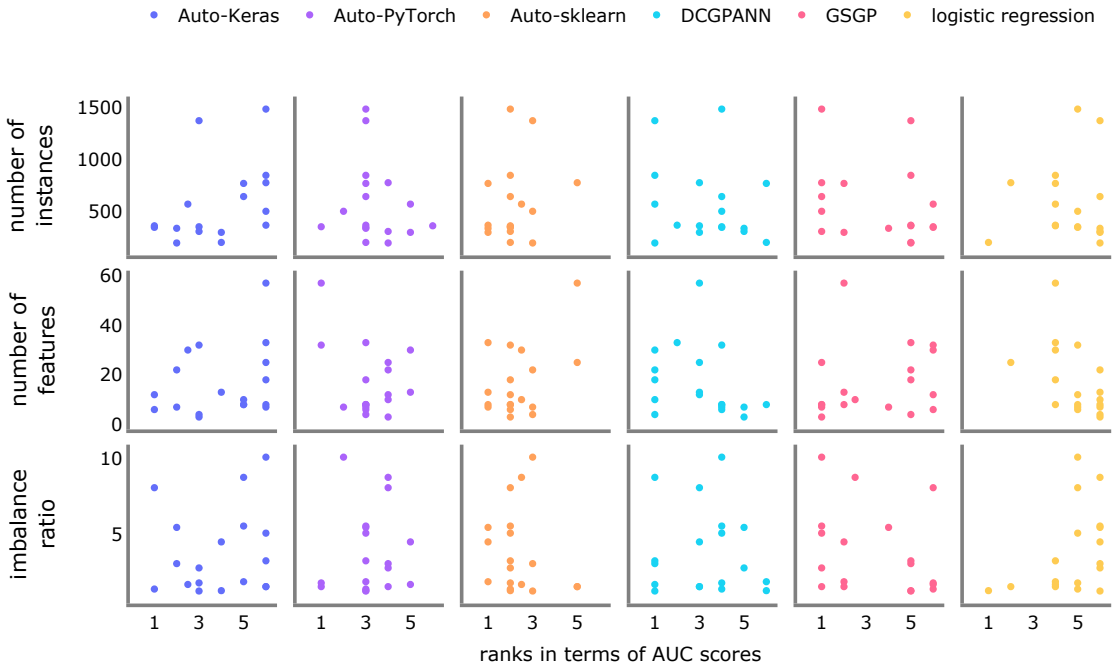
APPENDIX 26: Scatter plots showing the relationships between achieved ranks in terms of average G-mean scores and dataset characteristics



APPENDIX 27: Scatter plots showing the relationships between achieved average AUC scores and dataset characteristics



APPENDIX 28: Scatter plots showing the relationships between achieved ranks in terms of average AUC scores and dataset characteristics



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

Universidade Nova de Lisboa