ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba

Bachelor Thesis

Computer Science Degree

Study of grammatical genetic programming algorithms for mining class association rules

Author
Ventura Lucena Martínez

Assistant Professors
Carlos García-Martínez
José María Luna Ariza

June, 2022

UNIVERSIDAD D CORDOBA

Resumen

El cáncer es una enfermedad de alta incidencia que afecta, entre otros seres vivos, al ser humano, provocando el crecimiento de células anómalas en el cuerpo. Concretamente, el cáncer colorrectal es una enfermedad que afecta hoy en día a una parte considerable de la población, posicionándose como el segundo cáncer más mortífero en los últimos años. Existen múltiples técnicas que intentan combatirlo, desde su prevención hasta su erradicación, sin embargo, queda recorrido para que esto sea una realidad palpable.

Por otro lado, la minería de reglas de asociación consiste en la detección de relaciones entre los elementos de un conjunto de datos. Cuando se fija uno de dichos elementos como objetivo, hablamos de minería de reglas de asociación con clase. Los métodos de la literatura suelen producir demasiadas reglas que, además, tienen poco interés para el experto, pues muestran relaciones que le son bien conocidas. La programación genética gramatical es un método de exploración de soluciones a un problema, que respetan una estructura definida por la gramática. Se ha usado intensamente en diferentes problemas de aprendizaje automático, incluyendo algunos casos de minería de reglas de asociación, aunque no tanto de reglas de asociación con clase.

El objetivo de este proyecto es el de presentar una aproximación a la generación de reglas de asociación con clase a partir de métodos de programación genética. Para ello, se hará uso de un conjunto de datos de cáncer colorrectal proporcionado por el Hospital Universitario Reina Sofía con el que validar la o las propuestas, estudiando los valores de los parámetros y comparando los resultados extraídos, potencialmente interesantes para un experto, con otras propuestas.

Abstract

Cancer is a high incidence disease that affects, among other living beings, the human being, causing the growth of abnormal cells in the body. Specifically, colorectal cancer is a disease that today affects a considerable part of the population, positioning itself as the second deadliest cancer the last years. There are multiple techniques that try to combat it, from its prevention to its eradication, however, there is still a long way to go before this becomes a palpable reality.

On the other hand, association rule mining consists of detecting relationships between the elements of a dataset. When one of these elements is set as a target, we speak of class association rule mining. The methods of the literature usually produce too many rules that, in addition, are out of interest to the expert, since they show relationships that are well known to him. Grammatical genetic programming is a method for exploring solutions to a problem that follow a structure defined by the grammar. It has been used intensively in different machine learning problems, including some association rule mining cases, but not that much on class association rule ones.

The aim of this project is to present an approach for class association rule generation using genetic programming methods. To do this, a colorectal cancer dataset provided by the Reina Sofía University Hospital will be used to validate the proposal(s), studying parameters values and comparing the extracted results, potentially interesting for an expert, with other proposals.

Contents

List of Figures

List of Tables

Listings

List of Algorithms

1. Introduction

Data mining or data exploration is a field of statistics and computer science whose goal is to try to discover patterns in large volumes of data sets. It uses the methods of **machine learning**, **artificial intelligence**, **statistics** and **database systems**. The main objective of the data mining process is to extract information from a data set and expose it in an orderly manner, so that the raw data set is transformed into an understandable and readable structure for later use. In addition to the raw analysis stage, it involves aspects of data management, databases, data processing, model and inference considerations, interest metrics, consideration of computational complexity theory, post-processing of discovered structures, visualization and online updating.

On the other hand, in data mining and machine learning, **association rules** are used to discover facts that occur in common within a given data set. Various methods for learning association rules have been extensively investigated and have turned out to be very interesting for discovering relationships between variables in large data sets, among which the Apriori[1] or FP-Growth[2] algorithms can be highlighted.

Although useful, these algorithms have limitations when the data sets are large, due to their high memory usage. Conversely, if the data set is small, they may find many false associations that just happened by chance. In addition, they spend a great deal of time analyzing the data.

In relation to **evolutionary algorithms (EA)** [2], they are methods of optimization and search for solutions based on the postulates of biological evolution. They maintain a set of entities that represent possible solutions, which combine each other to produce new solutions, in such a way that the most suitable ones are able to prevail over time, evolving towards better solutions. Following the terminology of the theory of evolution, the entities that represent the solutions to the problem are called individuals, and the set of these, population. Individuals can be modified by genetic operators, such as *crossover* or *mutation*, to be later chosen as potential solutions.

In this sense, **grammatical genetic programming** is a **genetic algorithms (GA)**[3] variant in which a complex representation language is used to code the individuals of the population. The most frequently used type of representation is trees, although there are other possibilities. This technique is used to evolve computer programs and other knowledge abstractions, such as mathematical expressions or rule-based systems.

Finally, it is important to mention **grammatical evolution (GE)**[4], a very powerful method for algorithm optimization: given an objective function and a search space (grammar), it is possible to use evolutionary computation methods to develop an algorithm to optimally maximize (or at least efficient) the objective function.

[1] See section 4.1.1, *Apriori algorithm.*
[2] See section 4.1.2, *FP-Growth algorithm.*
[3] See section 4.2, *Genetic programming.*
[4] See section 4.2, *Grammatical evolution.*

This project aims to apply this set of techniques to perform an analysis on a colorectal cancer dataset, in such a way that valuable information that helps to prevent complications in patients suffering from this disease can be extracted.

2. Problem definition

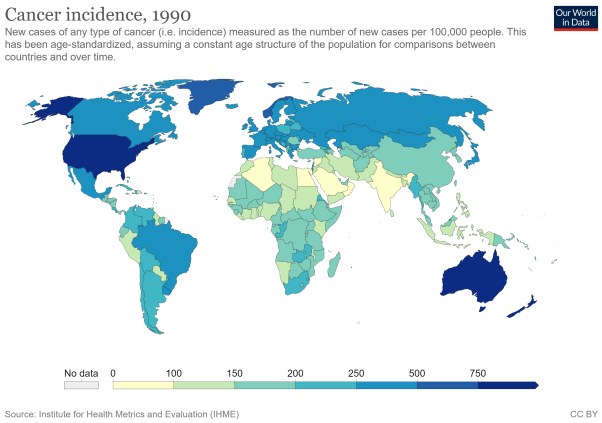
2.1. Real world problem definition

Sticking to the World Health Organization (WHO) **definition of cancer** [3], it is a *"rapid creation of abnormal cells that grow beyond their usual boundaries, and which can then invade adjoining parts of the body and spread to other organs"*.

Although during the last decades there has been a slight increase in the incidence of this type of disease, the number of deaths caused by it has increased by 75%, going from 5.76 million deaths per year in 1990, to 10.08 million deaths per year in 2019, with rising estimations.

Figure 1: Global cancer incidence, from Our World in Data [4].

(a) Cancer incidence in 1990.



(b) Cancer incidence in 2017.

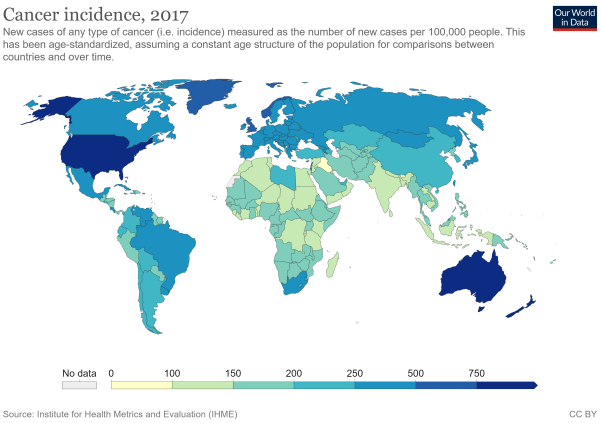
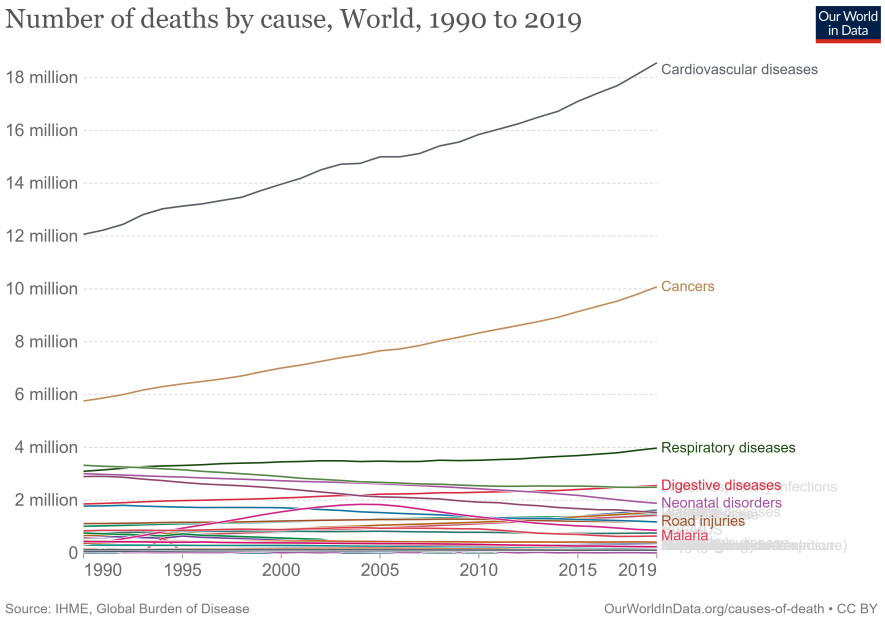


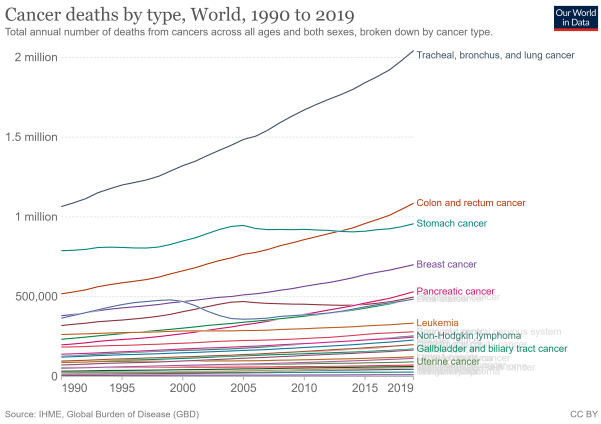
Figure 2: Annual number of deaths by cause (1990-2019), from Our World in Data [4].



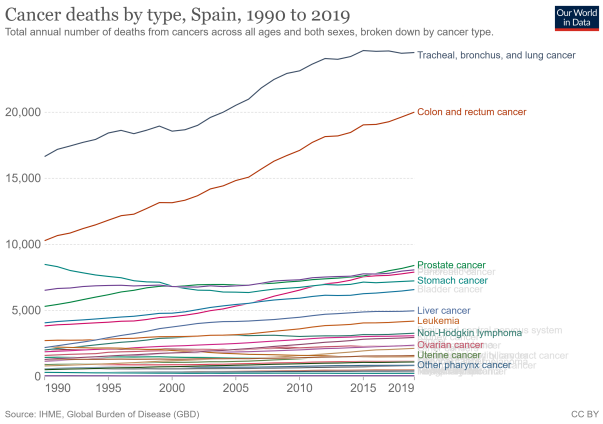
Delving deeper, **colorectal cancer (CRC)** stands out, being the second deadliest cancer worldwide since 2013, and the second deadliest cancer in Spain since 1988 [5]:

Figure 3: Cancer deaths by type (1990-2019), from Our World in Data [4].

(a) World.



(b) Spain.



There are numerous **risk factors** in this type of cancer, such as aging, personal history of colorectal cancer or colorectal polyps, personal history of inflammatory bowel disease, family history of colorectal cancer or adenomatous polyps, et cetera. However, it is estimated that more than one-half of all cases and deaths are attributable to modifiable risk factors, such as smoking, an unhealthy diet, high alcohol consumption, physical inactivity, and excess body weight [6, 7, 8].

In order to reduce its impact on society, there are multiple **CRC prevention methods**: CRC screening tests, surgery, radiation, chemoprevention[5], immunotherapy, stem cell transplants and, in other cases, well-differentiated methods, such as clinical calculators [10].

Therefore, in this sense, it is desired to study a little more the characteristics of the cases with CRC, the possible relationships between some of its symptoms and factors that take part in it, in order to improve the treatment of future cases.

2.1.1. Colorectal cancer dataset

Professionals of the Reina Sofia University Hospital (Cordoba, Spain) have fed a database about CRC, with 1516 patients and 87 attributes, for more than 10 years. This dataset can be divided in three main categories:

- Surgical attributes: surgical information about each patient.
- Pathological attributes: information of the tissues resected from the patient.
- Oncology treatment attributes: about the chemotherapy applied, the follow-up, and final state of the patient as recurrent or survival.

Some of the attributes that can be found in the dataset are shown in table 1:

[5] Among which aspirin stands out, even though they are not recommended in clinical practice. [9]

Table 1: Some features in CRC dataset.

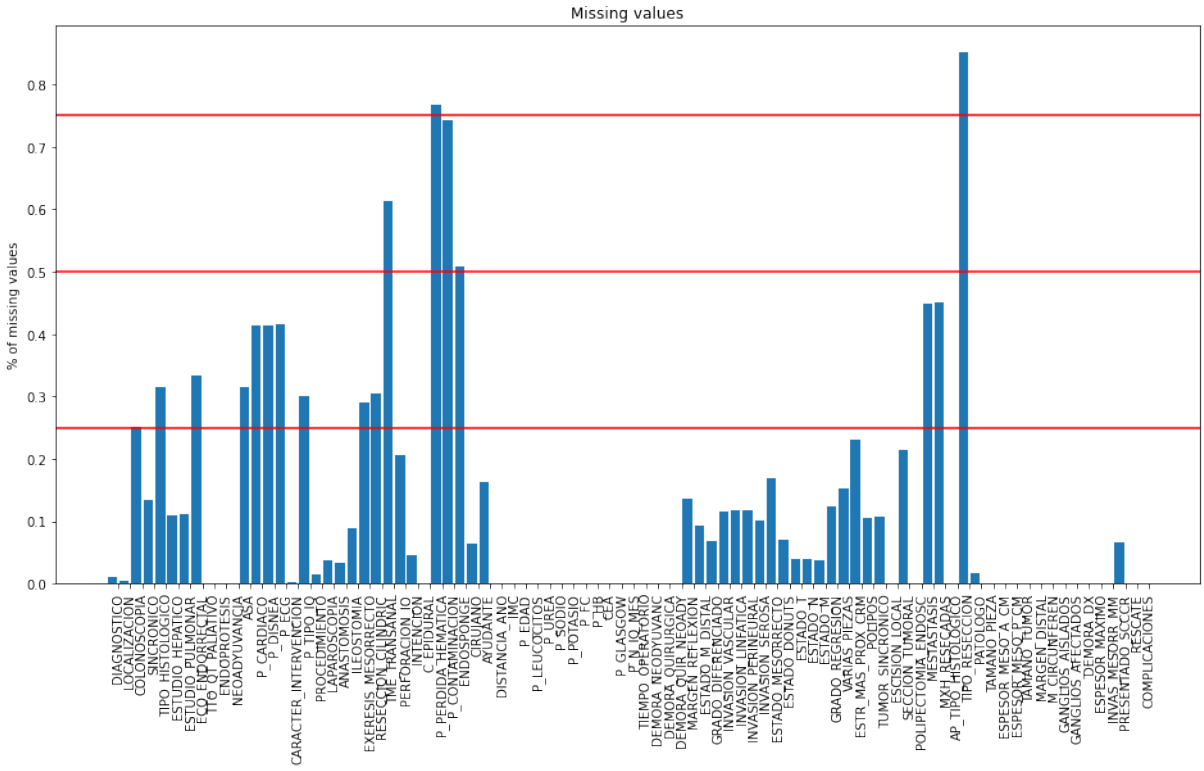
Feature	Description
WUS	it is the period of time that the patient has been Waiting between the diagnosis and the Surgery.
SUR	it identifies the Surgeon.
AS	it identifies the Assistant Surgeon.
ST	it is the used Surgical Technique.
DA	it is the Distance to the Anus.
LS	it is the Length of the Surgery.
CF	it is the Cardiac Frequency before the surgery.
LK	it is the number of LeuKocytes before the surgery.
PL	it identifies if the resected tissue contains PoLyps apart from the primary tumor.
LOC	it is the LOCation of the tumor.
AGE	it is the AGE of the patient.
BMI	it is the Body Mass Index.
BL	it is the Blood Loss during the surgery.
SRT	it is the Size of the Resected Tissue.
NS	it is the N State. It identifies if regional lymph nodes are involved.
NS	it is the patient's Number of Surgeries.
PNI	it identifies if PeriNeural Invasion took place.
DM	it is the Distal Margin.
ILN	it identifies if there are Isolated Lymph Nodes.
AT	it is the Adjuvant Treatment.
ALN	it is the number of Affected Lymph Nodes.
CM	it is the state of the Circumferential Margin.
ONC	it identifies the ONCologyst.
RTAS	it identifies if the patient was treated with Radio-Therapy After the Surgery.
LTR	it identifies the Level of the Tumor Regression after the treatment.
MT	it is the Maximum Thickness of the tumor in an axial section.
END	if the patient suffered from stitch dehiscence, treated with a transanal endosponge procedure.
C	it is one of the objective attributes and it identifies if a patient had any complication. There are 616 patients with C=true, and 900 with C=false.
R	it is the second objective attribute and it identifies if a patient had a Recurrence of the cancer. There are 487 patients with R=true and 1029 with R=false.
OC	it identifies if the patient has been explored in an Oncology Consultation.
TSRD	it is the period of Time between the Surgery and a Recurrence Diagnostic. When a patient has not recurrence, this attribute identifies the period of time between the surgery and the last check-up.
KS	it is the Kind of the Surgery (scheduled or urgent).
ND	it is the Number of Days that the patient has stayed in the hospital after the cancer surgery.
SR	it is the State of the Rings after an intestinal stitch.
WCR	it identifies Where the Cancer Recurrence takes place.
DSD	it is the number of Days between the Surgery and the patient Death. If the patient is still alive, it is the number of days between the surgery and the last checkup.
DSMCD	it is the number of Days between the Surgery and the Metachronous Cancer Diagnostic. When a patient has not metachronous cancer, this attribute identifies the period of time between the surgery and the last check-up.
CD	it is the Cause of a patient Death.
TSLRD	it is the period of Time between the Surgery and the Local Recurrence Diagnostic. When a patient has not local recurrence, this attribute identifies the period of time between the surgery and the last check-up.
PLR	it is the Place of the Local Recurrence.
Endosponge	if the patient suffered from stitch dehiscence, treated with a transanal procedure.
QOR	this evaluates the Quality Of the Resection when he/she suffered from rectal cancer.
WPD	it identifies Where the Patient was Diagnosed.
KC	it is the Kind of used Chemotherapy.
PSCI	it is the Patient State with respect to the level of the Cancer Invasion into the bowel wall.
PDM	if the Patient Died during being Monitored.
HC	it identifies the Histological Classification of the cancer that have been resected.

It should be considered that two facts are remarkable in order to use this dataset for classification purposes:

- The amount of missing values in the dataset is 16491, which is a 12.5% of the data. Figure 4 shows the ratio of missing values per attribute: the 80.45% of attributes

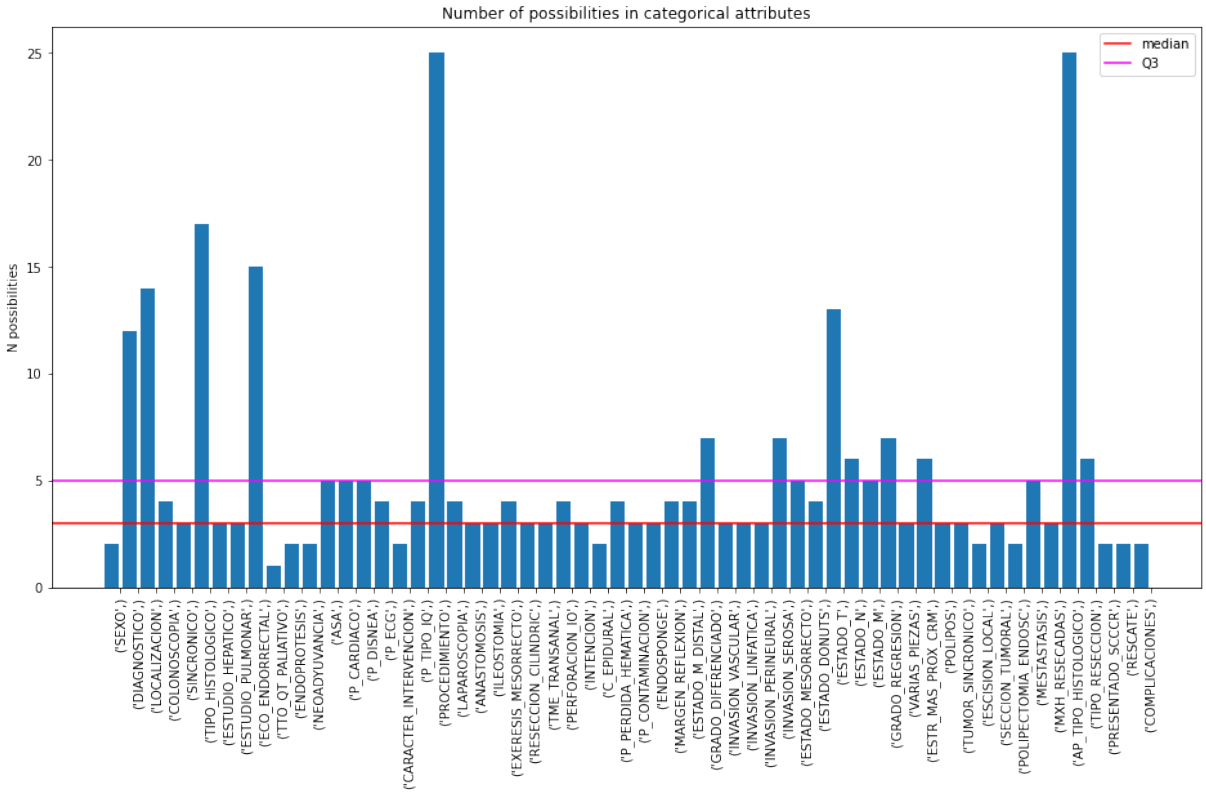
(70) have <= 25% of missing attributes, while the 19.54% remaining (17) have > 25% of missing attributes, considering 25% as a noteworthy threshold.

Figure 4: Histogram of missing values.



- The amount of different values in some attributes is large. The dataset contains a total of 58 categorical attributes, among which 50% (29) have 3 possible values (median), 27.58% have 5 possible values (quartile 3); the remaining 22.41% range from 6 to 25. This makes that categorical features have to be treated independently of each other, and so, they cannot be encoded into the same range of integers.

Figure 5: Number of possibilities in categorical attributes.



2.2. Technical problem definition

As a possible prevention proposal based on the extraction of association rules with grammatical genetic programming, the conditioning factors of the problem must be established, with the support of the **Product Design Specification (PDS)**:

2.2.1. Functionality

The algorithm must:

- Read the input dataset successfully.
- Read a grammar that generates production rules from the dataset.
- Process the data generated by the grammar and build antecedents and consequents, in such a way that association rules are generated.
- Extract interesting metrics from the association rules set, in such a way that valuable information can be obtained from the dataset.

- Filtering and cleaning of extracted results to keep just interesting rules.

2.2.2. Environment

As for the **environment in which the system is intended to be used**, its main users are experts in the field of science and data mining, due to the need to have knowledge of the information technology used. The extracted information must be analyzed, *a posteriori*, by experts related to colorectal cancer.

On the other hand, regarding the **programming environment**, the product is expected to work in conjunction with the following tools:

- Python as a base PonyGE2 language and its relation and facilities with multiple features and libraries related with machine learning and data science.
- Command line based user interface.

2.2.3. Expected Lifespan and Software Maintenance

It is difficult to determine the expected lifespan of a software. It would be pretentious to say that it will have a long life, since better solutions to the same problem may emerge. However, since the source code is hosted on the Github platform, the community can make improvements, keeping the software updated and correcting deficiencies and, finally, extending its life.

As a general idea, its life is expected to span until better approaches emerge and become popular. Nevertheless, the product is expected to be a source of inspirations for other class association rule mining approaches and colorectal cancer studies.

2.2.4. Competence

There are several alternatives that could be related to association rule mining with grammatical genetic programming methods. These have been grouped in table 2, where some of their properties are indicated.

It can be seen that some algorithms do not make use of grammars to shape the rules to be extracted; on the other hand, the memory usage in some approaches is higher than in others, due to the fact that they improve previous or similar versions of other algorithms and, finally, whether or not they maintain (or no information has been found in this regard) a modular structure that helps the development, adaptation and maintenance, among other aspects.

Although the market offers several options[6] for class association rule mining, there is

[6]See section 4.1, "*Frequent Itemset Mining*".

intention of developing a new GP proposal using PonyGE2 library in Python, due to the ease of its structure for its expansion and/or modification.

Table 2: Algorithms competence.

Alg	Specs (Section)	Grammar use	GUI	Memory use	Modular	Language
Apriori	4.1.1	No	No	High	No	Multiple langs.
FP-Growth	4.1.2	No	No	High	No	Multiple langs.
GEVA	4.2.1	Yes	Yes	-	Yes	Java
G3PARM	4.2.1	Yes	No	Moderate	-	Java
CBA2	4.3.1	No	No	Moderate	-	-
CPAR	4.3.1	No	No	Moderate	-	-
FARCHD	4.3.1	No	No	Moderate	-	-
PonyGE2	7.2.2	Yes	No	-	Yes	Python

When an approach is stated to make "Moderate" memory usage, it means that it improves this aspect over previous versions or over other algorithms in the same field. In any case, it is common to see memory problems in data mining algorithms.

2.2.5. External aspects

For the user, the software is accompanied by a fairly minimalist interface, making use of the command line terminal to execute it; in addition to a simple, intuitive and explicit documentation [11] for all the functionalities.

2.2.6. Standardization

For programming, standards of the languages used will be taken into account, in this case, Python for the extension of the PonyGE2 library and the data analysis tools. Delving into the data analysis tools, the following ones stand out: *Matplotlib* [12], *NumPy* [13], *Scipy* [14], *Scikit-Learn* [15] and *Pandas* [16].

2.2.7. Quality and Reliability

Quality It is the ability of a product or service to meet user needs. There are two possible approaches:

- Functional quality: reflects how well it complies with or conforms to a given design, based on functional requirements.
- Structural quality: refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as performance, maintainability, or scalability.

To ensure the quality of the software, the modular development of its different functionalities has been taken into account, in such a way that it is easily updated, reduces its complexity and, in turn, is understandable.

Reliability It is the probability of failure-free operation of a computer program for a specified context. Attention should be paid to:

- Meet the requirements before use.
- Algorithm tuning parameters, such as input data files and its extensions, elite size, amount of generations, population size or the grammar to use, among others.
- Have previous knowledge of the target of the data set, since it may require a modification in the code (By convention, last column of the data set).

2.2.8. Tests

To ensure the reliability and operation of the software, it will be subjected to a series of tests in which the following aspects will be analysed:

- Rule extraction:
  - Amount: $\approx$ 100 rules.
  - Length: 2-8 conditions.
- Metrics extraction: check that the metrics are in the range established by definition and that the data obtained is logical. The following metrics will be tested:
  - Fitness function based on Gini Impurity - range [0, 0.5].
  - Support - range [0, 1]
  - Precision - range [0, 1]
  - Recall - range [0, 1]
  - Lift - range [0, ∞]
  - Leverage - range [-1, 1]
  - Conviction - range [0, ∞]
  - Support (Minimum and average support) - range [0, 1]
  - Confidence (minimum and average confidence) - range [0, 1]
  - Number of rules.
  - Number of antecedent conditions.
  - Used attributes in rules - range [0, 87]

- Uncovered positive patterns - range [0, 1]

- Different approaches' performance for diversification[7] procedure:
  - Hamming distance.
  - Cosine similarity.
  - Similarity of individuals based on the number shared attributes.
  - Importance disimilarity between attributes.

2.2.9. Assurance

Assurance consists of specifying what treatment will be given to user's own data and also security against unauthorized copies.

The software is published on the Github platform and does not store any type of user information, so it is not necessary to present assurance measures in these aspects.

[7] See section 7.2.2, *Diversification.*

3. Objectives

The aim of this research is to obtain a grammatical genetic programming algorithm able to extract interesting association rules, from the provided CRC cancer dataset, that shows connections between attribute values of a database and their relationship with the occurrence of complications, in such a way that this information or methodology might be interesting to experts in the field, being useful, in turn, facing the identification and improvement of preventive methods.

At the same time, the performance of the algorithm used will also be analyzed, comparing it with the experiments[8] carried out using classical methods[9] and observing if the new approach is capable of alleviating the problems presented by said methods.

The objectives in an orderly manner are as follows:

1. Study of proposals for the extraction of class association rules[10].
2. Study of the software available for experimentation with grammatical genetic programming algorithms. For this, PonyGE2 [17] will be used.
3. Design of a proposal for a grammatical genetic programming algorithm for mining interesting class association rules.
4. Results analysis: experimental comparison of the proposal with different parameter settings and other existing approaches.

[8] Approaching the same problem from another point of view.
[9] See section 4.1, *Frequent Itemset Mining*.
[10] See section 4.1, *Frequent Itemset Mining*.

4. Background

Next, what genetic programming and grammatical evolution consist of will be detailed, while introducing some algorithms used in the field of association rule extraction, mainly in the **extraction of frequent data sets**.

4.1. Frequent Itemset Mining

> *"**Frequent Itemset Mining (FIM)** is the task of extracting any existing frequent itemset (having a frequency of occurrence not less than a threshold) in the data. This task was proposed in the early 1990s to discover elements that frequently occurred in the analysis of the market basket. [18]"*
>
> (*"Frequent itemset mining: A 25 years review"*, [19])

Analysis of the market basket If there are 2 items X and Y which are bought frequently, then it is good to put them together in the stores or offer some discount offer on one item when buying another item. This could increase sales. For example, it is likely that if a customer buys milk and bread, they will also buy butter. So, the association rule would be the following:

$$X \rightarrow Y \tag{1}$$

$$[Milk, Bread] \rightarrow [Butter] \tag{2}$$

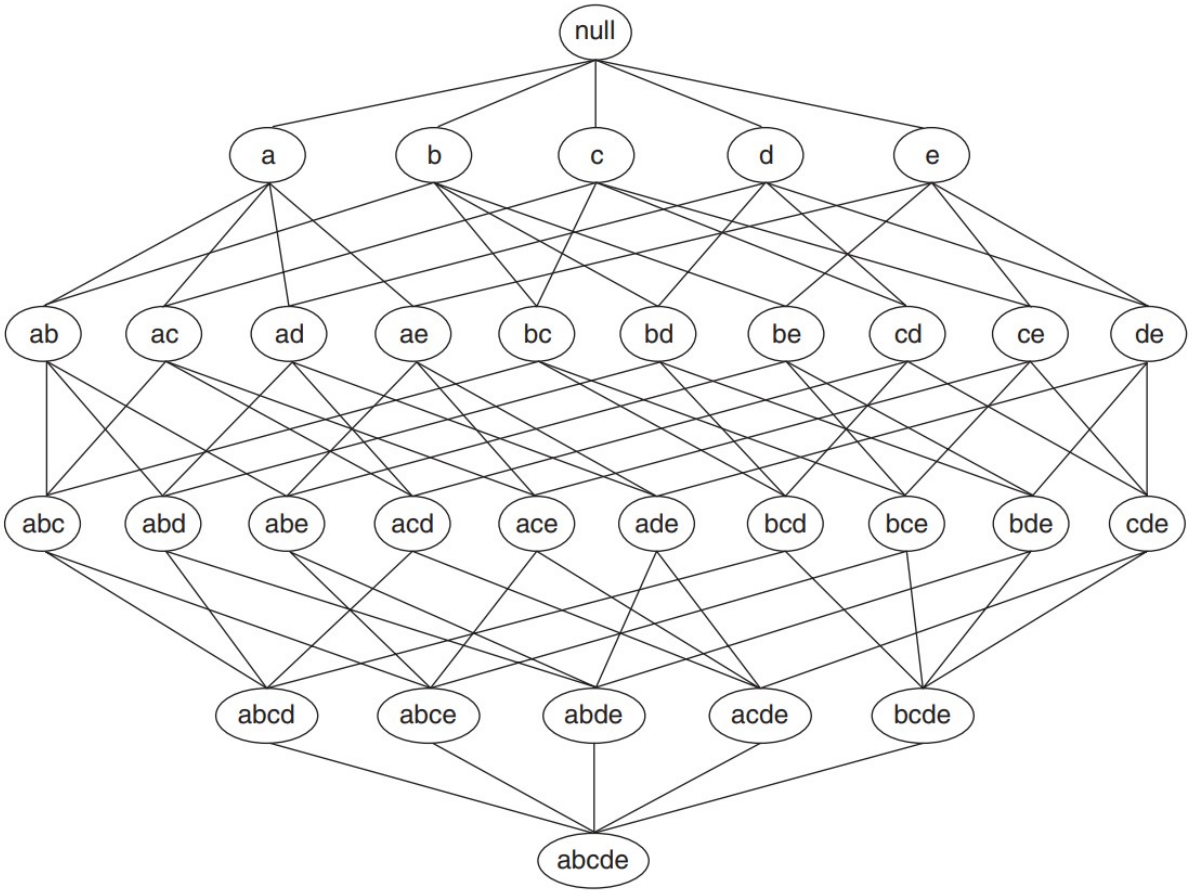
Association rules are made up of two distinct parts or itemsets, *antecedent* and *consequent*:

$$[Antecedent] \rightarrow [Consequent] \tag{3}$$

In short, FIM shows which elements appear together in a transaction or relationship. There are several methods to do this: a **lattice structure**[11] can be used to obtain all the possible itemsets. For instance, the corresponding itemset lattice supposing the following items $I = \{a, b, c, d, e\}$ would be as follows:

[11] The search space of itemsets that need to be explored can be exponentially large, considering that a data set that contains k items can potentially generate up to $2^k - 1$ frequent itemsets, excluding the null set.

Figure 6: Example of itemset lattice, from *"Introduction to data mining"* [20].



4.1.1. Apriori algorithm

Apriori algorithm [21, 22, 23] finds sets of frequent elements in a data set for the *boolean* association rule. The algorithm takes this name because it uses prior knowledge of the frequent properties of the set of elements. An iterative approach or level search is applied where k frequent itemsets are used to find $k + 1$ itemsets.

Furthermore, the algorithm makes use of the so-called **Apriori property**, in which the search space is reduced. This improves the efficiency of level-wise generation of frequent itemsets.

Apriori property All non-empty subsets of the set of frequent elements must be frequent. The key concept of the Apriori algorithm is its support measure **anti-monotonicity**.

Anti-monotone property *"If a condition is violated for some itemset i, then every descendant of i will also violate that condition."* [24]

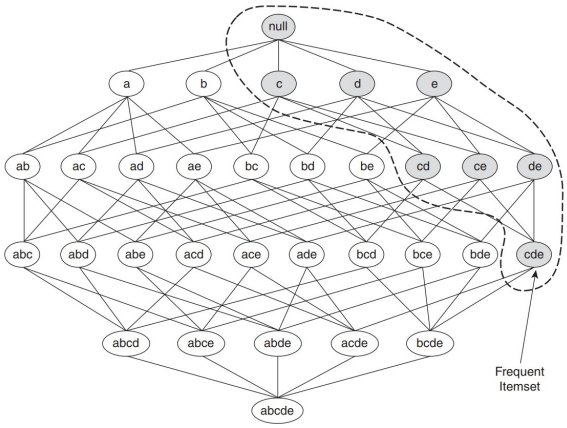
A measure f possesses the anti-monotone property if for every itemset X that is a proper subset of itemset Y, i.e. $X \subseteq Y$, we have $f(X) \geq f(Y)$. For instance, this is, if an itemset is infrequent, all its supersets will be also infrequent. As illustrated in figure 7, all the supersets of an infrequent itemset can be pruned to reduce exponentially the search space.

Considering the support measure:

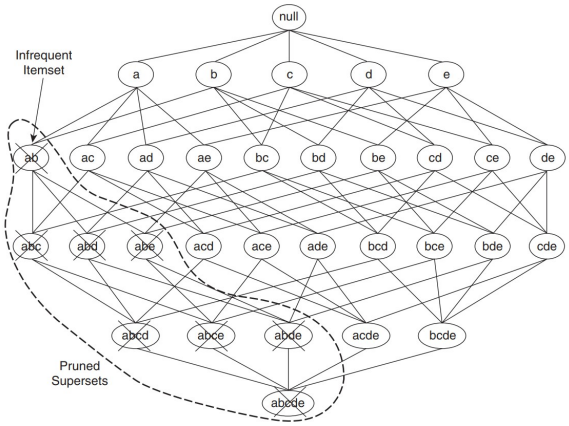
$$\forall X, Y : X \subseteq Y \rightarrow Support(X) \geq Support(Y) \tag{4}$$

Figure 7: Apriori and anti-monotone properties examples, from *"Introduction to data mining"* [20].

(a) *Apriori property.*



(b) Support-based pruning applying *anti-monotone property.*

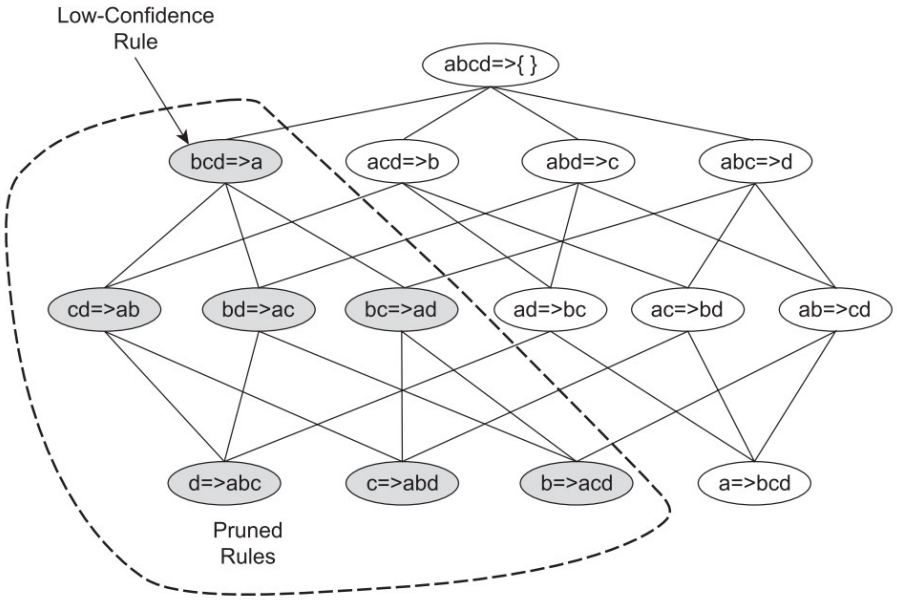


Rule generation in Apriori algorithm The Apriori algorithm is applied to extract association rules from such frequent itemsets. A **confidence-based pruning** approach is applied to do this. Contrary to the support measure, confidence does not satisfy the anti-monotone property, since the confidence for $X \rightarrow Y$ can be larger, smaller or equal to the confidence for another rule $\hat{X} \rightarrow \hat{Y}$, where $\hat{X} \subseteq X$ and $\hat{Y} \subseteq Y$. However, if the rules, whose confidences are being compared, are generated from the same frequent itemset, the following theorem is satisfied:

> *"Let Y be an itemset and X is a subset of Y. If a rule $X \rightarrow Y - X$ does not satisfy the confidence threshold, the any rule $\hat{X} \rightarrow Y - \hat{X}$, where $\hat{X}$ is a subset of X, must not satisfy the confidence threshold as well."*
>
> (*"Introduction to data mining"*)

At this point, if we take a lattice structure $I = \{a, b, c, d\}$ and calculate the confidence of each node, if any node is found to have a low confidence (determined by a threshold previously), according to the above theorem, the subgraph generated from said node can be pruned.

Figure 8: Example of pruning association rules using the confidence measure, from *"Introduction to data mining"* [20].



Limitations

- Choice of minimum support threshold: lowering support threshold results in more frequent itemsets. This may increase number of candidates and max length of frequent itemsets.
- Dimensionality (number of items) of the data set: more space is needed to store support count of each item. If number of frequent items also increases, both computation and I/O costs may also increase.
- Database size: since Apriori makes multiple passes, run time of algorithm may increase with number of transactions.
- Average transaction width: transaction width increases with denser data sets. This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width).
- Encoding scheme: not suitable for encoding the numerically-valued attributes as they may have different values in each record.

4.1.2. FP-Growth algorithm

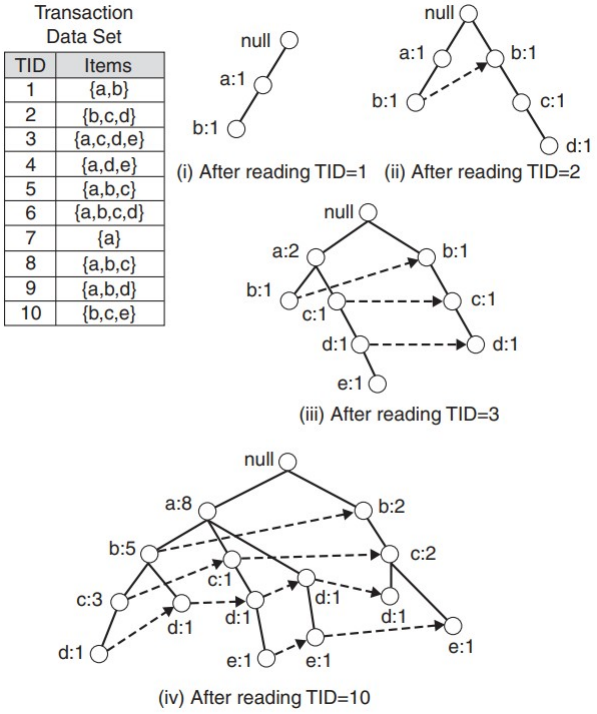
FP-Growth algorithm [25, 26] is based on a depth search procedure [27] that considered a prefix-tree-based compressed main memory representation of the input dataset.

FP-Growth corrects some of the deficiencies seen in section 4.1.1, such as the memory use (since it does not require generation of candidates) or the computational cost scanning the database. The algorithm take the advantage by using the ***FP-Tree*** [20] data structure, a much more compact version for saving data. There is no need to scan the database over and over again by using the *FP-Tree*. Instead, traversing it could do the same job more efficiently.

FP-Tree It is the core concept of the entire FP-Growth algorithm: it is the database's compressed representation of the itemset, i.e. a compressed representation of the input data. From it, it maintains and keeps track of the association between sets of elements.

The tree is built by taking each set of elements and mapping them to a path in the tree one at a time. The idea behind this structure is that items that occur more frequently will have a greater chance of sharing items.

Figure 9: Example of construction of an *FP-Tree*, from *"Introduction to data mining"* [20].



Throughout the algorithm, the problem to be faced is divided into different subproblems, i.e., the original tree is divided into smaller subtrees called **conditional *FP-***

Trees. From them, the algorithm makes a study of the suffixes that make up the different itemsets, so that when one of these suffixes has a support lower than the minimum support, it and the rest of the potential trees generated by the suffix itself will be considered not frequent immediately. This is the reason why FP-Growth is so efficient: the **divide and conquer approach** to mine the frequent itemsets.

Limitations

- When the database is huge and sparse, the *FP-Tree* will be large, increasing the complexity.
- The space requirement for recursion is truly a problem, given these huge datasets.

There are more proposals of FIM, which can be consulted in detail in [19].

4.2. Genetic programming and Grammatical evolution

A **GA** is a metaheuristic[12] inspired by the process of natural selection, belonging to the largest class of evolutionary algorithms. These types of algorithms are often used to generate high-quality solutions to search and optimization problems, relying on biologically inspired operators such as *selection*, *crossover*, or *mutation* [28].

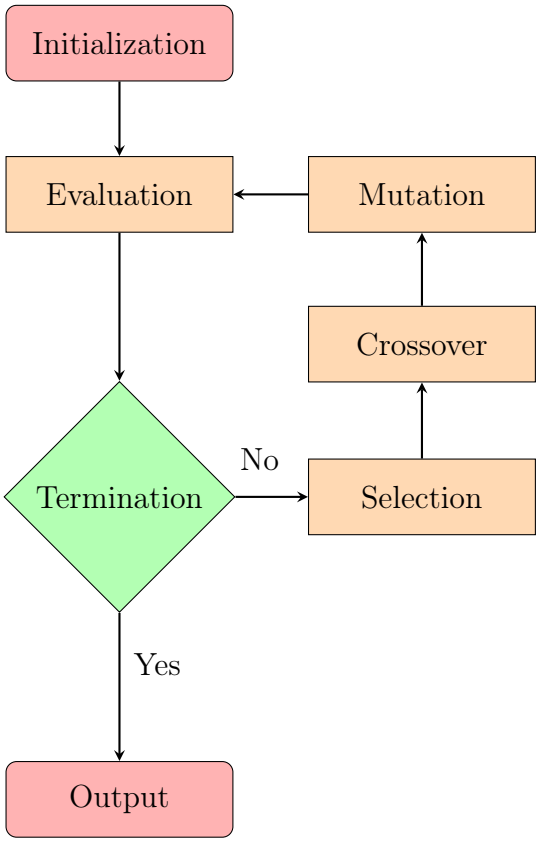
In general, members of each new generation are on average fitter than members of the previous generation. This recursive system terminates when a group of individuals reaches a predefined level of fitness[13] or proficiency, although it may happen that a particular run of the algorithm results in premature convergence to a local maximum that is not a globally optimal or even good solution. Usually several runs (tens to hundreds) are needed to produce a very good result.

GP is a GA, and the difference is that GP operates on computer programs, typically encoded as tree-structures while GA usually operates on coded strings. This implies that GP can be applied to a greater diversity of problems using more complex structures.

[12]In computer science and mathematical optimization, a metaheuristic is a heuristic procedure designed to find, generate, or select a heuristic (search algorithm partial) that can provide a good enough solution to an optimization problem, especially with incomplete or imperfect information, or limited computing power.

[13]The fitness measure gives an idea of how each individual performs in a particular problem environment. The nature of the measure varies with the problem.

Figure 10: Standard flowchart of GP.



Since the 1980s, much progress has been made in this regard, creating various subtypes of GP and taking them to different fields with numerous applications [29]:

- Biological and Genomic: DNA expression, gene annotacion, molecular structure optimization.
- Statistical and Numerical computing: quantum computing, search problems, solving complex optimization problems.
- Physical of materials: solid state physics, electronic and optical properties.
- Data mining and Machine learning: time prediction, classification problems, time series prediction.
- Financial modeling: predicting corporate bankruptcy, bond credit ratings, forecasting stocks indices.

GE [30] is an evolutionary computation technique, specifically, a GP technique, that works based on the grammar structure joining principles from molecular biology with the formal grammars. As in any other GP approach, the goal is to find an executable program, program fragment, or function that achieves a good fitness value for a given

objective function, by utilizing operations inspired by biological evolution. In general, genetic operators are applied to an entire string and then assigned to a program (or similar) using a grammar.

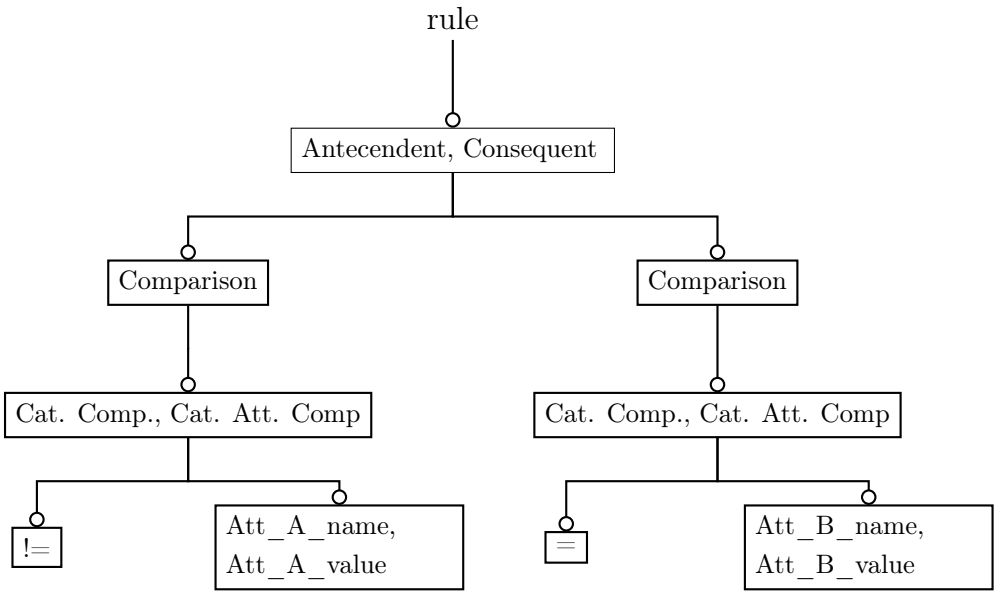
The problem with classical GP is that all functions had to be able to accept any output from any other function considered, so it used to work only with real numbers, leaving aside logical operators. By evolving following a grammar, you can constrain the solution space and introduce knowledge, and solve that problem.

The combination of these techniques produce as a result **grammatical genetic programming**, which has been used in several applications, such as the extraction of context-dependent association rules [31] or the induction of rule-based classifiers [32].

4.2.1. Evolutive approaches

G3PARM [33] is a proposal based on G3P[14], for mining association rules by using a grammar. Beyond using only an unique population of individuals (which in this case are represented by association rules), an auxiliary population of predefined size is presented in which a certain *support* and *confidence* is used as a threshold, in such a way that it maintains the best individuals. The used grammar to define production rules can be modified to make use of datasets of any data type, either numerical or categorical.

Figure 11: G3PARM: Example of genotype of an individual represented in a syntax-tree structure, from *"Design and behavior study of a grammar-guided genetic programming algorithm for mining association rules"* [33].

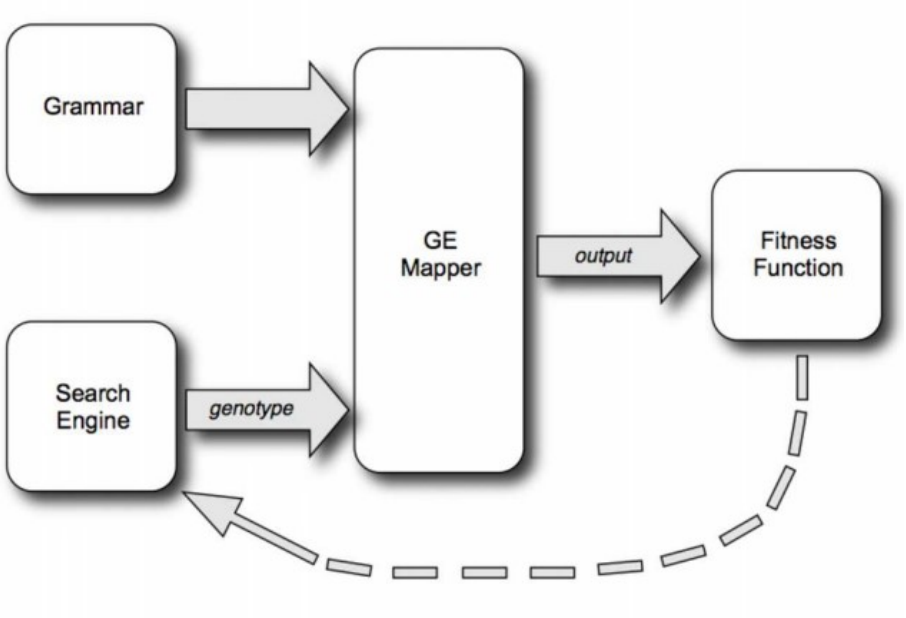


[14]G3P is a GP extension where the individuals are represented as derivation trees, that is a solution using a grammar.

G3PARM is capable of presenting solutions without the need of using large amounts of memory in a given time.

GEVA [34, 35][15] is an open source software implementation of GE in Java which provides a search engine framework in addition to a simple GUI and the genotype-phenotype mapper [16] of GE, which follows figure 12 structure:

Figure 12: Modular components of GE, from *"GEVA: Grammatical Evolution in Java"* [34].



This allowed to develop a framework where any search engine algorithm could be used to generate the genotypes that are used to direct the GE mapper's use of the grammar during the output generation.

Such strategy gave as a result alternative search engines as Grammatical Swarm and Grammatical Differential Evolution, variants of the original Particle Swarm and Differential Evolution algorithms, respectively [34].

4.3. Associative classification

Associative classification (AC) [37] is a special case of classification that uses association rules, in which only the class attribute is considered in the rule's right-hand side, i.e, the consequent:

[15]The reader is recommended to access the references to see the complete operation of GEVA, exposed with various examples accompanied by an explanation that facilitates its understanding.

[16]The mapping process works in conjunction with a grammar, to transform an ordinary string of integers into a possible solution to a problem. [36]

$$X \rightarrow Y \tag{5}$$

This means that its consequent provides a hint that a tuple satisfying its antecedent belongs to a specific class. AC takes the advantage over classic classification approaches by providing the user a simpler output in if-then rules, which eases its interpretation; and by being capable of updating a rule without affecting the complete rules set, whereas the same task requires some updates like reshaping the whole tree in the decision tree approach.

It is important to note that an AC task is different from association rule discovery:

- Association rules discovery aims to discover relations between datasets items with no class attribute involved, and the generated rules allow multiple attribute values in the rules consequent.
- AC aims to construct a classifier that can predict the classes of test data objects using single attribute consequents, the class attribute, within its rules.

Some outstandings AC approches are CBA2, CPAR or FARCHD.

4.3.1. Associative approaches

CBA2 (Classification Based Association) [38] is an improved version of the first AC proposal, CBA [39]. Even though it had a good performance in general, it was not suitable for unbalanced problems. Moreover, some interesting rules (with many conditions) that increase the accuracy of the classification may not be generated, due to the large number of combinations.

CBA2 tackled these problems by using multiple class minimum supports in rules generation, i.e., each class has a different minimum support, rather than using just one in the hole process; and a combination of decision trees with Naïve-Bayes method.

The idea is to segment the data by generating a set of associations between attribute-value pairs and the class attribute using Apriori, and then selecting the lowest error rate classifier on each segmentation to classify future data.

CPAR (Classification based on Predictive Association Rules) [40] stands in the middle between exhaustive and greedy algorithms, combining the advantages of both for iteratively generating effective rules for patterns not covered by previous rules. Subsequently, classification involves the evaluation of the k best rules, for each class, that cover the incoming pattern.

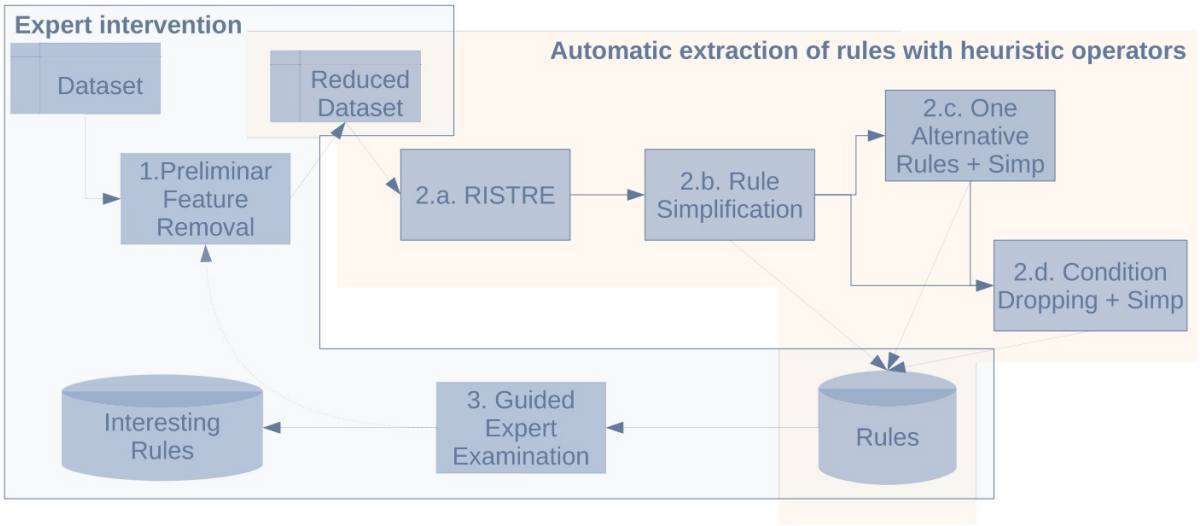
CPAR generates smaller rule sets with higher quality and lower redundancy in comparison with AC approaches like CBA and, as a result, is much more time-efficient in both rule generation and prediction, achieving as high accuracy as AC.

FARCHD (Fuzzy Association Rule-based Classification for High-Dimensional problems) [41] is a method to obtain an accurate, compact and efficient fuzzy rule-based classifier with a low computational cost. It firstly extract association rules by employing a depth-limited tree, which facilitates the interpretation of rules in linguistic terms and avoids unnatural boundaries in the partitioning of the attribute domains. Then, the most interesting rules are preselected in order to build a classifier by finally applying a GA which selects and tune a compact set of fuzzy association rules with high classification accuracy.

4.4. Heuristics for class association rule mining

It consists of the **combination of four operators** [42] for obtaining potentially interesting conjunctive class association rules from databases, following the figure 13 workflow:

Figure 13: Workflow to obtain interesting class association rules, from *"Heuristics for interesting class association rule mining a colorectal cancer database"* [42].



1. RISTRE (Recursive induction of supervised trees and subsequent rule extraction): allows the generation of many combinations of influential features by not only removing the feature at the tree's root and repeating the tree induction, but also inducing trees when other features at different depths are removed.

2. Rules simplification: this operator is combined with the classification and regression trees (CART) [43] methodology. After applying it, it is checked if the suppression of conditions or alternatives simplifies the understanding of the rule, that is, shortens it, without affecting its quality. This occurs as long as the confidence does not worsen or, in the case of categorical attributes, if no positive covered pattern presents that alternative, thus maintaining the confidence of the rule.

3. One categorical alternative rules generation: generate rules that consider just one alternative for their categorical conditions, due to its interest having higher interpretation and potential utility.

4. Condition dropping: the rules produced by operators 1 and 2 contain some conditions whose removal produces another rule that may be interesting. This operator revises all the conditions of the rule, evaluating the quality after the condition removal.

This method is capable of finding interesting rules among trivial associations and, what is more, it has the advantage, over standard class association rule mining methods, of avoiding getting lost among the large amount of possibilities.

5. Restrictions

In this section, all those existing restrictions in the field of design and that condition the choice of one or another alternative will be exposed. They are determined by the following factors:

- **Project restrictions:** are those inherent in the nature of the main problem and that cannot be modified, e.g., existing technology or temporary limitations.
- **Strategy restrictions:** design variables in which you will have to choose between several possibilities. The final solution depends on these choices. For instance, the use or not of graphics, what type of operating system and programming language are more suitable, etc.

5.1. Project restrictions

- Dataset size: medium size - 1516 x 87.
- Human restrictions: the project has been carried out individually by the student[17], with the support of the assistant professors[18].
- Temporal restrictions: the total time cost for the value of the credits (12) must not exceed 300 hours.
- Hardware restrictions: the author's personal equipment has been used. The full features can be seen in section 6.2.2.
- Python [44], as a programming language, due to Academic Tutors wanted the resulting product to be in Python.

5.2. Strategy restrictions

- Ubuntu 20.04, as main operative system, due to:
  - The prior familiarization with the OS allows the management and movement between directories to be more fluid and efficient.
  - Various distributions based on Linux, such as Ubuntu, come pre-installed with various programming tools, for example, some data analysis libraries. Other systems, such as Windows, require the installation of this type of dependencies.

[17] See section 6.1.1, *Author*.
[18] See section 6.1.2, *Assistant Professors*.

- Visual Studio Code [45] is a code editor that offers support for many programming languages and also provides numerous extensions and tools to improve and facilitate the experience, such as interactions with GitHub. Its interface allows easy access to project directories as well as to an integrated terminal and debugging tools, allowing it to be usable and efficient. Pycharm [46] was considered because it is designed specifically for Python and also has some of the previous commented tools, like GitHub. However, Visual Studio Code was selected, due to the previously familiarization with it.
- The following Python libraries:
  - *Matplotlib*: library for graph generation.
  - *NumPy*: library that supports creating large multidimensional arrays, along with a large collection of high-level mathematical functions to operate with.
  - *Pandas*: *NumPy* extension for data manipulation and analysis that provides data structures and operations to manipulate numerical tables.
  - *Scikit-Learn*: library for machine learning that includes various classification, regression, and cluster analysis algorithms.
  - *Scipy*: as an additional library of mathematical tools and algorithms.
- Git [47], as a version control software and GitHub [48], as a platform to host repositories that use Git. They facilitate the administration of the project and the collaboration between its participants.
- Overleaf [49], as a LateX text composition system, oriented to the creation of documents such as scientific articles and books, with a high typographic level. It facilitates the insertion of mathematical elements.

6. Resources

The following resources have been made available:

6.1. Human resources

6.1.1. Author

Table 3: Author data.

Name	Ventura Lucena Martínez
Email	i72lumav@uco.es
Degree	Computer Science
Mention	Computing
Scholar year	4th

6.1.2. Assistant Professors

Table 4: Academic Tutor 1 data.

Name	Carlos García-Martínez
Email	cgarcia@uco.es
Position	Associate Professor of the Department of Computer Science and Numerical Analysis

Table 5: Academic Tutor 2 data.

Name	José María Luna Ariza
Email	jmluna@uco.es
Position	Interim Substitute Professor of the Department of Computer Science and Numerical Analysis

6.2. Material resources

6.2.1. Software

- Ubuntu 20.04, as main operative system.
- Visual Studio Code, as a programming environment.

- Python, as a programming language.
- PonyGE2, as a grammatical evolution Python software.
- *Matplotlib*, *NumPy*, *Scipy*, *Scikit-Learn* and *Pandas*, as tools for the application of machine learning techniques.
- Git as version control software.
- GitHub, to host Git repositories in the cloud.
- Overleaf, as a LateX text composition system for documentation.

6.2.2. Hardware

- Processor (CPU): AMD A10-7700K APU with Radeon (TM) R7 Graphics (4CPUs), $\sim$ 3.4GHz.
- Graphics Processing Unit (GPU): AMD Radeon R9 200 Series.
- 8GB RAM.
- 2TB solid state storage drive.

7. System Analysis

7.1. Software requirements specification (SRS)

This section describes the set of requirements that the system must satisfy. These are divided into two groups: **user requirements** and **system requirements**.

7.1.1. User requirements (UR)

URs are the expectations that a user or customer has of the system to meet their needs. The following are identified:

UR-1. As a user, I want to have a system that is capable of reading a dataset.

UR-2. As a user, I want to be able to extract class association rules from a dataset by means of a genetic programming algorithm.

UR-3. As a user, I want to filter the extracted association rules based on different criteria:

- Extract only rules with consequents equal to a provided value.
- Extract only rules whose *confidence* is greater than a provided value.

UR-4. As a user, I want to extract the following metrics[19] from:

- Each rule: *confidence*, *recall*, *lift*, *leverage*, *conviction*, antecedent's *support*, consequent's *support*, rule's *support* and covered patterns.
- Each set of rules: minimum and average *support*, minimum and average *confidence*, number of rules extracted, number of conditions within the antecedent, attributes used, positive patterns not covered.

UR-5. As a user, I want to be able to see a chart of the fitness of the evolutionary process.

UR-6. As a user, I want to save the information in a readable way in a file for easy access and management.

7.1.2. System requirements (SR)

SRs are the services or functionalities that the system must provide, so that all URs are covered. To do this, they will be divided into the following types: **information**, **functional**, **non-functional** and **interface**.

[19] See section 7.2.2, *Metrics*.

Information requirements (IFR) IFRs describe the information that must be stored and managed by the system to support the processes. For proper operation:

IFR-1. The system must be able to read input data in a ".arff" file format[20].

IFR-2. The system must read the algorithm parameters from a file in ".txt" file format, as it is predefined in the original PonyGE2 software [21].

IFR-3. The grammar definition must be stored in a ".bnf" file format[22].

IFR-4. The system must extract the association rules and their associated metrics in a ".csv" file format[23].

Functional requirements (FR) FRs are those that indicate how the functionalities of the system should be in order to comply with the URs. The following are identified:

FR-1. The system must be able to read the files in the established formats:

- Dataset: ".csv" or ".arff".
- Parameters: ".txt".
- Grammar: ".bnf".

FR-2. The system must be configured by using a parameters plain ".txt" file for later execution.

FR-3. The system must process the data in such a way that it is capable of generating antecedents and consequents from the input data set through a grammar.

FR-4. The system must extract a set of metrics of the generated association rules.

Non-functional requirements (NFR) NFRs refer to all requirements that describe performance characteristics, such as usability, accesibility, efficiency, failure tolerance, security, among others. The non-functional system requirements are:

NFR-1. The algorithm must meet the functional requirements with the minimum number of failures, that is, it must present functionality and guarantee.

NFR-2. The algorithm has to be accessible to experts in the field of data science. To do this, all functionality must be well documented.

NFR-3. The algorithm must be constructed in such a way that it presents the highest possible maintainability.

NFR-4. The algorithm must be reusable.

[20]See annex A.1, *ARFF format.*
[21]See annex A.4, *TXT format for PonyGE2 parametes.*
[22]See annex A.2, *BNF format.*
[23]See annex A.3, *CSV format.*

Interface requirements (ITR) ITRs are those that are related in some way to how the system interface should be. The system interface requirements are:

ITR-1. The application interface should be usable, minimalist and intuitive, using the command-line interface (CLI), just like the default version of PonyGE2.

7.2. Analysis

7.2.1. Use case

UC-1. Rule extraction.

UC-1. UC Diagram - Association rule extraction.

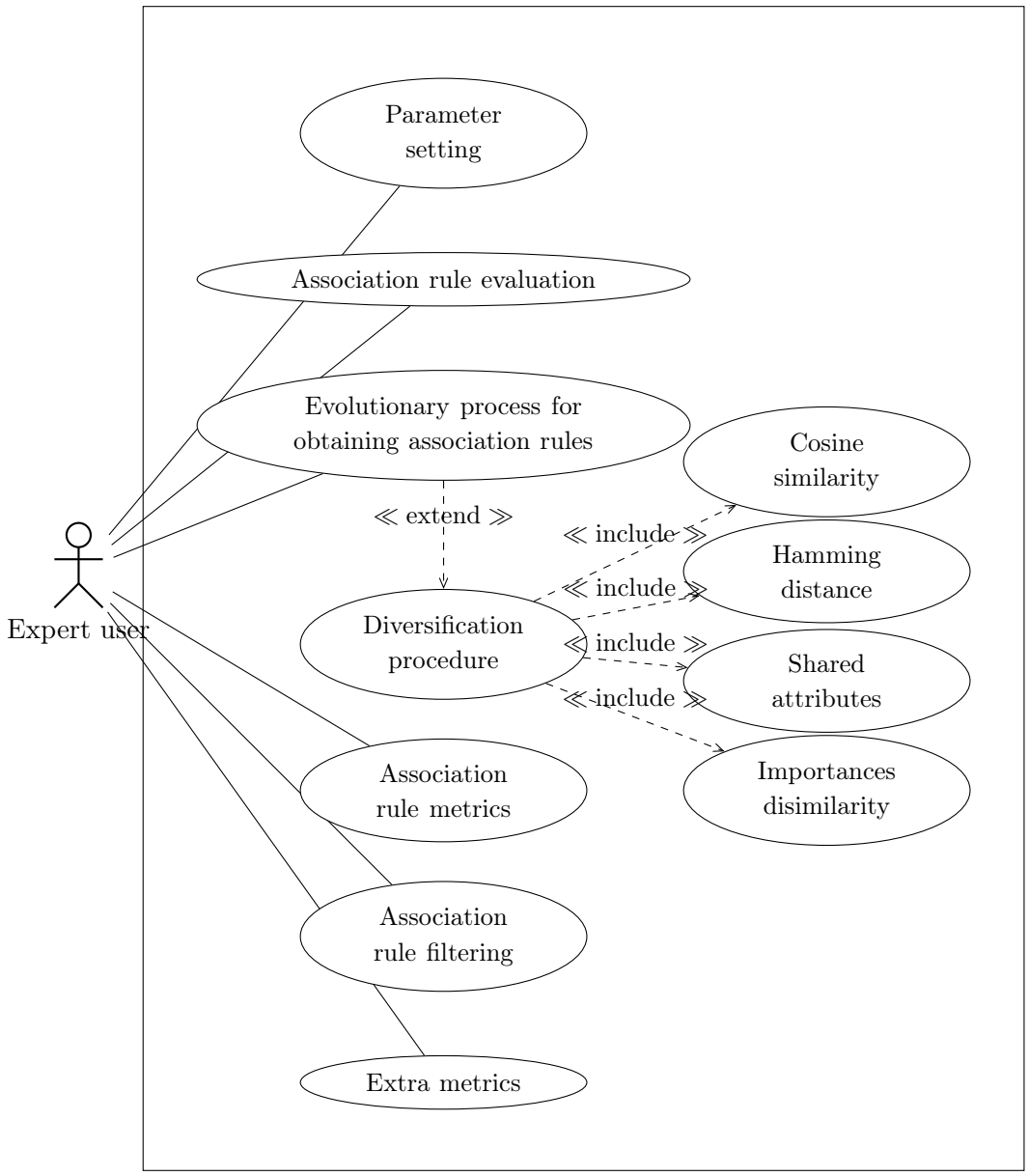


Table 6: UC-1.1: Parameter settings - Association rule extraction.

Use case	Parameter settings.
Description	Configuration of parameters of the PonyGE2 grammar evolution algorithm.
Actors	Expert user.
Preconditions	Have a dataset on which to extract association rules.
Main flow	1) Execution of the algorithm without setting parameters. 2) Use of default parameters.
Alternative flow	a) Custom parameter settings. i. Set custom parameters in the configuration file. b) Custom parameters settings failed. i. Set custom parameters in the configuration file. ii. Getting error message.
Postconditions	a) Parameter assignment is successful. b) Parameters resetting.

Table 7: UC-1.2: Association rules evaluation - Association rule extraction.

Use case	Association rules evaluation.
Description	Performs the specific processing required to extract association rules with the extension of PonyGE2.
Actors	Expert user.
Preconditions	1) Have a dataset on which to extract association rules. 2) Have a grammar.
Main flow	1) Extraction of antecedents and consequents from a nested condition list, based on the solutions of the evolutionary algorithm. 2) Extraction of labels of the obtained antecedents, analyzing it within the dataset. 3) Get the antecedent support. 4) Extraction of probabilities according to each label. 5) Compute fitness function based on the weighted Gini Impurity index.
Alternative flow	-
Postconditions	Get a set of antecedents and consequents, and its fitness value.

Table 8: UC-1.3: Evolutionary process for association rules obtaining - Association rule extraction.

Use case	Evolutionary process.
Description	Compute an evolutionary algorithm to obtain association rules.
Actors	Expert user.
Preconditions	Have performed the data processing for the association rules.
Main flow	1) Generate a population of individuals with a fitness value. 2) Save the best individuals and discard the worst, according to fitness value. 3) Save the average fitness of the population. 4) Repeat the process iteratively.
Alternative flow	-
Postconditions	Obtaining a final population of individuals, i.e., a set of association rules, evaluated with weighted Gini Impurity index.

Table 9: UC-1.4: Diversification procedure - Association rule extraction.

Use case	Diversification procedure.
Description	In order for the method to offer various association rules, several diversification techniques have been introduced in the algorithm.
Actors	-
Preconditions	1) Set "SHARING_FITNESS" parameter to "True". 2) Select one among the existing options: - Hamming distance. - Cosine similarity. - Shared attributes. - Importances disimilarity. 3) Have obtained a population of individuals.
Main flow	1) Generate a population of individuals. 2) Diversify the obtained population. 3) Continue with the evolutionary process.
Alternative flow	Do not compute the diversification process ("SHARING_FITNESS" parameter to "False".)
Postconditions	Obtain a diversified population.

Table 10: UC-1.5: Association rules metrics - Association rule extraction.

Use case	Association rules metrics.
Description	Extract a set of interesting metrics of the obtained association rules.
Actors	Expert user.
Preconditions	Get a set of association rules.
Main flow	Compute metrics: 1) Support (Antecedent, consequent and entire rule). 2) Precision. 3) Recall. 4) Lift. 5) Leverage. 6) Conviction. 7) Covered targets.
Alternative flow	-
Postconditions	Extract a set of metrics associated to each rule.

Table 11: UC-1.6: Association rules filtering - Association rule extraction.

Use case	Association rules filtering.
Description	Filter the obtained association rules under a certain criterion associated to its metrics.
Actors	Expert user.
Preconditions	Extract a set of metrics associated to each rule.
Main flow	Execution of the filtering with default thresholds.
Alternative flow	a) Custom parameter settings. i. Set custom criterion within the source code. b) Custom criterion settings failed. i. Set custom criterion within the source code. ii. Getting error message.
Postconditions	Obtaining a set of filtered association rules.

Table 12: UC-1.7: Extra metrics - Association rule extraction.

Use case	Extra metrics.
Description	Extract a set of extra interesting metrics of the obtained association rules.
Actors	Expert user.
Preconditions	1) Get a set of association rules. 2) Filter a set of association rules.
Main flow	Compute metrics: 1) Support (Minimum and average support). 2) Confidence (minimum and average confidence). 3) Number of rules. 4) Number of antecedent conditions. 5) Used attributes in rules. 6) Uncovered positive patterns.
Alternative flow	-
Postconditions	Extract a set of extra metrics associated to the set of association rules.

7.2.2. Algorithms and Techniques

It is necessary to mention that two ways for obtaining class association rules from GE methods are considered: either rules can be evolved, or trees can be evolved to obtain their decision rules. This section covers the algorithms and techniques that will be used for the proposal: firstly, an introduction to the GE programming library will be seen; secondly, the metrics that will be used to evaluate both the rules and the rule sets, later, the fitness function for the generation of classification trees and, lastly, strategies for the diversification of the algorithm.

PonyGE2: Grammatical Evolution in Python PonyGE [50] is a small but functional implementation of GE in Python. Due to the unique implementation in a single source file, the disorganization of the code and the requirements of new users, it was decided to abandon the original idea in order to correct what was originally "pony-sized" and had been unmanageable turn. Thus, PonyGE was restructured in a modular way, so that the code was adapted to allow the use of multiple search engines and operators: PoneGE2.

These changes have endowed the algorithm with the ability to combine representation types: both, genome and tree representations are implemented simultaneously. In addition, it allows synergies to be created between them by being able to mix operators of any type of representation.

Since PonyGE2 is an approach that makes use of GE, a grammar (in ".bnf" text file

extension) must be defined prior to its use. It can include both a complete language or a subset of a language that is suitable for the problem to be solved. Inasmuch as the objective is to extract association rules, whose structure is `IF-THEN` rules (i.e., `IF` some conditions are satisfied, `THEN` some others, too), we can pay attention to the example 7.2.2 provided with PonyGE2, which offers the possibility of inducing classification trees for labeled datasets with the `if_else_classifier` grammar:

Listing 1: Grammar example in PonyGE2 for inducing classification trees.

```
<cf> ::= np.where(<cond>, <cf>, <cf>) | <os>
<cond> ::= (<var> == <is>) | <cond> & (<var> == <is>)
<is> ::= GE_RANGE:dataset_n_is
<os> ::= GE_RANGE:dataset_n_os
<var> ::= x[:, <varidx>]
<varidx> ::= GE_RANGE:dataset_n_vars
```

Two parts are distinguished: the terminal symbols located on the left side of `::=` and the rule composed of terminal and/or non-terminal symbols, located at the right side.

Notice that, apart from the `GE_RANGE` tags, the language is Python code that can be evaluated with the *Numpy* package, as long as `x` stores the training dataset as a *Numpy* `ndarray` object.

The `GE_RANGE` tags refer to properties of the associated `if_else_classifier` fitness function which were expected to be passed as additional parameters. They are replaced by numerical sets with the possibilities and indexes of the input features, and the possible target labels, in Python code. The traditional way would be to write `<n> ::= 0 | 1 | 2 | 3 | ... |`, while `GE_RANGE` tag creates dynamic ranges of numbers automatically [51]. This is used for the number of input symbols, the number of output symbols and the dataset columns, denoted by the `<is>`, `<os>` and `<varidx>` tags, respectively.

The fitness function can access to the attributes by using the tag `varidx` within the training dataset. Usually, it would require each variable to be explicitly mentioned in the grammar, for instance `<vars> ::= x[0] | x[1] | x[2] | ... |`; but a new tag `var` can be created in such a way that stores the dataset values indicated by the previous tag `varidx` and the Python syntax [52].

The `<cf>` tags refer to non-terminal root symbols that represent any individual of the algorithm, i.e., trees: `np.where(x[0] == 2, 3, np.where(x[1]==1 and x[2] == 1, 5, 9))`.

Conditions are generated with the structure of the non-terminal symbol `<cond>`, for instance, `x[0] == 2` on the one side, and `x[1] == 1 and x[2] == 1` on the other.

On the other hand, observing the previous grammar we can find some deficiencies:

1. It cannot effectively manage categorical features with different number of possibili-

ties, because the range `<is>` is the same for all the features.

2. Solution interpretability gets reduced, due to features names and values are encoded with integers.

3. It cannot effectively deal with numerical features nor deal with common operators that are required by them, such as relational operators as $\geq, \leq, >, <, ...$

These problems cannot be solved with potentially better but general grammars, due to feature names and feature possibilities have to be correctly paired. To address theses issues, the following smart operators had been introduced [1]:

Dataset-based production rules To deal with drawbacks 1 and 2, it is necessary to change the initial grammar in such a way that it does not generate invalid conditions that generate superfluous solutions, wasting time on it. Some examples like `x['SEX'] == 'Primary cancer'` or `x['TUMOR_SIZE'] < 'Appendix'` can be generated, resulting uninteresting-meaningless solutions. Thus, it would be necessary for the conditions to have a correct condition-value pairing, so that each feature has its comparison with its possible range of values. This had been addressed with the following changes in PonyGE2:

- ***Pandas* DataFrames** [53] adding: improves interpretability by allowing string labels as both feature names and feature values.
- **The ability of automatically extending the grammar** with dataset-based production rules for appropriate conditions. **Simple cases** and **elaborate cases** will be differentiated:

Table 13: Dataset-based production rules non-terminal symbols.

Simple dataset-based production rules	Elaborated dataset-based production rules
`<GE_GENERATE:dataset_eq_conditions>`	`<GE_GENERATE:dataset_inset_conditions>`
`<GE_GENERATE:dataset_neq_conditions>`	`<GE_GENERATE:dataset_notin_conditions>`
`<GE_GENERATE:dataset_lessequal_conditions>`	
`<GE_GENERATE:dataset_greater_conditions>`	

- **Simple dataset-based production rules:** conditions of equality and inequality (`==` and `!=`) for categorical features, and the less-than-or-equal-to and greater than ($<=$ and $>$) for numerical features. When one of the symbols of table 13 is read by the algorithm, it reads the dataset and adds new production rules to the grammar. Notice that with the new production rules, each feature is associated to its particular set of values.

Listing 2: New production rules for equality conditions, from *"Smart Operators for Inducing Colorectal Cancer Classification Trees with PonyGE2 Grammatical Evolution Python Package"* [1].

```
<GE_GENERATE:dataset_eq_conditions> ::= x[/feature_1/] ==
   <value_feature_1> | x[/feature_2/] == <value_feature_2>
    | ...
<value_feature_1> ::= /value_1_feature_1/ | /
   value_2_feature_1/ | ...
<value_feature_2> ::= /value_1_feature_2/ | ...
...
```

- **Elaborated dataset-based production rules:** conditions checking that a feature takes or does not take values from a set of possibilities.
  - The "in set" and "not in" non-terminal symbols are for categorical features, that generate set production rules. This way, the rules can consider different alternatives, for the same feature, to be satisfied.

Listing 3: New production rules for "not in" conditions, from *"Smart Operators for Inducing Colorectal Cancer Classification Trees with PonyGE2 Grammatical Evolution Python Package"* [1].

```
<GE_GENERATE: dataset_notin_conditions> ::= (~np.isin(x[/
   feature_1/], <subset_values_feature_1>)) | ... (~np.
   isin(x[/feature_2/], <subset_values_feature_2>)) | ...

# For each categorical feature, subsets with cardinality
   in [2,8], by default,  because of the exponential
   nature of the powerset function.
<subset_values_feature_1> ::= (/value_1_feature_1/, /
   value_2_feature_1/) | ...
<subset_values_feature_2> ::= (/value_1_feature_2/, /
   value_2_feature_2/) | ...
```

Listing 4: Grammar for classification trees: *if_else_classifier_heterogeneous_data.bnf*

```
<cf> ::= np.where(<cond>, <cf>, <cf>) | <GE_GENERATE:
   dataset_target_labels>
<cond> ::= <GE_GENERATE:dataset_eq_conditions> |
           <GE_GENERATE:dataset_neq_conditions> |
           <GE_GENERATE:dataset_lessequal_conditions> |
           <GE_GENERATE:dataset_inset_conditions> |
           <GE_GENERATE:dataset_notin_conditions> |
           <GE_GENERATE:dataset_greater_conditions> |
           <cond> & <cond>
```

Pruning and Correcting Operators In EA, most of the operators apply stochastic decisions. Applying this kind of operators is very inefficient due to the algorithm produces and evaluates constantly trees with unimportant nodes. The following cases are distinguished:

- Unreachable nodes: intermediate solutions probably contain subtrees that are not used in the classification process, because no dataset pattern fulfills the conditions. This would involve checking additional conditions that the algorithm could produce that would be meaningless.
    - A *pruning* operator had been implemented, replacing the parent of the unreachable node by a leaf with the most likely class of the patterns that reach the parent node.
- Same decision: some subtrees may produce leaf nodes or decision nodes with the same prediction on all their branches.
    - The previous *pruning* operator had been also used to solve this problem.
- Poor splits: some internal nodes may be distinguishing the reaching patterns not very effectively, improving the quality measures of the whole tree by an insignificant amount, probably overtraining the tree, improving the classification of training patterns but deteriorating that of testing patterns.
    - The reduction of the Gini impurity of the set of patterns reaching the node was evaluated. If this impurity reduction was less than a threshold, then the subtree was replaced by a leaf with the most common class.
- Wrong decision: the final prediction is the wrong one according to the training set.
    - A fix had been implemented that changed the decision node.

Metrics The metrics serve to measure, in a quantifiable way, the performance, in this case of rules and rule sets, with respect to a reference system.

- **Main metrics** The following metrics serve to measure the quality of individual rules.
    - **Support:** defined for itemsets, not association rules. Used to measure the abundance or frequency (often interpreted as significance or importance) of an itemset in a dataset. In the case of association rules, the *support* is usually calculated as the percentage of patterns that meet a condition, either in the antecedent or the consequent.
        - **Antecedent support:** computes the proportion of transactions or patterns that contain A.
        $$support(A) = P(A) \tag{6}$$
        Range: [0, 1]
        - **Consequent support:** computes the proportion of transactions or patterns that contain C.
        $$support(C) = P(C) \tag{7}$$
        Range: [0, 1]

By convention, the *support* of a rule is usually calculated as the percentage of patterns that satisfy both the antecedent and the consequent of the rule:

$$support(A \rightarrow C) = support(A \cup C) = P(A \cap C) \tag{8}$$

Range: [0, 1]

- **Confidence:** probability of seeing C in a transaction given that it also contains A. The metric is not bidirectional.

$$confidence(A \rightarrow C) = \frac{support(A \rightarrow C)}{support(A)} = \frac{P(A \cap C)}{P(A)} = P(C \mid A) \tag{9}$$

Range: [0, 1] (1 implies that A and C always occur together)

- **Recall:** number of positives recovered, i.e., ratio of all positive patterns that satisfy the antecedent.

$$Recall(A \rightarrow C) = \frac{P(A \cap C)}{P(C)} \tag{10}$$

Range: [0, 1]

- **Lift:** is the rise in probability of having C with the knowledge of A being present, over the probability of having C without any knowledge about presence of A. In cases where A actually leads to C, the value of *lift* will be greater than 1, i.e, it vouches for high association between A and C; whereas a value less than 1 shows that having A does not increase the chances of occurrence of C in spite of the rule showing a high confidence value. If A and C are independent, the *lift* score will be 1.

$$lift(A \rightarrow C) = \frac{confidence(A \rightarrow C)}{support(C)} = \frac{P(C \mid A)}{P(C)} = \frac{P(A \cap C)}{P(A)P(C)} \tag{11}$$

Range: [0, ∞] (1 means independence)

- **Leverage:** computes the difference between the observed frequency A and C appearing together and the frequency that would be expected if A and C were independent. A *leverage* value of 0 indicates independence.

$$\begin{aligned} leverage(A \rightarrow C) = support(A \rightarrow C) - support(A) \times support(C) = \\ = P(A \cap C) - P(A)P(C) \end{aligned} \tag{12}$$

Range: [-1, 1]

- **Conviction:** compares the probability that A appears without C if they were dependent on the actual frequency of the appearance of A without C.

$$conviction(A \rightarrow C) = \frac{1 - support(C)}{1 - confidence(A \rightarrow C)} = \frac{P(A)P(\overline{C})}{P(A \cap \overline{C})} \tag{13}$$

Range: [0, ∞] (1 indicates independence, rules that always hold have ∞)

- **Extra metrics** Apart from the quality metrics for rules, the following metrics will be used to measure rule sets quality.

  - **Support:**
    - Minimum: minimum *support* in the entire rule set.
    - Average: average *support* in the entire rule set.
  - **Confidence:**
    - Minimum: minimum *confidence* in the entire rule set.
    - Average: average *confidence* in the entire rule set.
  - **Number of rules:** interesting-extracted rules.
  - **Number of antecedent conditions:** indicates the number of rules that have c_i conditions within the antecedent. The structure is as follows, being $c \equiv$ conditions and $r \equiv$ rules:

$$c = [1, 2, 3, 4, 5, 6, 7, 8] \quad r = [0, 1, 1, 2, 3, 5, 8, 13] \tag{14}$$

  - **Used attributes in rules:** amount of used and different attributes in the entire rule set.
  - **% of attribute use frequency:** by computing a subset of metrics.
    - Max: appearance frequency of the most used feature.
    - Q3: third quartile.
    - Median.
    - Q2: second quartile.
    - Min: appearance frequency of the least used feature.

    The structure is as follows:

$$\{max, q_3, median, q_2, min\} \tag{15}$$

  - **% of uncovered positive patterns:** percentage of patterns that satisfy the class label but are not covered by a rule.

Gini impurity index The Gini impurity index is a method used in decision tree algorithms to decide the optimal split from a root node, and subsequent splits, i.e. tells us what is the probability of misclassifying an observation. Note that the lower the Gini, the better the split. Its **weighted-formal definition** is the following (for a branch, i.e., a rule):

$$Gini = \left(1 - \sum_{i}^{n\_labels} p_i^2\right) \cdot w \qquad [0, 0.5] \tag{16}$$

Where:

p_i is the probability of appearance of the label i.

w is the weight, computed as the antecedent *support* of the analyzed rule.

For an entire tree:

$$Fitness = \sum_{i=0}^{n\_rules} Gini_i = \sum_{i=0}^{n\_rules} \left[\left(1 - \sum_{j}^{n\_labels} p_j^2 \right) \cdot w_i \right] \qquad [0, 0.5] \qquad (17)$$

Where:

$Gini_i$ is the Gini impurity of the rule i.

p_j is the probability of appearance of the label j.

w_i is the weight of the rule i, computed as the antecedent *support* of such rule.

This metric will be used as the fitness function of each tree.

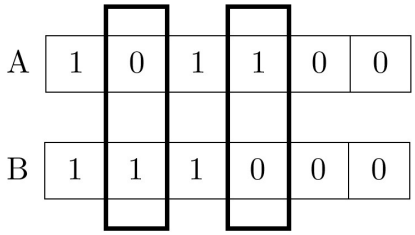
Diversification Diversity is the variation or differences in the genetic material among individuals or candidate solutions in the EA population. In general, diversity promotes exploration of the solution space to locate a single good solution (or multiples when there is more than one solution) by delaying convergence [54]. If there is not enough diversity, then the individuals are too similar one another, and if it drops very quickly, the population converges to not very good solutions.

In addition to the *crossover* and *mutation* operators, commonly used in EA, a *crowding* technique will be used, as well as measurements of distance or similarity between individuals' attributes (depending on the user's choice).

Crowding *Crowding* is a method that prevents premature convergence. To determine the similarity, a measure of distance or similarity is used, in conjunction with binary arrays, that represent the attributes used by individuals. In this project, the *crowding* method aims to penalize solutions that are similar to better ones by using this kind of measures, in order to choose individuals as different as possible.

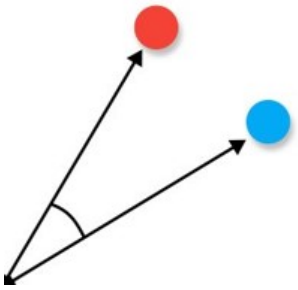
Hamming distance The Hamming distance between two strings of equal length is the number of positions in which the corresponding symbols are different. In other words, it measures the minimum number of substitutions needed to change one chain for another.

Figure 14: Hamming distance representation, from *"9 Distance Measures in Data Science"* [55].



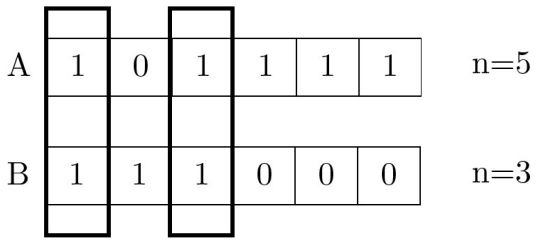
Cosine similarity Cosine similarity [24] measures the similarity between two vectors of an inner product space. Measured by the cosine of the angle between two vectors, it determines whether two vectors are pointing in roughly the same direction. Its value is equal to 1 if the angle is 0, that is, if both vectors point to the same place. If the vectors were orthogonal, the cosine would cancel, and if they pointed in the opposite direction, their value would be -1. In this way, the value of this metric is between -1 and 1, that is, in the closed interval [-1, 1]. It is particularly used in positive space, where the outcome is neatly bounded in [0, 1].

Figure 15: Cosine similarity representation, from *"9 Distance Measures in Data Science"* [55].



Similarity based on the number of shared attributes This custom metric measures the similarity between two individuals by counting the number of attributes they share, divided by the least number of attributes in use between both individuals (so that they are similar even when one inserts more attributes that may not serve any purpose)..

Figure 16: New similarity approach representation.



$$Similarity = \frac{Commons}{n_{min}} = \frac{2}{3} \tag{18}$$

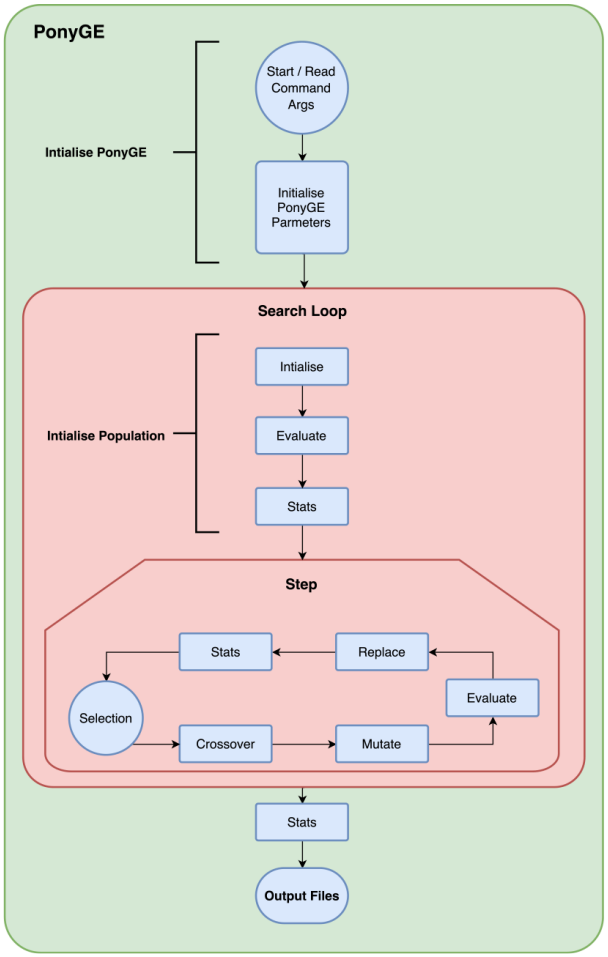
8. System Design and Implementation

Previously, the PonyGE2 algorithm has been briefly seen, what type of grammars serve as basis, what deficiencies the grammar that comes as an example has and some solution proposals.

Faced with the dilemma of choosing to evolve rules or trees, **the decision has finally been made to evolve trees**, because it is the alternative that allows using the evaluation function based on Gini impurity. Hence, in this section we will show how the previous contents have been used to implement a system that generates trees based on the Gini impurity, which contain association rules as branches, ending with the implementation of the metrics for the evaluation of the rules.

The flow chart that PonyGE2 follows is presented to make it easier for the reader to locate each step, which will be described in the next section:

Figure 17: PonyGE2 control flow diagram, from *"PonyGE2: Grammatical Evolution in Python"* [17].



Once PonyGE2 has read the grammar, it will start a process called `search_loop`, a standard search process for an EA that loops over a given number of generations. The

population of individuals is generated in the `initialise` procedure within a Python list with a series of associated attributes, being interesting in this section:

- `fitness`: represents the value of an individual. `Float` type.
- `invalid`[24]: in some cases it is possible that an incomplete mapping could occur, generating an individual with still non-terminal symbols on it. Such an individual is set invalid as it will never undergo a complete mapping to a set of terminals, nor therefore, will generate a potential solution. `Bool` type.
- `phenotype`: the actual physical representation of the chromosome. `String` type. An example of phenotype would be as follows:

Listing 5: PonyGE2 phenotype example.

```
phenotype = "np.where((~np.isin(x['EXERESIS_MESORRECTO'], ('No
    Aplicable', 'No Realizada', 'Total'))), 'Si', 'No')"
```

Then, an entire population of individuals gets evaluated in the `evaluate` procedure. **An individual is evaluated by its phenotype using a fitness function.** In this case, **two different parts stand out** in the created fitness function:

1. **Extraction:** the phenotype of an individual cannot be directly evaluated. This is because the character string that makes it up has a nested structure of the form `np.where(<condition>, np.where(<condition>, ...))` (may internally contain nested conditions splitted by `&`). It is necessary to divide the character string and process it, finally identifying antecedents and consequents. Thus, the antecedents and consequents obtained from the previous example are:

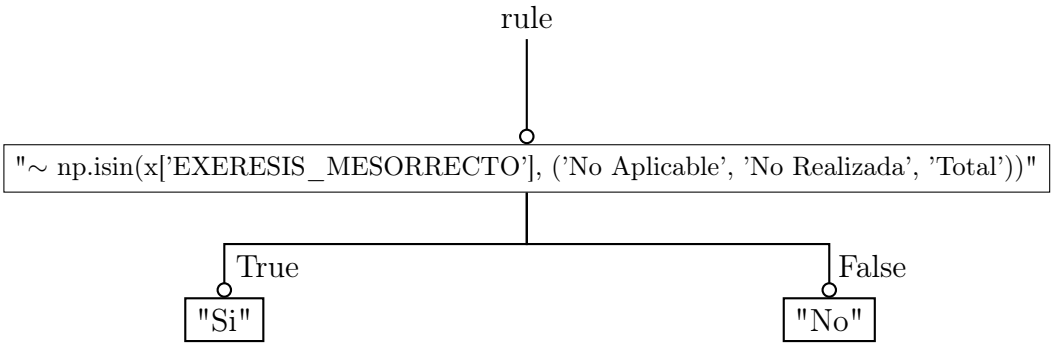
Listing 6: Antecedents and consequents extraction example.

```
antecedents=["(~np.isin(x['EXERESIS_MESORRECTO'], ('No
   Aplicable', 'No Realizada', 'Total')))", "~((~np.isin(x['
   EXERESIS_MESORRECTO'], ('No Aplicable', 'No Realizada', '
   Total'))))"]

consequents=['Si', 'No']
```

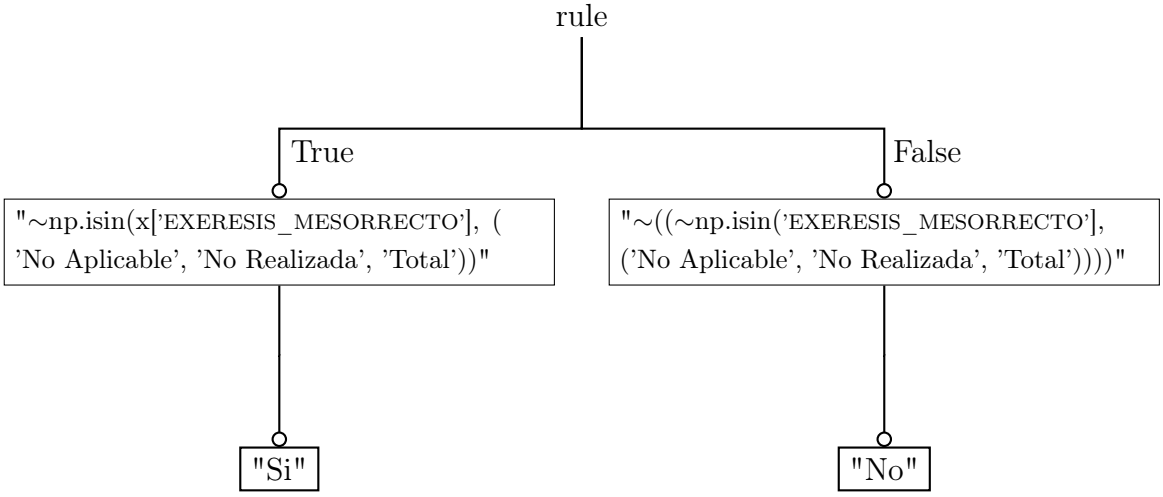
[24]See more information in section 4.3 of [17]

Figure 18: Association rules tree representation (1).



Notice that rule = "False" ≡ negation of the rule itself. Hence:

Figure 19: Association rules tree representation (2).

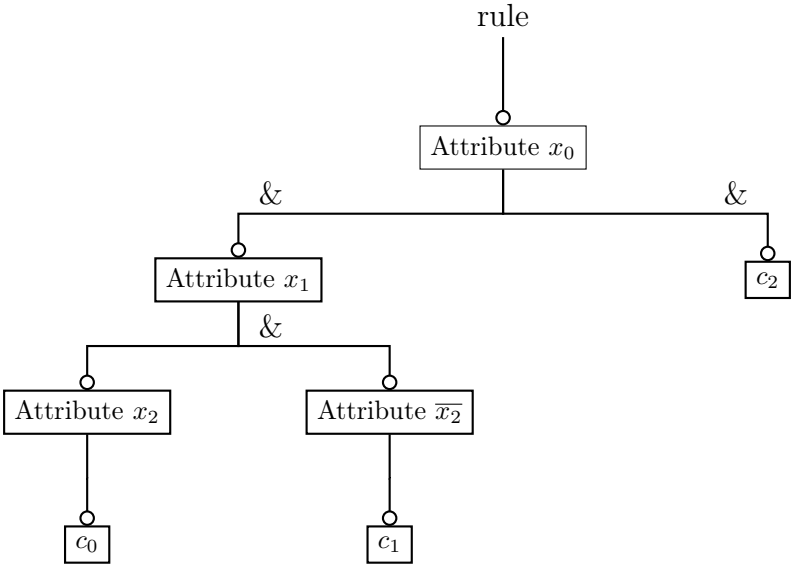


As expected, there will be as many antecedents as consequents. Note, that both are Python lists, where each component is paired with the one from the other list, this is:

$$Antecedent_i \rightarrow Consequent_i \tag{19}$$

Finally, the more nested conditions, the more tree levels:

Figure 20: Association rules tree representation (3).



Rule 1: ((x[x_0] == value) & (np.isin(x[x_1], (value))) & (x[x_2] == value)) $\rightarrow c_0$

Rule 2: ((x[x_0] == value) & (np.isin(x[x_1], (value))) & ($\sim$x[x_2] == value)) $\rightarrow c_1$

Rule 3: (x[x_0] == value) $\rightarrow c_2$

Note that the equality operator `==` and the `np.isin()` *Numpy* method has been used to expose different possibilities, whereas `value` indicates the possible values of the x_i attribute.

2. **Evaluation:** it consists of calculating the fitness of a tree, that is, of the set of extracted rules in the previous step. The support of each antecedent is calculated by **evaluating** it **within the dataset** (that is, obtaining how many records satisfy the antecedent condition with respect to all those considered), the probabilities of each label `['Si']-['No']` are obtained with respect to the target; and the **weighted Gini impurity** is computed with the support of each antecedent.

Algorithm 1 Rules fitness evaluation.

Require: $antecedents\_length = consequents\_length$
1: $weighted\_gini\_list \leftarrow list()$
2: **for** i from 0 to $antecedents\_length$ **do**
3: $labels \leftarrow getLabels(rule[i])$
4: $antecedent\_support \leftarrow getMetrics(antecedents[i])$
5: $probabilities \leftarrow getLabelsProbs(labels)$
6: $weighted\_gini\_list \leftarrow getWeightedGiniImpurity(probabilities)$
7: **end for**

Finally, the fitness associated to the tree will be equivalent to the sum of the set of weighted Ginis[25] calculated:

$$Fitness = \sum_{i=0}^{n\_rules} \left[\left(1 - \sum_{j}^{n\_labels} p_j^2 \right) \cdot w_i \right] \tag{20}$$

Those **trees that contain one or several invalid phenotypes will directly have `np.nan` fitness**, i.e., null fitness, in such a way that they perish evolutionary process.

8.1. Metrics

Once the evolutionary process (`step` in figure 17) is finished, the measures are extracted. For **each extracted association rule**, the metrics of section 7.2.2 (main metrics) have been applied. To do this it is necessary to know which records satisfy the antecedent, to have the set of targets and a consequent:

- Records that satisfy the antecedent: they can be obtained in a simple way, by passing an antecedent to the `eval`[26] function. Using the example listing 4 above:

Listing 7: Records that satisfy an antecedent example.

```
eval("(~np.isin(x['EXERESIS_MESORRECTO'], ('No Aplicable', 'No
    Realizada', 'Total')))")

Output:
[True True True ... True False False]
```

- Set of targets: the target in this problem is called "COMPLICACIONES", allocated at the last column of the dataset. It is represented by `y`:

Listing 8: Targets example.

```
y

Output:
['Si' 'No' 'No' ... 'Si' 'Si' 'Si']
```

- A consequent: obtained previously while the extraction procedure. Using the example listing 4 above:

Listing 9: Consequent example.

```
consequent

Output:
'Si'
```

[25] See equation 16, *Weighted Gini Impurity index*.

[26] `eval` is a Python native function that interprets a string passed as parameter as a Python code.

Once the evolutionary process has concluded, all the extracted data is exported in its corresponding ".csv" file using *Pandas* DataFrames:

- "trees.csv": includes an index asociated to each tree, each phenotype and its fitness value.
- "metrics.csv": includes the phenotype of rule's tree, the rules and its metrics.

8.2. Updating

In the first instance, once the set of association rules and their metrics had been obtained, it may happen that there are rules whose confidence is less than 0.5. That is why a functionality had been implemented that builds the opposite association rule by updating its consequent, in such a way that its confidence is also opposite and, therefore, greater than 0.5. Suppose the previous example:

Listing 10: Rule updating example.

```
rule = "(~np.isin(x['EXERESIS_MESORRECTO'], ('No Aplicable', 'No
   Realizada', 'Total'))) --> 'Si'"

if rule.find("--> Si") != -1:
    rule = rule.replace("--> Si", "--> No")
elif rule.find("--> No") != -1:
    rule = rule.replace("--> No", "--> Si")

print(rule)

Output:
"(~np.isin(x['EXERESIS_MESORRECTO'], ('No Aplicable', 'No Realizada
   ', 'Total'))) --> 'No'"
```

Afterwards, the *Wrong decision* smart operator, seen in section 7.2.2, was imported replacing this functionality that had been useful initially.

8.3. Filtering

Since the data has been saved in *Pandas* DataFrames in the previous steps, filtering data by user criteria is easy:

Listing 11: Discard association rules with `consecuent = No`.

```
df[df["Rule"].str.contains("--> No") == False]
```

Listing 12: Discard association rules with `confidence < 0.7`.

```
df.drop(df[df["Confidence"] < 0.7].index)
```

Specifically, what these filters indicate is that it is important to maintain those rules that show complications in CRC cases and that the probability that this transaction occurs is greater or equal than 0.7.

8.4. Extra metrics

For **each extracted association rule**, the metrics of section 7.2.2 (extra metrics) have been applied. It has been again a straightforward approach thanks to *Pandas* DataFrames.

8.5. Diversification

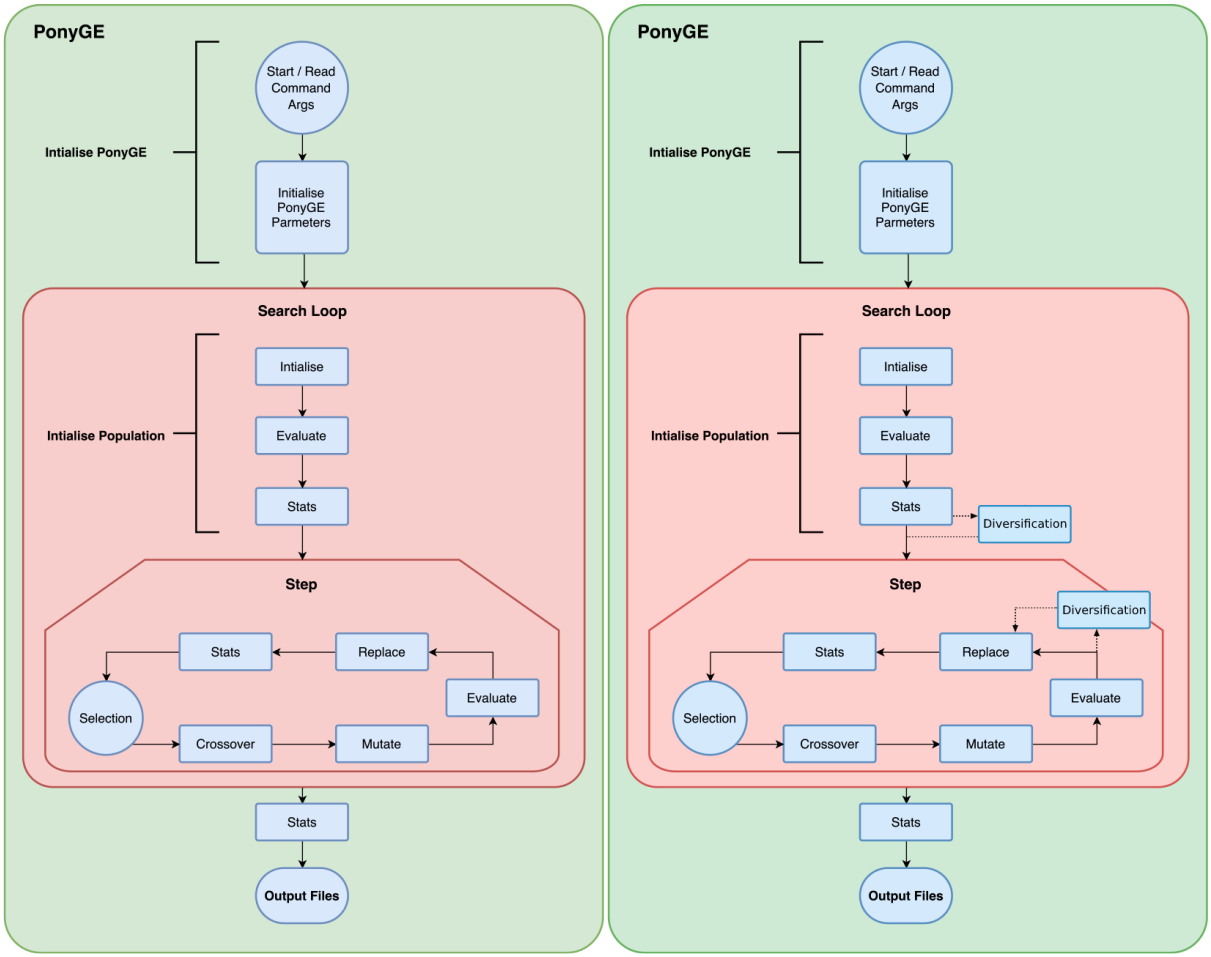
On the one hand, in order to make this procedure easy to execute, a parameter called `SHARING_FITNESS` of type `bool` has been added to PonyGE2's parameters configuration file. A value of `True` applies the procedure.

On the other hand, this procedure slightly alters the control flow of the algorithm: to apply diversity to the population of individuals through new non-native methods, the diversification tag has been added to the figure 17 diagram:

Figure 21: PonyGE2 control flow diagram comparison.

(a) Original control flow diagram.

(b) New control flow diagram.



Algorithm 2 Diversification procedure.

Require: *individuals*
Ensure: *individuals* ▷ Diversificated individuals.
1: $n\_features \leftarrow getNDatasetFeatures()$
2: $descendingSort(individuals)$
3: $n\_trees \leftarrow length(individuals)$
4: $features \leftarrow getUsedFeatures(individuals, n\_features)$
5: **if** $params[\text{“}SHARING\_PROCEDURE\text{”}] = \text{“}distance\text{”}$ **then**
6: $A, A\_flat = computeDistances(features, n\_trees)$ ▷ A = distance matrix.
7: **else**
8: $A, A\_flat = computeSimilarity(features, n\_trees)$ ▷ A = similarity matrix.
9: **end if**
10: $crowding = computeCrowding(A, A\_flat, n\_trees, params[\text{“}SHARING\_PROCEDURE\text{”}])$
11: $updateFitness(individuals, crowding, n\_trees)$
return *individuals*

To carry it out, firstly it has been determined what features each individual in the

population uses by extracting the indexes of the pattern features used in the phenotype. Remembering the structure of a phenotype:

Listing 13: Representation of individuals used features.

```
"np.where((~np.isin(x['EXERESIS_MESORRECTO'], ('No Aplicable', 'No
   Realizada', 'Total'))), 'Si', 'No')"
```

which is actually (thanks to *Pandas* DataFrames support[27]):

Listing 14: Representation of individuals used features.

```
"np.where((~np.isin(x.iloc[:,22], ('No Aplicable', 'No Realizada',
   'Total'))), 'Si', 'No')"
```

it can be seen that the indexes within the `ndarray` represents the used features. By analysing each string, extracting the indexes and storing the used features in a binary array, we can group the whole information into another array associated to each indexed individual with the following structure:

Listing 15: Representation of individuals used features.

```
[
    [individual_0, [feature_0, feature_1, ..., feature_k-1]]
    [individual_1, [feature_0, feature_1, ..., feature_k-1]]
    [individual_2, [feature_0, feature_1, ..., feature_k-1]]
    ...
    [individual_n-1, [feature_0, feature_1, ..., feature_k-1]]
]
```

For instance, the equation 21 represents that individual 8 uses the features 2, 3 and 5 (starting from 0 in both cases).

$$[7, [0, 1, 1, 0, 1, 0]] \tag{21}$$

Now, a *crowding* method is executed.

Crowding As described in section 7.2.2, *crowding* is a procedure to avoid premature convergence, in this case, by penalizing individuals that are similar to better ones. Three different approaches have been implemented, in which the second part of the previous structure is used, that is, the binary array of used attributes. Depending on the method, a distance[28] or similarity[29] matrix is built to compare pairs of individuals' used features from the features array:

[27] See section 7.2.2, *Dataset-based production rules*.

[28] See section 7.2.2, *Hamming distance*.

[29] See section 7.2.2 and 7.2.2, *Cosine similarity* and *Similarity based on the number of shared attributes*, respectively.

Table 14: Similarity matrix example.

	i_0	i_1	i_2	...	i_{n-2}	i_{n-1}
i_0	1	-	-	-	-	-
i_1	$d_{0,1}$	1	-	-	-	-
i_2	$d_{0,2}$	$d_{1,2}$	1	-	-	-
...	$d_{0,...}$	$d_{1,...}$	$d_{2,...}$	1	-	-
i_{n-2}	$d_{0,n-2}$	$d_{1,n-2}$	$d_{2,n-2}$	$d_{...,n-2}$	1	-
i_{n-1}	$d_{0,n-1}$	$d_{1,n-1}$	$d_{2,n-1}$	$d_{...,n-1}$	$d_{n-2,n-1}$	1

Table 15: Distance matrix example.

	i_0	i_1	i_2	...	i_{n-2}	i_{n-1}
i_0	0	-	-	-	-	-
i_1	$d_{0,1}$	0	-	-	-	-
i_2	$d_{0,2}$	$d_{1,2}$	0	-	-	-
...	$d_{0,...}$	$d_{1,...}$	$d_{2,...}$	0	-	-
i_{n-2}	$d_{0,n-2}$	$d_{1,n-2}$	$d_{2,n-2}$	$d_{...,n-2}$	0	-
i_{n-1}	$d_{0,n-1}$	$d_{1,n-1}$	$d_{2,n-1}$	$d_{...,n-1}$	$d_{n-2,n-1}$	0

Note that both measures are in range [0,1] and antagonistic:

- A **similarity equals to 1** and a **distance equals to 0** determines that the **elements to be compared have no differences**.
- A **similarity equals to 0** and a **distance equals to 1** determines that the **elements to be compared are totally opposed**.

A sorted-flattened array is also build up with the values of the matrix, in order to extract the percentile 0.25 if the method is based on distance, or 0.75 if the method is based on similarity. This value will serve as a threshold to mark individuals to be penalized.

Algorithm 3 *Crowding* procedure.

Require: $A, A\_flat, n\_trees, procedure$
Ensure: *crowding*

```
if procedure = "distance" then
    p_25 ← percentile_25(A_flat)
    crowding ← zerosList(n_trees)
    for i from n_trees to 0 do
        for j from i to 0 do
            if A[i][j] <= p_25 then
                crowding[i] ← +1
            end if
        end for
    end for
end if
if procedure = "similarity" then
    p_75 ← percentile_75(A_flat)
    crowding ← zerosList(n_trees)
    for i from n_trees to 0 do
        for j from i to 0 do
            if A[i][j] >= p_75 then
                crowding[i] ← +1
            end if
        end for
    end for
end if
return crowding
```

The idea here is to penalize individuals who are worse than similar or slightly distant to others. Therefore, what is done is to go through the population from the worst individual to the best, counting how many of the other individuals it has in its vicinity.

Finally, the penalty[30] is applied by updating each individual with the calculated rate-reduction:

$$New\_Fitness_i = Crowding_i \cdot Fitness_i + Fitness_i \tag{22}$$

8.6. Alternative diversification approach

Under the same fundamentals as the previous diversification process, an alternative mechanism has been implemented with the aim of measuring the "importance" of the attributes of one individual with respect to another. Previously, if an attribute has already appeared in better solutions, the individual gets penalized. Now the number of rules in which each attribute appears is also being counted (the attribute at the root appears in all the rules) and taken in account in order to penalize an individual.

This is because, as will be seen later in section 9, the algorithm generates numerous trees (solutions) whose attributes located at the root[31] node are the same.

[30] The fitness value is used as penalty so that, if the value of crowding is 0 the fitness is not updated.
[31] Attributes located at the root node are those that appear at the beginning of the first `np.where`.

Algorithm 4 Alternative diversification approach.

Require: $individuals$
Ensure: $individuals$ ▷ Diversificated individuals.

$descendingSort(individuals)$
$n\_trees \leftarrow length(individuals)$
if $params[\text{“}SHARING\_PROCEDURE\text{''}] = \text{“}importance\_disimilarity\text{''}$ **then**
    $importances = getFeatureImportancesFromInds(individuals)$
    $A, A\_flat = ratioNewRules(importances)$
    $crowding = computeCrowding(A, A\_flat, n\_trees, procedure = \text{”}distance\text{”})$
end if
$updateFitness(individuals, crowding, n\_trees)$

return $individuals$

9. Testing

This section presents the tests accomplished on the implementation of the PonyGE2 extension using the CRC dataset. The main objectives are both to identify that the outputs obtained are consistent with the expected outputs, and to determine the correct operation of the algorithm that satisfies the requirements identified in section 7, in other words, a **black-box testing**. In the case of **white-box tests**, certain code fragments were checked during development to ensure that they worked correctly, such as antecedents and consequents extraction, metrics when evaluating rules, and the fitness function in the case of trees.

Note that due to the numerous implementations and their different possible configurations, each test will be accompanied by a table indicating the settings used and some graphs showing the quality of the best individuals and the average over the population.

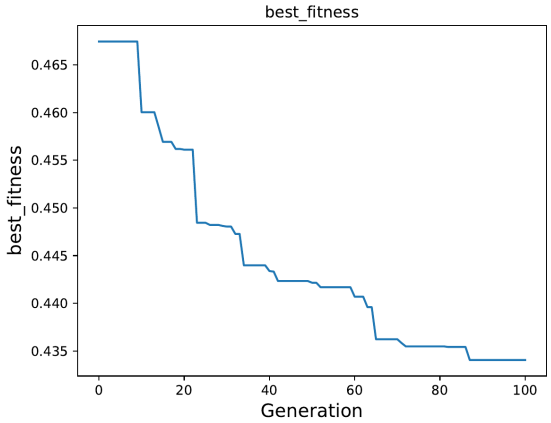
9.1. Test 1

Table 16: Test 1 parameters.

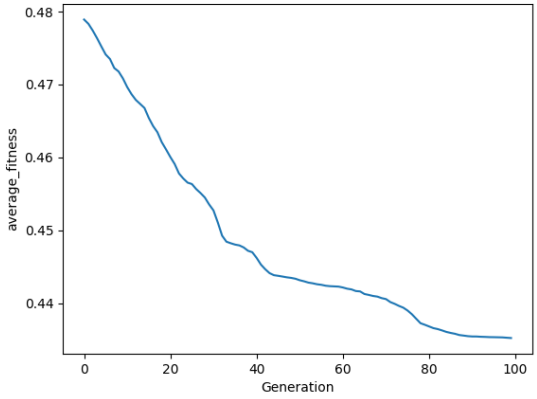
Parameter	**Value**
Crossover	Variable one-point
Crossover probability	0.75
Elite size	50
Fitness function	Weighted Gini Impurity
Generations	100
Initialization	Position Independent Grow
Maximum initial tree depth	10
Maximum tree depth	17
Mutation	Integer flip per codon
Mutation probability	None ($\frac{1}{genome\_length}$)
Population size	200
Replacement	Generational
Selection	Tournament
Tournament size	2

Figure 22: Test 1 without *crowding*.

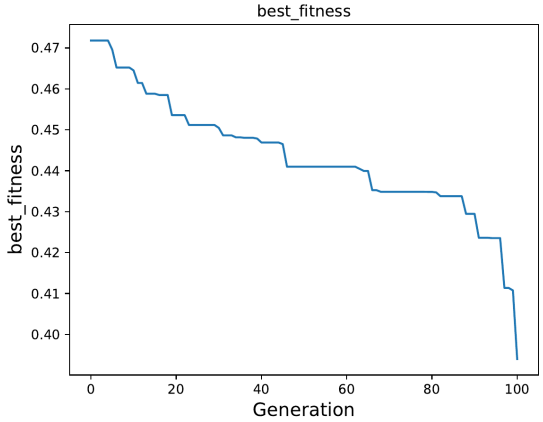
(a) Best fitness chart - Execution 1.



(b) Elite average fitness chart - Execution 1.



(c) Best fitness chart - Execution 2.



(d) Elite average fitness chart - Execution 2.

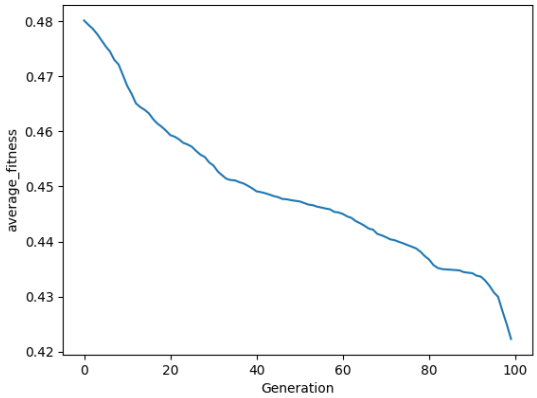
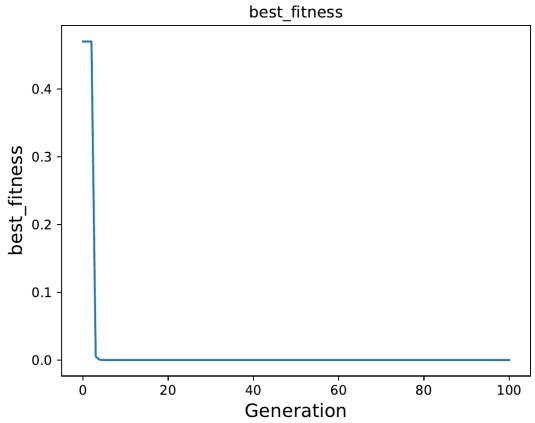
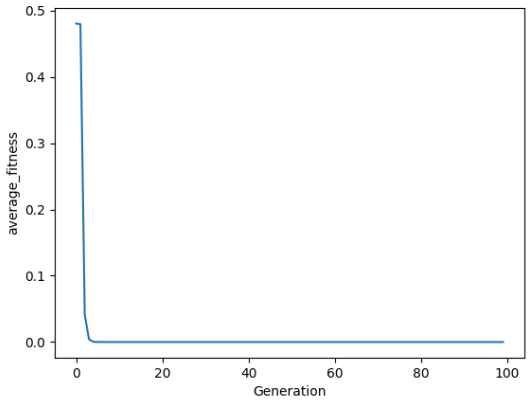


Figure 23: Test 1 with *crowding*: Cosine similarity.

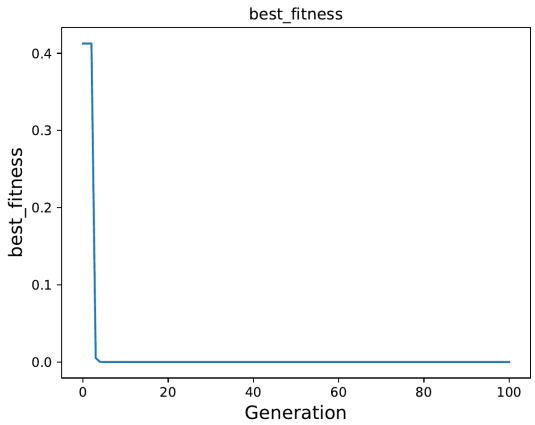
(a) Best fitness chart - Execution 1.



(b) Elite average fitness chart - Execution 1.



(c) Best fitness chart - Execution 2.



(d) Elite average fitness chart - Execution 2.

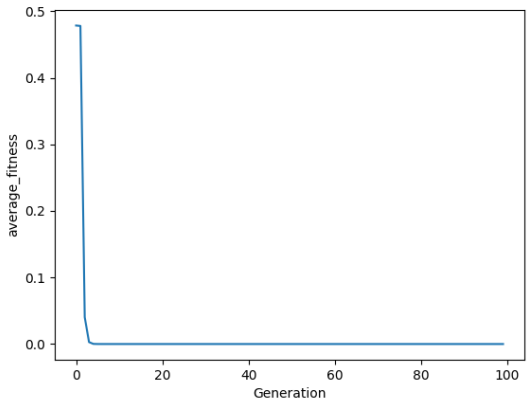


Table 17: Test 1 results.

Exec.	Crowding	Trees	Rules	Avg. fitness	Avg. precision
1	None	50	820	0.435166571	0.45136433
1	Cosine similarity	50	186	9.64519E-68	0.462388269
2	None	50	480	0.420828092	0.516951942
2	Cosine similarity	50	161	7.77581E-90	0.487623607

Conclusions With just two executions, it was seen that very similar individuals were generated in the runs without *crowding*, while in the runs with *crowding* the fitness function converged too quickly. It was found that the penalty[32] function had been implemented in the wrong way, specifically, in the reverse way to how it should work, rewarding similar

[32]Initially, the penalty function was designed as equation 23, which was finally corrected and changed by equation 22.

individuals instead of penalizing them. This was because the Gini impurity is a function whose best value is 0 and worst 0.5:

$$New\_Fitness_i = \frac{Fitness_i}{Crowding_i} \tag{23}$$

For both cases, the size of the population was reduced from 200 to 100 and the size of the elite from 50 to 5. Regarding to *crowding*, the penalty function was corrected, so that if an individual was very similar to better one the value of his fitness was increased.

9.2. Test 2

After the previous considerations, both the population and elite sizes were reduced. Also, a bug was found in the code that incorrectly generated the similarity matrix: instead of looking like the example in table 14, it was built like table 18. The code was corrected and executed.

Table 18: Wrongly-constructed similarity matrix.

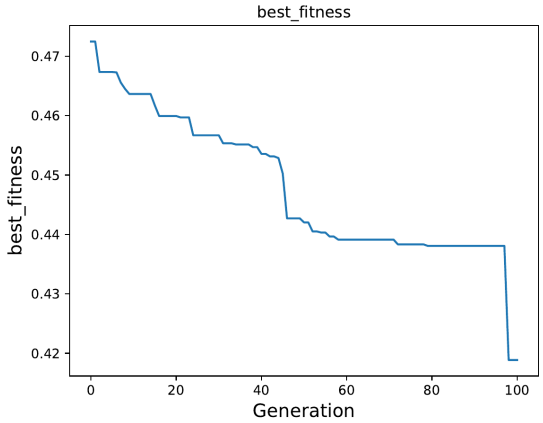
	i_0	i_1	i_2	...	i_{n-2}	i_{n-1}
i_0	1	$d_{0,1}$	$d_{0,2}$	$d_{0,...}$	$d_{0,n-2}$	$d_{0,n-1}$
i_1	$d_{0,1}$	1	$d_{1,2}$	$d_{1,...}$	$d_{1,n-2}$	-
i_2	$d_{0,2}$	$d_{1,2}$	1	-	-	-
...	$d_{0,...}$	$d_{1,...}$	$d_{2,...}$	-	-	-
i_{n-2}	$d_{0,n-2}$	$d_{1,n-2}$	-	-	-	-
i_{n-1}	$d_{0,n-1}$	-	-	-	-	-

Table 19: Test 2 parameters.

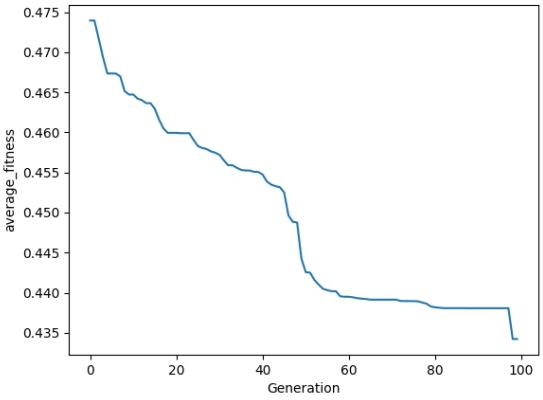
Parameter	**Value**
Crossover	Variable one-point
Crossover probability	0.75
Elite size	5
Fitness function	Weighted Gini Index
Generations	100
Initialization	Position Independent Grow
Maximum initial tree depth	10
Maximum tree depth	17
Mutation	Integer flip per codon
Mutation probability	None ($\frac{1}{genome\_length}$)
Population size	100
Replacement	Generational
Selection	Tournament
Tournament size	2

Figure 24: Test 2 without *crowding*.

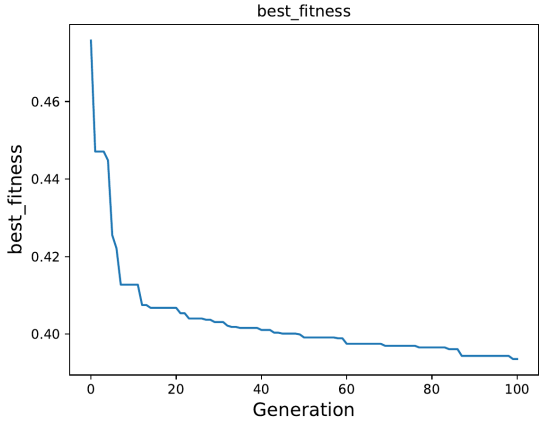
(a) Best fitness chart - Execution 1.



(b) Elite average fitness chart - Execution 1.



(c) Best fitness chart - Execution 2.



(d) Elite average fitness chart - Execution 2.

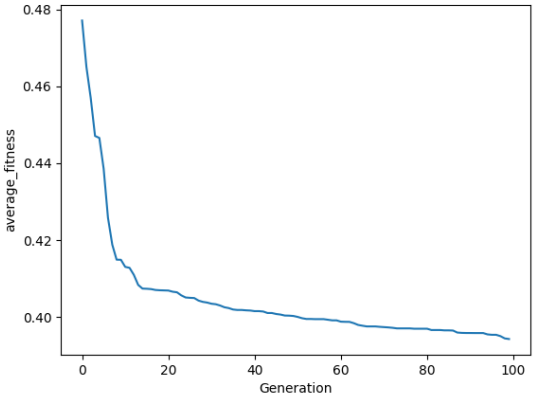
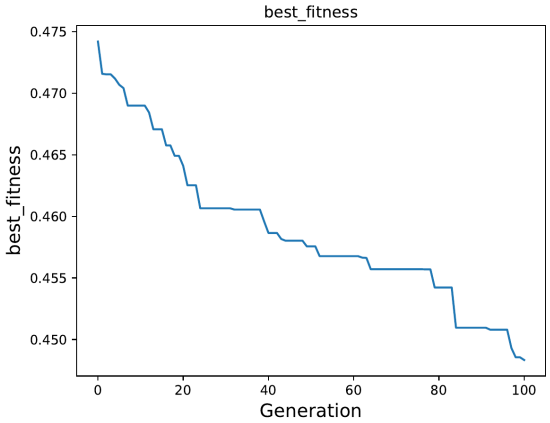
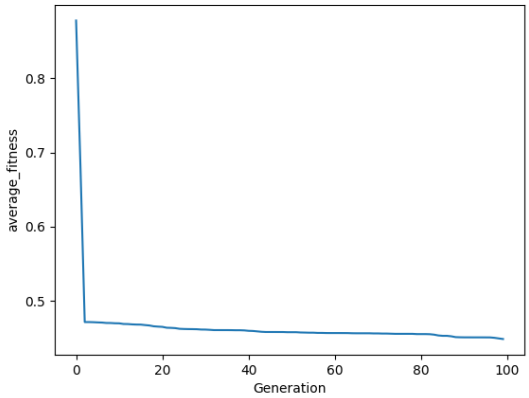


Figure 25: Test 2 with *crowding*: Cosine similarity.

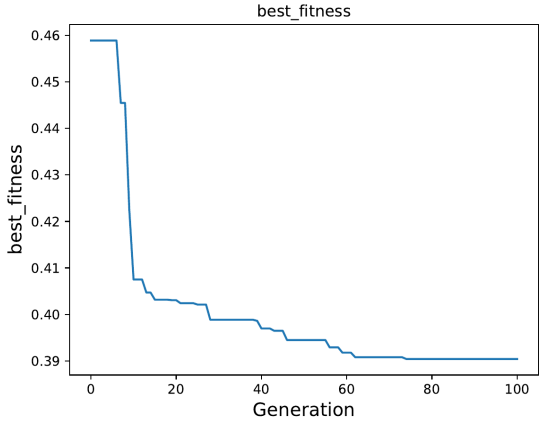
(a) Best fitness chart - Execution 1.



(b) Elite average fitness chart - Execution 1.



(c) Best fitness chart - Execution 2.



(d) Elite average fitness chart - Execution 2.

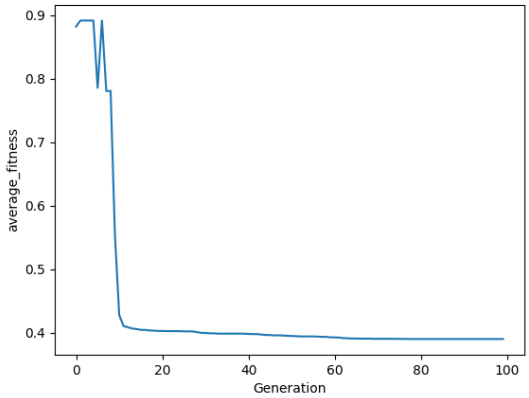


Table 20: Test 2 results.

Exec.	**Crowding**	**Trees**	**Rules**	**Avg. fitness**	**Avg. precision**
1	None	5	59	0.434224656	0.390292469
1	Cosine similarity	5	145	0.448512727	0.449475249
2	None	5	107	0.394239238	0.45678485
2	Cosine similarity	5	70	0.390404976	0.501740958

Conclusions Same outcomes were drawn without *crowding*: the function converged from more to less, as expected. Note that in graphs b and d of figure 25, values for the Gini impurity greater than 0.5 are reached. The cause of this is the penalty suffered by individuals similar to better ones, increasing their fitness and, therefore, worsening their value.

9.3. Test 3

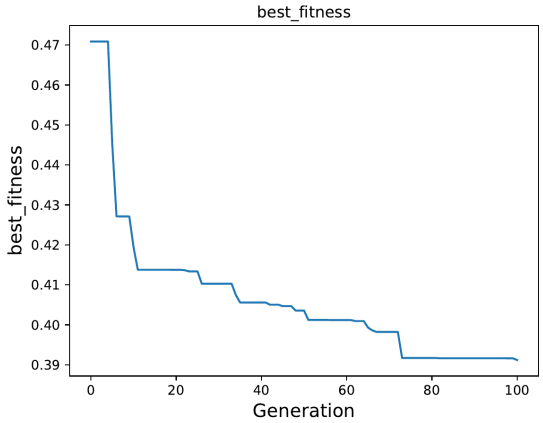
To carry out test 3, it was decided to limit the total penalization to 1, the double of the maximum value of Gini.

Table 21: Test 3 parameters.

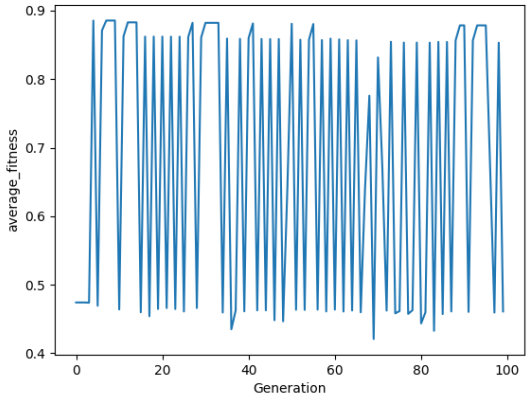
Parameter	**Value**
Crossover	Variable one-point
Crossover probability	0.75
Elite size	5
Fitness function	Weighted Gini Index
Generations	100
Initialization	Position Independent Grow
Maximum initial tree depth	10
Maximum tree depth	17
Mutation	Integer flip per codon
Mutation probability	None ($\frac{1}{genome\_length}$)
Population size	100
Replacement	Generational
Selection	Tournament
Tournament size	2

Figure 26: Test 3 with *crowding*: Cosine similarity.

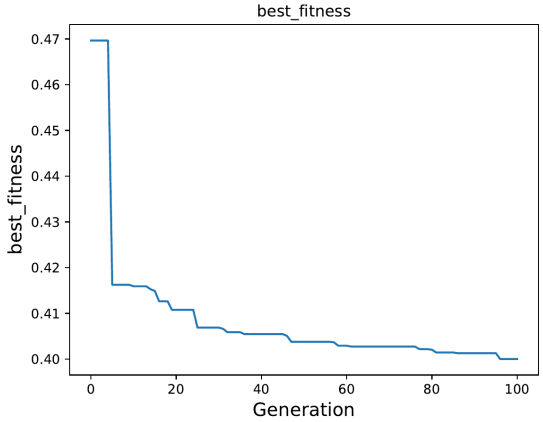
(a) Best fitness chart - Execution 1.



(b) Elite average fitness chart - Execution 1.



(c) Best fitness chart - Execution 2.



(d) Elite average fitness chart - Execution 2.

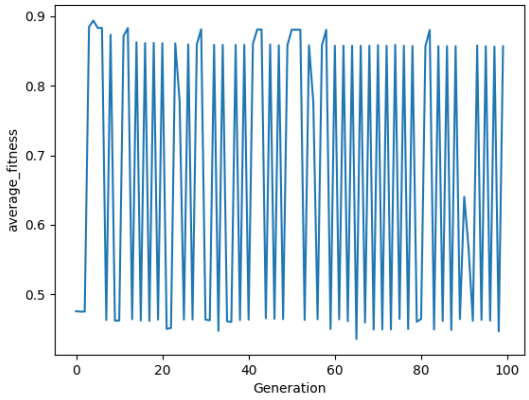


Table 22: Test 3 results.

Exec.	**Crowding**	**Trees**	**Rules**	**Avg. fitness**	**Avg. precision**
1	Cosine similarity	68	258	0.989195258	0.479473876
2	Cosine similarity	76	275	0.950907664	0.497586801

Conclusions Although the performance of the algorithm may seem good at first, it was not. The number of generated trees increased significantly, but with a low rate of really valid trees, that is, with Gini $<= 0.5$ (among 2.94%-9.21%). Graphs *a* and *c* show a correct behavior, since it only contains the best fitness of each generation, however, the statistics of the generated trees presented the introduced above. On the other hand, the elite average fitness chart shows constant ups and downs, possibly produced by the constant repetition of individuals.

While these data did not show an ideal performance, it seemed that the diversification worked roughly correctly, generating more individuals than without the crowding procedure. It will be seen later that this was not entirely correct either.

The limit established in the elite average fitness was eliminated, so that we could see which trees were heavily penalized.

9.4. Test 4

Revising test 3 code, it was found that the average fitness was having those ups and downs due to miss insertion of individuals in the new generation of the evolutionary process: the elite from the old population was being inserted at the beginning of the new population, when they might not really be the best in the new population. This also implied for penalized individuals. To fix this mistake, the whole new population was sorted before computing the average fitness.

After applying the solutions of the previous test, some functionalities were introduced with the aim of updating and filtering the results, extracting higher quality rules. In fact, new indicators were implemented, in order to clarify the results. These was previously commented in sections 8.2, 8.3 and 8.4:

- Updating rules with `precision` under 0.5.
- Filtering rules with `precision` under 0.7.
- Extract extra metrics[33].

Besides, after seeing that there were both similar trees and rules, two new measures were developed to evaluate the similarity or dissimilarity between individuals, which at that time had only been used the cosine similarity: the hamming distance and a similarity measure based on the number of shared attributes. Last, the average fitness chart is computed for the entire population instead of just the elite.

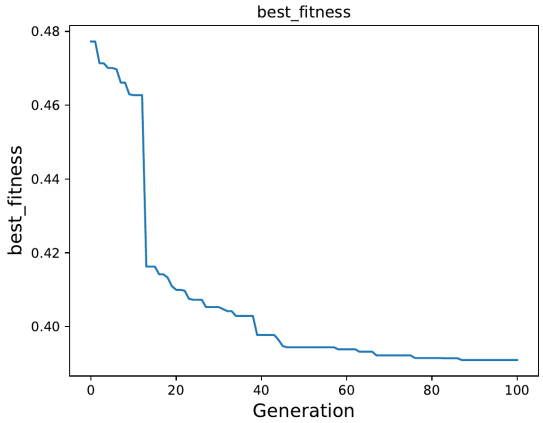
[33]See the entire metrics set in section 7.2.2, *Metrics*.

Table 23: Test 4 parameters.

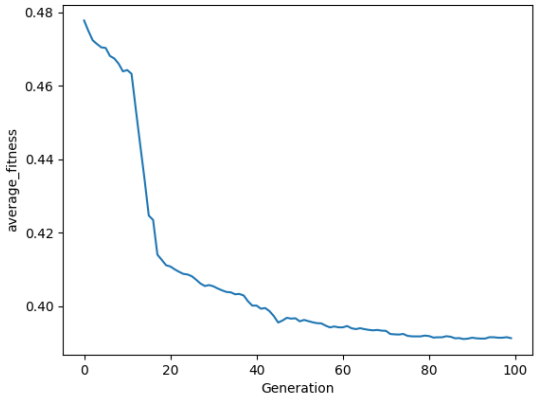
Parameter	**Value**
Crossover	Variable one-point
Crossover probability	0.75
Elite size	5
Fitness function	Weighted Gini Index
Generations	100
Initialization	Position Independent Grow
Maximum initial tree depth	10
Maximum tree depth	17
Mutation	Integer flip per codon
Mutation probability	None ($\frac{1}{genome\_length}$)
Population size	100
Replacement	Generational
Selection	Tournament
Tournament size	2

Figure 27: Test 4 with *crowding*: Hamming distance.

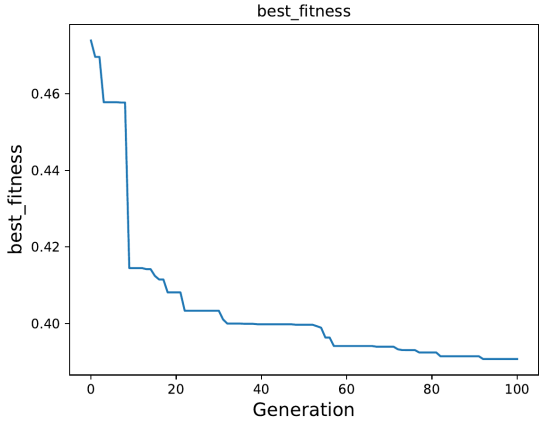
(a) Best fitness chart - Execution 1.



(b) Population average fitness chart - Execution 1.



(c) Best fitness chart - Execution 2.



(d) Population average fitness chart - Execution 2.

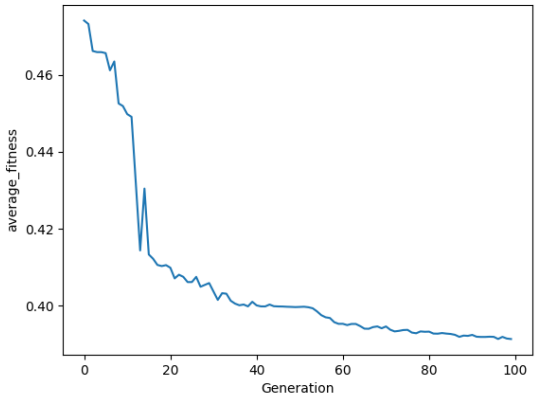
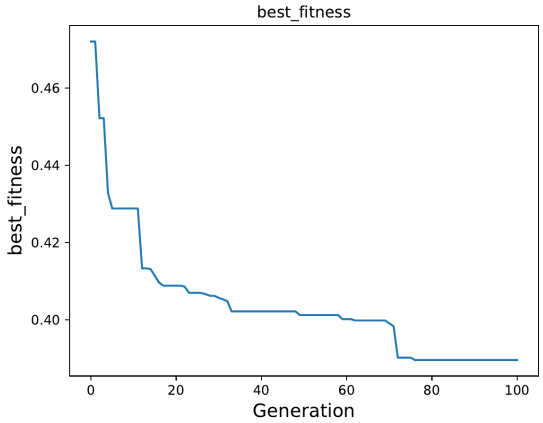
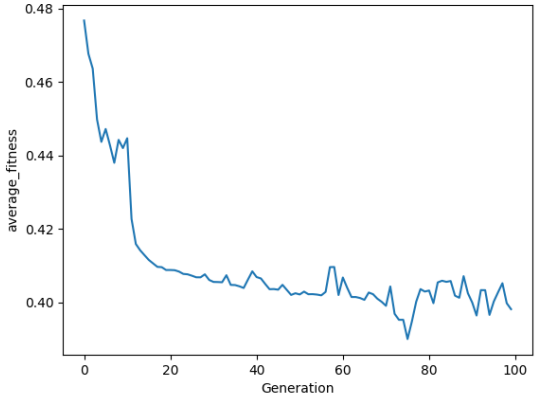


Figure 28: Test 4 with *crowding*: N. Shared Attributes.

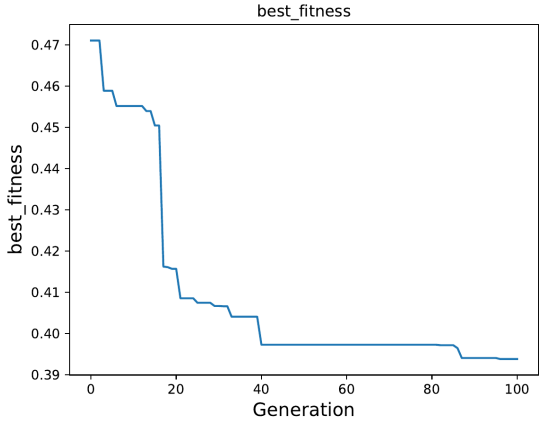
(a) Best fitness chart - Execution 1.



(b) Population average fitness chart - Execution 1.



(c) Best fitness chart - Execution 2.



(d) Population average fitness chart - Execution 2.

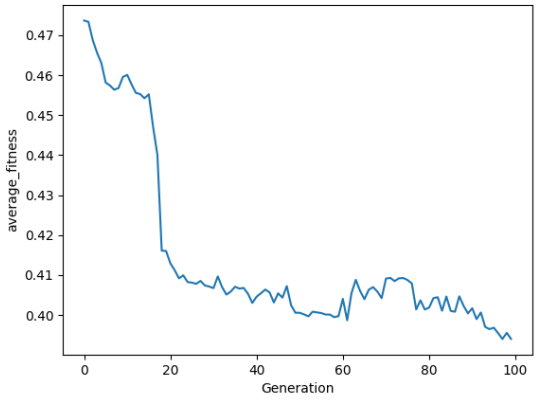


Table 24: Test 4 results (1).

Exec.	**Crowding**	**Trees**	**Rules**	**Avg. fitness**
1	Hamming distance	59	22	0.436662083
1	N. Shared attributes	70	46	0.45347102
2	Hamming distance	59	87	0.443959389
2	N. Shared attributes	61	41	0.431027426

Table 25: Test 4 results (2).

%LHS.Sup. Min:Avg	%Conf. Min:Avg	Cond. ant.	Used attr.	%Attr.use freq.	%Unc. pos.
0.0659:1.46	81.81:96.38	{1,3,4,5,6,10} {8,2,4,3,3,2}	18	{86.36,0,0,0,0}	5.84
0.06596:3.09	72.72:95.24	{1,2,3,4,5,8} {6,14,7,12,6,1}	23	{91.30,2.17,0,0,0}	11.01
0.0659:1.50	73.33:94.65	{1,2,3,4,5,6,7,8,9,10,11,13} {3,21,20,3,2,9,12,8,4,1,3,1}	39	{93.10,3.44,0,0,0}	8.33
0.06596:3.83	75:95.34	{1,2,3,5,6,9} {4,22,7,6,1,1}	30	{95.12,2.43,0,0,0}	4.51

Conclusions Even though the amount of trees kept similar to previous executions, the amount of rules had been reduced considerably, due to the updating and filtering. This reduction brought higher quality rules, with an average precision among 94%-97% and with a total attribute use among 20%-45%, but still with some problems:

- In most of cases, very long rules were issued, upon the wanted threshold of 8.
- The attribute that appeared the most in each tree covered the majority of rules, in the best of cases 86.36%. This indicates the predominance of 1 attribute over the others.

At least, the percentage of uncovered positive patterns was low, i.e., most of the positive patterns were correctly classified respect to the target.

9.5. Test 5

Before giving a solution to what was mentioned in the previous paragraph, we realized that there were two significant errors in the implemented code. The first of these was related to diversification. As seen in the diagram *b* of figure 21, it is executed in two different parts: one in the procedure `search_loop` and another in the procedure `step`. While in `search_loop` it was executed correctly, in `step` the correct population was not being passed as a parameter: previously the population was updated with the operators of *selection*, *crossover* and *mutation* and was saved in another variable (`new_pop`), but the one used was the initial passed to said procedure, (`individuals`). In short, diversification was not fully applied.

The second was related to the percentage of positive-uncovered patterns. The way to calculate them had been as shown in the expression 24, while the correct way was as shown in the expression 25:

$$UP = \frac{(CP - CPP)}{CP} \cdot 100 \qquad (24) \qquad\qquad UP = \frac{(PP - CPP)}{PP} \cdot 100 \qquad (25)$$

Denoting:

- UP: Uncovered positive patterns.
- PP: Positive patterns.
- CP: Covered patterns.
- CPP: Covered positive patterns.

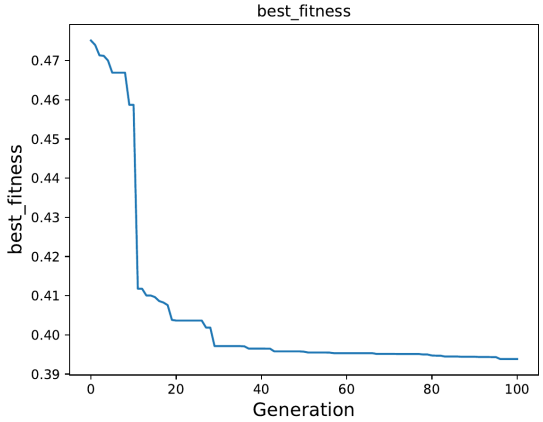
Thus, test 4 was rerun with both mistakes corrected.

Table 26: Test 5 parameters.

Parameter	**Value**
Crossover	Variable one-point
Crossover probability	0.75
Elite size	5
Fitness function	Weighted Gini Index
Generations	100
Initialization	Position Independent Grow
Maximum initial tree depth	10
Maximum tree depth	17
Mutation	Integer flip per codon
Mutation probability	None ($\frac{1}{genome\_length}$)
Population size	100
Replacement	Generational
Selection	Tournament
Tournament size	2

Figure 29: Test 5 with *crowding*: Cosine similarity.

(a) Best fitness chart.



(b) Population average fitness chart.

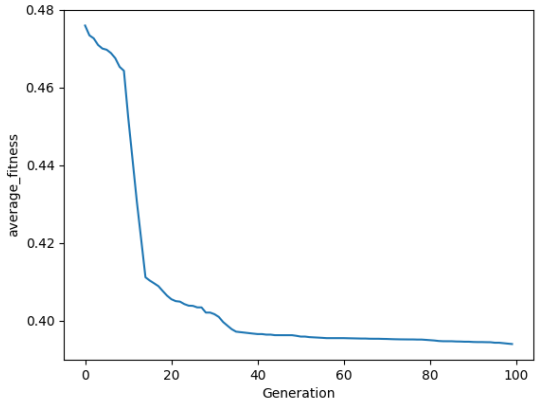
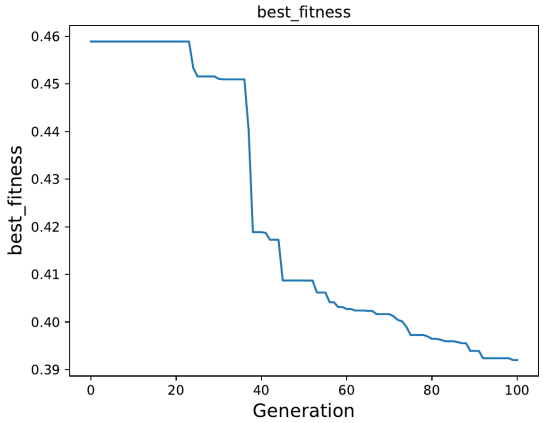


Figure 30: Test 5 with *crowding*: Hamming distance.

(a) Best fitness chart.



(b) Population average fitness chart.

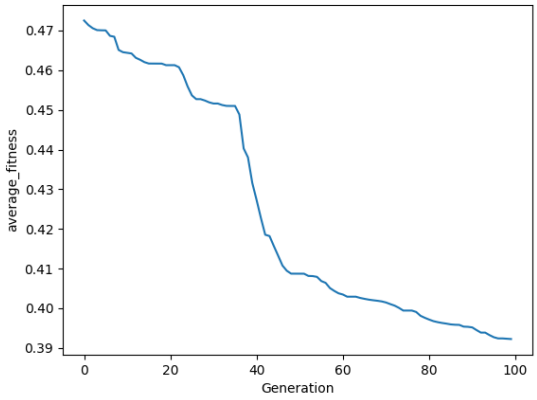
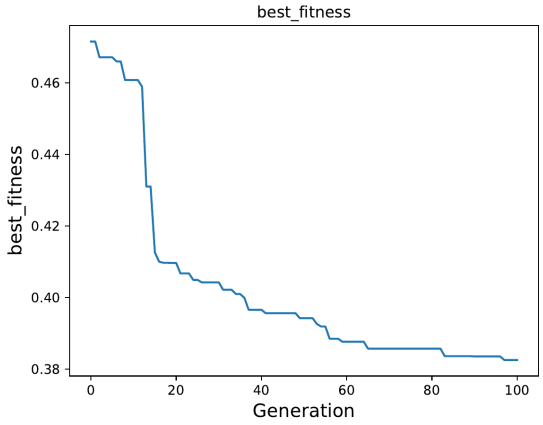


Figure 31: Test 5 with *crowding*: N. Shared Attributes.

(a) Best fitness chart.



(b) Population average fitness chart.

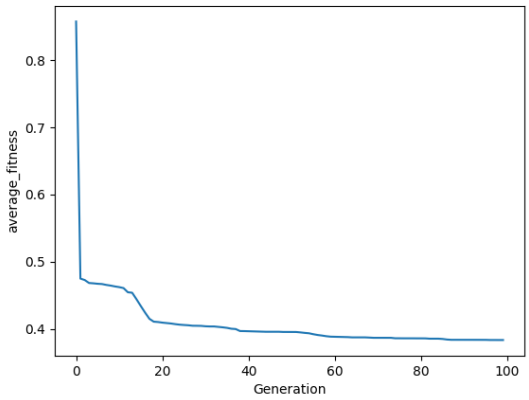


Table 27: Test 5 results (1).

Exec.	**Crowding**	**Trees**	**Rules**	**Avg. fitness**
1	Cosine similarity	60	52	4.593789383
1	Hamming distance	58	25	1.928986444
1	N. Shared attributes	69	23	4.860457536

Table 28: Test 5 results (2).

%LHS.Sup. Min:Avg	%Conf. Min:Avg	Cond. ant.	Used attr.	%Attr.use freq.	%Unc. pos.
0.0659:2.67	75:96.24	{1,2,3,4,5,6,7,8,11,12} {3,16,14,5,5,3,2,1,1,2}	33	{88.46,5.76,0,0,0}	74.67
0.0659:2.24	72.72:0.9252	{1,2,3,4,7,12,13,14} {2,1,1,10,2,2,3,4}	34	{92,4,0,0,0}	72.4
0.0659:2.14	75:94.13	{1,2,3,4,6,8,9,10} {4,6,5,1,2,1,2,2}	31	{78.26,4.34,0,0,0}	73.86

Conclusions The results of the three methods were slightly better in both plots than the results provided by the algorithm without diversification. However, there were new problems to be corrected, in addition to some that had not yet been solved:

- Numerous conditions.
- The attribute that appeared the most in each tree was still covering the majority of rules.
- When correcting the calculation of uncovered positive patterns, the indicator shot up to very high levels, above 70%.

The elite was eliminated for the following tests and the mutation probability was increased to 0.2, in order to obtain a population with a higher level of diversification.

Although the results were not completely as expected, the algorithm was working correctly at this point.

9.6. Test 6

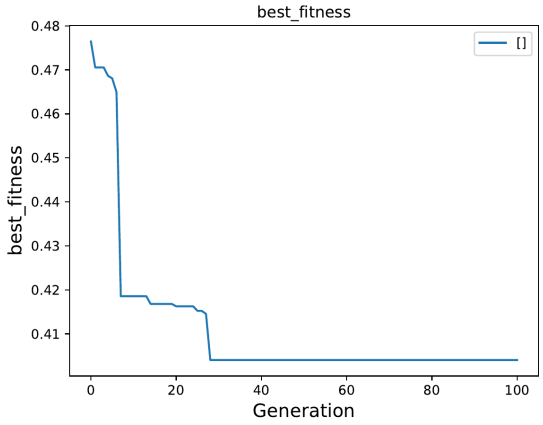
A PonyGE2 version with the pruning and correcting operators seen in section 7.2.2 was added as attempt to increase the efficiency of the algorithm.

Table 29: Test 6 parameters.

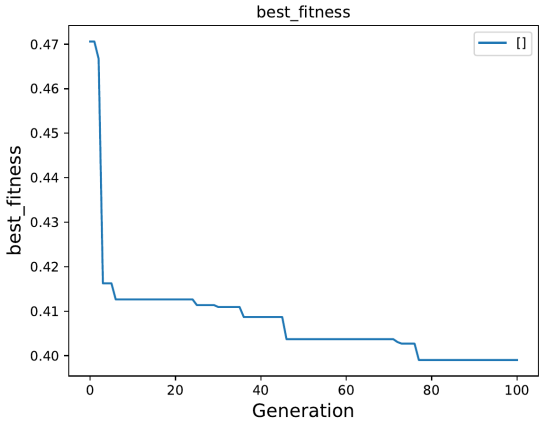
Parameter	**Value**
Crossover	multiple_x_operators
Crossover operators	subtree, prune_if_else_tree
Crossover probability	0.75
Elite size	0
Fitness function	Weighted Gini Index
Generations	100
Initialization	Position Independent Grow
Maximum initial tree depth	10
Maximum tree depth	17
Mutation	multiple_mut_operators
Mutation operators	subtree, prune_if_else_tree
Mutation probability	0.2
Population size	100
Replacement	Generational
Selection	Tournament
Tournament size	2

Figure 32: Test 6 with *crowding*.

(a) Best fitness chart: Cosine similarity.



(b) Best fitness chart: Hamming distance.



(c) Best fitness chart: N. Shared Attributes.

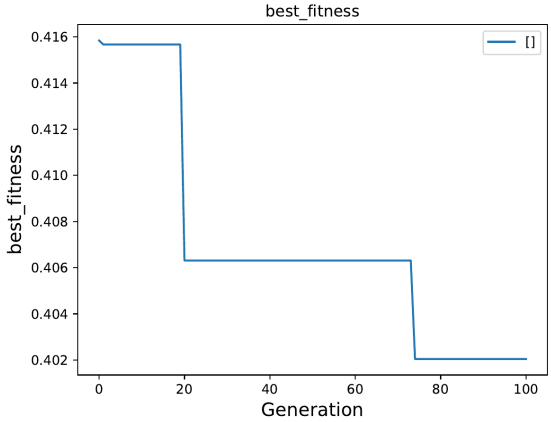


Table 30: Test 6 results (1).

Exec.	**Crowding**	**Trees**	**Rules**	**Avg. fitness**
1	Cosine similarity	83	38	22.34150326
1	Hamming distance	67	59	21.54072293
1	N. Shared attributes	81	21	22.44175443

Table 31: Test 6 results (2).

%LHS.Sup. Min:Avg	%Conf. Min:Avg	Cond. ant.	Used attr.	%Attr.use freq.	%Unc. pos.
0.0659:2.85	75:97.44	{1,2,3,4,5,6,7,8} {5,5,3,5,11,4,4,1}	50	{57.89,5.26,2.63,0}	70.61
0.0659:2.33	70:90.1	{1,2,3,4,5,6,7,8,9,10,11} {1,18,1,1,2,7,7,12,4,1,5}	39	{91.52,5.08,0,0,0}	68.5
0.0659:1.11	73.33:93.24	{1,2,3,5,6,7,8,9} {1,7,2,4,1,3,2,1}	36	{38.1,7.14,0,0,0}	71.26

Conclusions No significant changes were found.

9.7. Test 7

In pursuance of checking if a lower percentage of uncovered positive patterns was achieved with another fitness function, the Weighted Gini Impurity was changed to F1-Score[34] [57], which comes by default in PonyGE2's `classification.py`.

This implied two major changes:

1. Returning to the use of the penalty function previously seen in formula 23: due to the best F1-Score value is 1 instead of 0, the way to penalize individuals is by dividing its fitness by its associated crowding, in order to decrease the value.

2. Changing the *zeros* list to a *ones* list in algorithm 3: if an individual got no penalization, a division by 0 would be executed with a *zeros* list popping an exception; with a *ones* list, if the individual got no penalization its fitness would keep the same value.

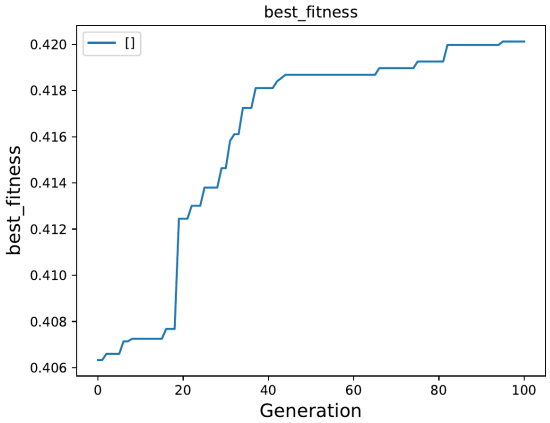
[34]The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal [56].

Table 32: Test 7 parameters.

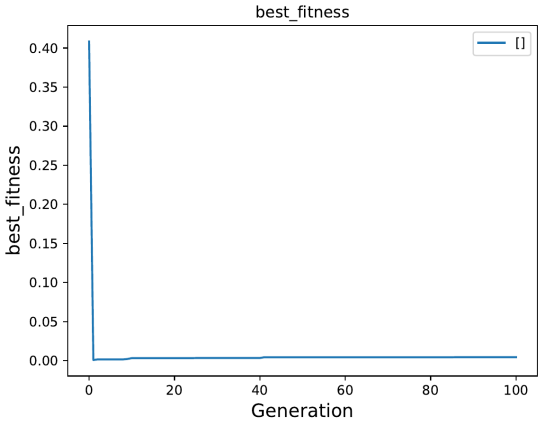
Parameter	**Value**
Crossover	multiple_x_operators
Crossover operators	subtree, prune_if_else_tree
Crossover probability	0.75
Elite size	0
Fitness function	F1-Score
Generations	100
Initialization	Position Independent Grow
Maximum initial tree depth	10
Maximum tree depth	17
Mutation	multiple_mut_operators
Mutation operators	subtree, prune_if_else_tree
Mutation probability	0.2
Population size	100
Replacement	Generational
Selection	Tournament
Tournament size	2

Figure 33: Test 7.

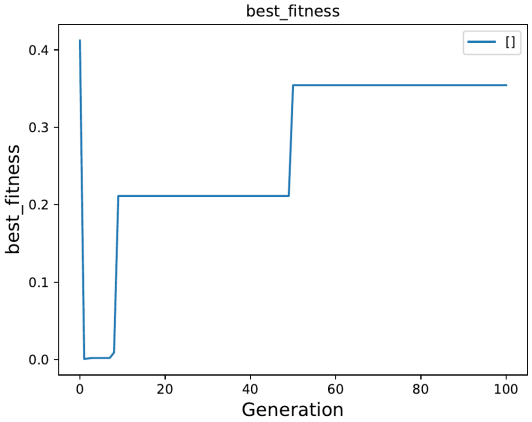
(a) Best fitness chart: No *crowding*.



(b) Best fitness chart: Cosine similarity.



(c) Best fitness chart: Hamming distance.



(d) Best fitness chart: N. Shared Attributes.

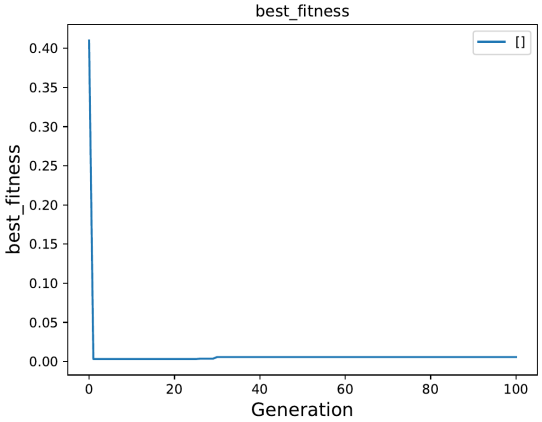


Table 33: Test 7 results (1).

Exec.	**Crowding**	**Trees**	**Rules**	**Avg. fitness**
1	None	100	47	0.384760125
1	Cosine similarity	70	9	0.00010926
1	Hamming distance	99	69	0.001616331
1	N. Shared attributes	88	30	0.000377606

Table 34: Test 7 results (2).

%LHS.Sup. Min:Avg	%Conf. Min:Avg	Cond. ant.	Used attr.	%Attr.use freq.	%Unc. pos.
0.0659:0.1	75:98.61	{5,6,7,8,9,10,11,13,14,15,18} {2,1,5,2,7,5,1,1,10,11,2}	48	{97.87,7.44,2.12,2.12,0}	97.72
0.0659:0.46	70:88.18	{1,2,3,6,8,10} {2,3,1,1,1,1}	24	{33.33,11.11,0,0,0}	92.20
0.0659:0.12	71.42:98.66	{7,9,10,11,12,13,14,15,16,17} {1,6,20,12,8,4,1,13,3,1}	58	{98.55,13.76,2.89,2.89,0}	91.88
0.0659:0.27	80:98.79	{2,3,4,13,14,15,17,18} {1,3,8,3,1,12,1,1}	55	{60,10,3.33,3.33,0}	91.88

Conclusions Good results were not obtained: the percentage of uncovered positive patterns was not reduced, but also increased and, in fact, the average fitness of the generated trees also worsened.

9.8. Test 8

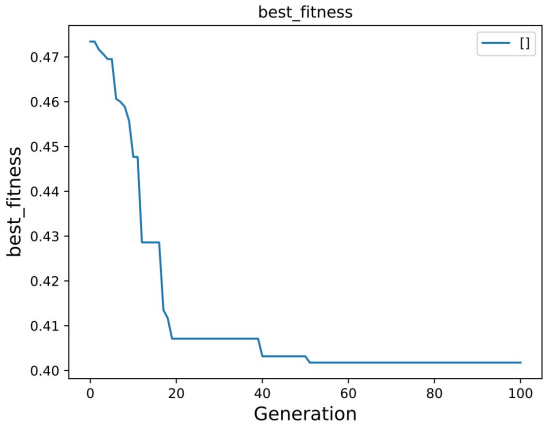
With the same objective as the previous test, the original fitness function has been returned and the model of diversification according to importances exposed in section 8.6 has been introduced.

Table 35: Test 8 parameters.

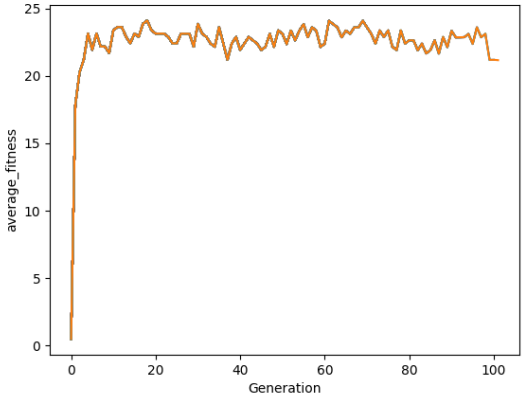
Parameter	Value
Crossover	multiple_x_operators
Crossover operators	subtree, prune_if_else_tree
Crossover probability	0.75
Elite size	0
Fitness function	Importance disimilarity
Generations	100
Initialization	Position Independent Grow
Maximum initial tree depth	10
Maximum tree depth	17
Mutation	multiple_mut_operators
Mutation operators	subtree, prune_if_else_tree
Mutation probability	0.2
Population size	100
Replacement	Generational
Selection	Tournament
Tournament size	2

Figure 34: Test 8 with *crowding*: Importance of attributes.

(a) Best fitness chart.



(b) Population average fitness chart.



(c) % Unc. positive patterns chart.

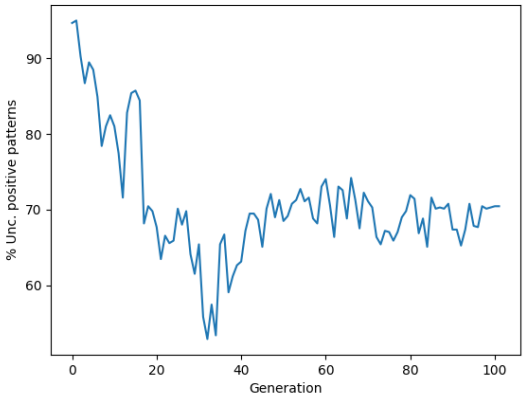


Table 36: Test 8 results (1).

Exec.	**Crowding**	**Avg. Trees**	**Avg. Rules**	**Avg. fitness**
1	Importance disimilarity	83	71	21.1710085

Table 37: Test 8 results (2).

Avg. %LHS.Sup. Min:Avg	**Avg. %Conf. Min:Avg**	**Avg. Used attr.**	**Avg. %Unc. pos.**
0.0659:3.5960243	72.63:94.41	55	70.72

Conclusions The percentage of uncovered positive patterns is still higher that desired. Note that an additional chart showing this indicator through the process has been provided, where it can be seen that it descends through generations but eventually, it keeps ranging at a high-undesired level, as commented. This is the last attempt of reducing the value of the indicator.

Two last comments regarding to the tests is that:

1. Good fitness values were produced for the best individual, compared to the original algorithm.
2. Diversification is appreciated, since the average fitness of the population is much higher.

10. Experimentation

This section details the results obtained for the search for association rules in the patients of the dataset who have complications. 616 patients with this characteristic can be found within the dataset, out of 1516. The results will be compared with others obtained by different approaches over the same dataset, previously obtained and studied in [42]. Notice that it is highly recommended to read the referenced research, due to some remarkable clarifications of its results.

10.1. Results on complications

Table 38: Algorithms for class association rule mining statistics, from [42].

Alg	%LHS.Sup. Min:Avg	%Conf. Min:Avg	#rules	|antecedent| {1,2,3,4,5}	Used attrs.	Attr. %used freq.	%Unc.pos.
Apriori	5:-	90:-	0		-		
	5:-	80:-	0		-		
	5:5.8	70:72	996	{0,0,17,178,801}	52	{60,15,4,1,0}	39
	2.5:2.6	90:96	2	{1,0,0,0,1}	6	{50,...,50}	89
	2.5:2.8	80:82	907	{1,0,21,129,756}	65	{42,10,4,1,0}	36
	2.5:3.1	70:73	$3e^4$	{1,23,492,$5e^3$,$3e^4$}	85	{27,8,4,1,0}	3
	1:1.2	90:94	711	{1,17,101,592,0*}	82	{36,6,3,1,0}	37
	1:1.3	80:84	$1e^4$	{3,48,$1e^3$,e^4,0*}	85	{20,6,4,2,0}	3
	1:1.5	70:75	$9e^4$	{4,236,$6e^3$,$8e^4$,0*}	86	{13,6,4,3,0}	0
	0.5:0.7	90:94	$2e^4$	{3,90,$2e^3$,$2e^4$,0*}	86	{18,6,4,3,0}	3
	0.5:0.7	80:86	$1e^5$	{5,246,$8e^3$,$1e^5$,0*}	86	{13,6,4,3,0}	0
	0.5:0.8	70:77	$4e^5$	{7,759,$3e^4$,$4e^5$,0*}	86	{10,6,5,3,0}	0
Rules from AC methods							
CBA2	0.46:2.89	60:77	17	{17,0,0,0,0}	13	{24,6,6,6,6}	53
CBA2.KM	0.46:2.72	51:74	20	{20,0,0,0,0}	16	{20,10,5,5,5}	51
CPAR	0:0.86	0:90	$1e^3$	{5,120,439,397,198}'	84	{36,7,4,2,0}	2.92
CPAR.KM	0:0.37	0:92	$4e^3$	{6,170,772,$2e^3$,$1e^3$}'	86	{32,10,4,2,0}	1.14
FARCHD	0:5.24	0:58	277	{11,266,0,0,0}	83	{16,6,3,2,0}	2.27
FARCHD.KM	0:3.32	0:57	479	{11,2,466,0,0}	85	{20,6,3,1,0}	1.46
Heuristic operators							
1. RISTRE	2.5:3.9	70:74	189	{1,12,42,60,74}	58	{31,10,5,3,1}	20
2. Simpl.	1.0:4.2	70:73	156	{0,13,38,60,45}	56	{32,10,5,3,1}	17:14
3. One alt.	0.5:1.0	70:88	179	{1,23,65,67,23}	53	{27,9,3,2,1}	37:14
4. Cond. Drop.	0.6:2.3	70:76	217	{0,39,115,63,0}	52	{25,7,3,2,1}	23:12
Total	0.5:2.8	70:78	741	{2,87,260,250,142}	63	{28,7,3,2,0}	12
Experimental approach							
PonyGE2	0.0659:0.44	75:95.52	44	{0,2,0,1,4}'	39	{97.72,4.54,0,0,0}	63.63
PonyGE2.CS	0.0659:2.26	75:95.81	60	{0,0,24,8,4}'	53	{71.66,5,1.66,1.66,0}	71.26
PonyGE2.HD	0.0659:0.69	70.58:90.33	40	{0,1,1,1,11}'	48	{85,10,2.5,2.5,0}	81
PonyGE2.NSA	0.0659:4.73	80:96.23	38	{0,0,2,1,2}'	55	{71.05,10.52,2.63,2.63,0}	74.67
PonyGE2.ID	0.0659:2.81	70:96.80	39	{0,7,4,10,5}'	39	{89.74,2.56,0,0,0}	72.07

* indicates that the corresponding execution was interrupted because of a RAM memory consumption above that available (18GB for the process), and ', that the algorithm generated rules with up to nine conditions.

Table 38 exposes the statistics of the mined rules, from which the following information can be extracted:

Support and Confidence PonyGE2 produce too specific rules in all its versions, due to its high *confidence* and low *support* values. In the case of the minimum *support*, it can be seen that there is always a rule that covers just one pattern ($\frac{1}{1516} = 0.000659$). It is clear that the vanilla and the one with the Hamming distance crowding diversification versions stand out in this sense (high *confidence* and low *support* values). Nevertheless, notice that the fact that the minimum *support* is greater than 0 means that all rules cover at least one pattern. This is not, for instance, the case of CPAR, CPAR.KM, FARCHD and FARCHD.KM, where invalid rules (those not covering any pattern) are produced due to the fact of having generated rules from a dataset with many missing values artificially imputed [42].

Number of rules PonyGE2 is one of the algorithms that extract less rules, just overtaken by versions 1, 2 and 3 of Apriori and both CBA2 approaches. In reality, it can mine many more rules, up to 385 with its Hamming distance variant, but they got narrowed down by the implemented filters, in order to obtain higher quality rules. Its examination may be less tedious than in the cases in which so many are obtained, especially in those cases in which the complexity increases with a high number of antecedent conditions. Nevertheless, we think that this could easily be tuned by adjusting the size of the population of the algorithm.

Number of conditions Apriori exploits all the possible combinations for the established *support* and *confidence*, AC methods got closed with a maximum of 3 conditions, except CPAR and CPAR.KM, which are up to more than 9 conditions within its antecedents; and the heuristic operators keep to 5 conditions with a *"bell-shape distribution"*. PonyGE2 versions have distributions similar to those of the heuristic operators. Although it is not shown in the table, rules having up to 15 conditions got generated, having its higher concentration in the middle.

Number and distribution of relevant features This has been one of the main problems that have been tried to address when extracting association rules and, however, it has not been possible to solve it so far: when it comes to a high percentage in the most used attribute, having generated few rules, it indicates that that attribute predominate over the others. Thus, PonyGE2 associate positive cases with more similar descriptions. On the other hand, the opposite occurs in the case of Apriori and AC methods, except CBA2 and CBA2.KM, where similar conclusions as PonyGE2 are reached. CPAR and CPAR.KM with the heuristic operators got also similar results to each other, reaching more distributed results in the first ones, due to a greater number of rules.

Note that having a huge amount of rules with distributed attributes may harden the study of rules and, also, it is more likely to generate trivial and/or uninteresting rules.

Uncovered positives Another PonyGE2 issue has been the uncovered positive patterns ratio, which has not been possible to reduce: it has the most raised levels. This happens due to a kind of "monopolization" from the most used attribute, that cannot involve all the patterns. The nature of the algorithm causes trees to be created with a root node (attribute), from which all the rules are born. This implies that the percentage of appearance of these attributes is much higher than those of the branches, therefore, it explains its hoarding. Again, Apriori and the AC methods, also with the exception of CBA2 and CBA2.KM because of producing too few rules, cover almost the total of positive cases. The heuristic operators avoids the eluding patterns and focuse to the most discriminant conditions through a guided search, which penalizes the total coverage of positives in a good way.

10.2. Some potentially interesting rules

Table 39: Some potentially interesting rules.

Rule	Antecedent	Precision	Support	Lift	Covered
R1	P_N_IQ_MES > 1.0	0.95	0.09	2.36	148
R2	ESTADO_MESORRECTO != Moderado & P_N_IQ_MES > 2.0	1	0.03	2.46	46
R3	ENDOSPONGE ∈ (No, Si)	0.9	0.02	2.23	43
R4	C_EPIDURAL = True & RESECCION_CILINDRIC = Si & CIRUJANO = 12 & ENDOSPONGE = No & TIEMPO_OPERATORIO <= 215.0	0.91	0.02	2.24	34
R5	C_EPIDURAL = True & LOCALIZACION ∈ (Angulo Hepatico, Colon Descendente, Colon Transverso, Pancolonico, Recto Superior, Union Rectosigmoidea) & CIRUJANO = 12 & ENDOSPONGE = No & TIEMPO_OPERATORIO <= 215.0	0.85	0.01	2.09	20
R6	P_SODIO > 131.0 & P_CARDIACO ∈ (ICC Leve, ICC Moderada, No Aplicable) & P_DISNEA ∉ (Moderada, No, Severa)	0.8	0.007	1.97	15
R7	SECCION_TUMORAL = Si & DIAGNOSTICO ∉ (Adenoma Velloso, Displasia, Diverticulosis, No Patologia, Peritonitis, Poliposis Colonica) & ESPESOR_MAXIMO <= 0.7	0.73	0.007	1.8	15
R8	SECCION_TUMORAL = Si & ASA ∉ (I, II, No Aplicable) & DIAGNOSTICO ∉ (Adenoma Velloso, Diverticulosis, No Patologia, Obstruccion Intestinal, Otras Lesiones, Peritonitis)	0.78	0.007	1.93	14
R9	PERFORACION_IO = Espontanea & MESTASTASIS ∈ (No, No Aplicable, Otras, Peritoneo)	0.83	0.006	2.05	12
R10	P_TIPO_IQ != Menor & PATOLOGO != 20 & SECCION_TUMORAL ∉ (No, Si) & GANGLIOS_AISLADOS <= 23.0 & P_TIPO_IQ ∈ (Mayor Compleja, Menor) & DISTANCIA_ANO > 37.0	0.75	0.002	1.84	4
R11	DIAGNOSTICO = Cancer Sincronico & MESTASTASIS ∈ (Higado, No Aplicable, Otras)	1	0.0006	2.46	1

Some information that can be extracted from table 39:

- All the extracted rules have a *lift* > 1, which means that the antecedent actually

leads to the consequent, i.e., the probability of complications is among 1.8-2.46 (depending on the rule) times greater with the given antecedent than without it.

- R1 and R2 indicate that the attribute P_N_IQ_MES produces complications with a high probability, near to 1, when its value is higher than 1. In the case of R2, it seems that when mesorectum condition is different from moderate, complications are a sure thing.
- R4 and R5 cover multiple cases treated by a particular surgeon, accompanied by other circumstances, where complications appeared with probability 0.9 and 0.91, respectively.
- Middle and last rules may be interesting for an expert, due to the conjunction of multiple symptoms. This makes a more specific analysis rather, than more generic, and therefore may be rarer.

11. Conclusions

This project is an attempt of proposal of a methodology for obtaining interesting class association rules from datasets by using Gramatical Genetic Programming in Python.

PonyGE2 is a novel and might be a potentially powerful approach to do such task. The algorithm generates trees in which the nodes are identified as attributes, the leaf nodes as the class label and the branches as the entire association rule. Following the CART theory, a Gini impurity index has been developed to measure the misclassification of the observations to optimize the split, using the labels and its probability of appearance. The summation of each Gini of the generated rules that form a tree, weighted with the antecedents *support*, served as fitness function to identify the best trees among the generated. The algorithm was completed with a diversification procedure that aimed to increment the diversity of the rules by applying multiple techniques: Cosine similarity, Hamming distance, similarity based on the shared attributes among rules and the measure of the importances among attributes. Last, but not least, some filters where implemented in order to discriminate the best rules from the worst to decrease its quantity and ease its later study.

Since it is a first version of the algorithm for this task, it could be said that PonyGE2 has behaved quite well, but it still needs lots of polish. Although it is capable of extracting a medium-sized set of rules by defining a context-free grammar and adapting the original algorithm, the truth is that a considerable part of them presents deficiencies: some of them reach to a no-sense statement, due to the fact that contains attributes that are mutually exclusive (it is well know that this kind of issues may happens frequently in computer science); others may be obvious, because other rules already reach to the same conclusion with less conditions, so that, can be considered as repeated. Indeed, it spends time and computational resources in extracting this kind of rules, which can be partially corrected by applying certain filters and/or operators to the algorithm, although this does not always work. Moreover, two main problems has been issued during the developing and testing phases: the first one was handling with the "monopolization" of the rules by some attributes, which creates repeated and superfluous rules; the second, the lack of coverage of positive patterns, which links to the first reason. Although it has been tried to solve it with the application of the diversification techniques mentioned above and increasing the probability of mutation of the GA, it has not been successfully addressed.

However, breaking this barrier can present a very wide range of possibilities, in which the algorithm can generate curious rules such as those presented in table 39 and, with luck, can generate greatly valuable information for experts.

References

[1] J.A. Delgado-Osuna, Carlos García-Martínez, and Sebastián Ventura. "Smart Operators for Inducing Colorectal Cancer Classification Trees with PonyGE2 Grammatical Evolution Python Package". In: *IEEE Congress on Evolutionary Computation (IEEE CEC 2022)* (2022). Aceptado y pendiente de publicación (2022-06-04).

[2] A.E. Eiben and J.E. Smith. "Introduction to evolutionary computing". In: *Natural Computing Series* (2015). DOI: 10.1007/978-3-662-44874-8.

[3] *Cancer*. URL: https://www.who.int/en/news-room/fact-sheets/detail/cancer.

[4] Max Roser and Hannah Ritchie. *Cancer*. 2015. URL: https://ourworldindata.org/cancer.

[5] *Defunciones por causas (lista reducida) por sexo y grupos de edad*. URL: https://www.ine.es/jaxiT3/Datos.htm?t=7947#!tabs-grafico.

[6] Farhad Islami et al. "Proportion and number of cancer cases and deaths attributable to potentially modifiable risk factors in the United States". In: *CA: A Cancer Journal for Clinicians* 68.1 (2017), 31–54. DOI: 10.3322/caac.21440.

[7] Rebecca L. Siegel et al. "Colorectal cancer statistics, 2020". In: *CA: A Cancer Journal for Clinicians* 70.3 (2020), 145–164. DOI: 10.3322/caac.21601.

[8] *Factores de Riesgo del Cáncer colorrectal*. URL: https://www.cancer.org/content/cancer/es/cancer/colon-rectal-cancer/causas-riesgos-prevencion/factores-de-riesgo.html.

[9] Priyanka Kanth and John M Inadomi. "Screening and prevention of colorectal cancer". In: *BMJ* (2021). DOI: 10.1136/bmj.n1855.

[10] Martin R. Weiser et al. "Development and assessment of a clinical calculator for estimating the likelihood of recurrence and survival among patients with locally advanced rectal cancer treated with chemotherapy, radiotherapy, and surgery". In: *JAMA Network Open* 4.11 (2021). DOI: 10.1001/jamanetworkopen.2021.33457.

[11] PonyGE. *Ponyge2 Wiki*. URL: https://github.com/PonyGE/PonyGE2/wiki.

[12] *Matplotlib*. Accessed: 2022-04-04. URL: https://matplotlib.org/.

[13] *Numpy*. Accessed: 2022-04-04. URL: https://numpy.org/.

[14] *SciPy*. Accessed: 2022-04-04. URL: https://www.scipy.org/.

[15] *Scikit-Learn*. Accessed: 2022-04-04. URL: https://scikit-learn.org/stable/.

[16] *Pandas*. Accessed: 2022-04-04. URL: https://pandas.pydata.org/.

[17] McDermott J. Fagan D. Forstenlechner S. Hemberg E. Fenton M. and M. O'Neill. "PonyGE2: Grammatical Evolution in Python". In: *Proceedings of 2017 GECCO Workshop on Evolutionary Computation Software Systems* (July 15-19, 2017), 8 pages. URL: https://arxiv.org/abs/1703.08535v2.

[18] Imielinski T. Swami A. N. Agrawal R. "Mining association rules between sets of items in large databases". In: *In Proceedings of the 1993 ACM SIGMOD international conference on management of data (ICDM '93)* (1993), pp. 207–216. DOI: 10.1145/170035.170072.

[19] José María Luna, Philippe Fournier-Viger, and Sebastián Ventura. "Frequent itemset mining: A 25 years review". In: *WIREs Data Mining and Knowledge Discovery* 9.6 (2019). DOI: 10.1002/widm.1329.

[20] Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar. *Introduction to data mining, Chapter 5*. 2nd. Pearson, 2019. URL: https://www-users.cse.umn.edu/~kumar001/dmbook/ch5_association_analysis.pdf.

[21] Mohammed Al-Maolegi1 and Bassam Arkok. "An improved Apriori algorithm for association rules". In: *International Journal on Natural Language Computing (IJNLC) Vol. 3, No.1* (February 2014). URL: https://arxiv.org/pdf/1403.3948.pdf.

[22] *Apriori: Association Rule Mining In-depth Explanation and Python Implementation*. Towards Data Science. URL: https://towardsdatascience.com/apriori-association-rule-mining-explanation-and-python-implementation-290b42afdfc6.

[23] R. Agrawal and R. Srikant. "Fast algorithms for mining association rules in large databases". In: *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases* (1994), pp. 487–499.

[24] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and techniques 3rd edition*. Elsevier Science, 2011, p. 194.

[25] *Frequent Pattern Growth Algorithm*. Towards Data Science. URL: https://towardsdatascience.com/fp-growth-frequent-pattern-generation-in-data-mining-with-python-implementation-244e561ab1c3.

[26] Pei J. Yin Y. Han J. "Mining frequent patterns without candidate generation." In: *ACM SIGMOD Record* (June 2000), pp. 1–12. DOI: 10.1145/335191.335372.

[27] *Depth Search Algorithm*. Wikipedia. URL: https://en.wikipedia.org/wiki/Depth-first_search.

[28] *Operador de mutación genética*. Wikipedia. URL: https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm).

[29] "A survey of genetic programming and its applications". In: *KSII Transactions on Internet and Information Systems* 13.4 (2019). DOI: 10.3837/tiis.2019.04.002.

[30] M. O'Neill and C. Ryan. "Grammatical evolution". In: *IEEE Transactions on Evolutionary Computation* 5.4 (2001), pp. 349–358. DOI: 10.1109/4235.942529.

[31] José María Luna, Mykola Pechenizkiy, María José del Jesus, and Sebastián Ventura. "Mining Context-Aware Association Rules Using Grammar-Based Genetic Programming". In: *IEEE Transactions on Cybernetics* 48.11 (2018), pp. 3030–3044. DOI: 10.1109/TCYB.2017.2750919.

[32] Pedro Espejo, Cristóbal Romero, Sebastian Ventura, and Cesar Martínez. "C. Induction of Classification Rules with Grammar-Based Genetic Programming". In: (Jan. 2005).

[33] José M. Luna, José Raúl Romero, and Sebastián Ventura. "Design and behavior study of a grammar-guided genetic programming algorithm for Mining Association rules". In: *Knowledge and Information Systems* 32.1 (2011), 53–76. DOI: 10.1007/s10115-011-0419-z.

[34] Michael O'Neill, Erik Hemberg, Conor Gilligan, Eliott Bartley, James McDermott, and Anthony Brabazon. "GEVA". In: *ACM SIGEVOlution* 3.2 (2008), 17–22. DOI: 10.1145/1527063.1527066.

[35] Michael O'Neill, Erik Hemberg, Conor Gilligan, Eliott Bartley, James McDermott, and Anthony Brabazon. "GEVA (v2.0)". In: (2011). URL: http://ncra.ucd.ie/GEVA/geva.pdf.

[36] David Fagan and Eoin Murphy. "Mapping in grammatical evolution". In: *Handbook of Grammatical Evolution* (2018), 79–108. DOI: 10.1007/978-3-319-78717-6_4.

[37] Fadi Thabtah. "A review of Associative Classification Mining". In: *The Knowledge Engineering Review* 22.1 (2007), 37–65. DOI: 10.1017/s0269888907001026.

[38] Bing Liu, Yiming Ma, and Ching-Kian Wong. "Classification Using Association Rules: Weaknesses and enhancements". In: *Data Mining for Scientific and Engineering Applications* (2001), 591–605. DOI: 10.1007/978-1-4615-1733-7_30.

[39] Bing Liu, Yiming Ma, and Wynne Hsu. "Integrating Classification and Association Rule Mining". In: *KDD'98: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (1998), pp. 80–86.

[40] Xiaoxin Yin and Jiawei Han. "CPAR: Classification based on Predictive Association rules". In: *Proceedings of the 2003 SIAM International Conference on Data Mining* (2003). DOI: 10.1137/1.9781611972733.40.

[41] Jesús Alcala-Fdez, Rafael Alcala, and Francisco Herrera. "A Fuzzy Association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning". In: *IEEE Transactions on Fuzzy Systems* 19.5 (2011), 857–872. DOI: 10.1109/tfuzz.2011.2147794.

[42] José A. Delgado-Osuna, Carlos García-Martínez, José Gómez-Barbadillo, and Sebastián Ventura. "Heuristics for interesting Class Association rule mining a colorectal cancer database". In: *Information Processing and Management* 57.3 (2020), p. 102207. DOI: 10.1016/j.ipm.2020.102207.

[43] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and regression trees*. CRC Press, 1984.

[44] *Python*. Accessed: 2022-04-27. Python Software Foundation. URL: https://www.python.org/.

[45] Microsoft. *Visual studio code - code editing. redefined*. 2021. URL: https://code.visualstudio.com/.

[46] *PyCharm: El IDE de Python para desarrolladores profesionales*. Accessed: 2022-06-04. URL: https://www.jetbrains.com/es-es/pycharm/.

[47] *Git*. Accessed: 2022-04-27. URL: https://git-scm.com/.

[48] *GitHub*. Accessed: 2022-04-27. URL: https://github.com/.

[49] *Overleaf, editor de Latex Online*. Accessed: 2022-04-27. URL: https://es.overleaf.com/.

[50] Jmmcd. *Ponyge: Grammatical evolution (GE) in one file.* URL: https://github.com/jmmcd/ponyge.

[51] PonyGE. *Grammars · PONYGE/Ponyge2 Wiki*. Accessed: 2022-04-27. URL: https://github.com/PonyGE/PonyGE2/wiki/Grammars#variable-ranges-in-grammars.

[52] PonyGE. *Evaluation · PONYGE/Ponyge2 Wiki*. Accessed: 2022-04-27. URL: https://github.com/PonyGE/PonyGE2/wiki/Evaluation#additional-functionality.

[53] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56 –61. DOI: 10.25080/Majora-92bf1922-00a.

[54] Maumita Bhattacharya. "Diversity Handling In Evolutionary Landscape". In: (2014). DOI: https://doi.org/10.48550/arXiv.1411.4148.

[55] Maarten Grootendorst. *9 distance measures in Data Science*. Accessed: 2022-04-16. 2021. URL: https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa.

[56] *Sklearn F1-Score*. Accessed: 2022-05-13. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

[57] Thomas Wood. *F-score*. Accessed: 2022-05-11. 2019. URL: https://deepai.org/machine-learning-glossary-and-terms/f-score.

[58] *Attribute-Relation File Format (ARFF)*. URL: https://waikato.github.io/weka-wiki/formats_and_processing/arff_stable/.

[59] TylerMSFT. *ATL Registrar and Backus-Naur Form (BNF) syntax*. URL: https://docs.microsoft.com/en-us/cpp/atl/understanding-backus-naur-form-bnf-syntax?view=msvc-170.

[60] Kashif Iqbal. *CSV file format*. Accessed: 2022-06-05. 2019. URL: https://docs.fileformat.com/spreadsheet/csv/#csv-file-format.

A. Annex I: Structure of files

A.1. ARFF format

ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes [58].

Two well-differentiated parts must be taken into account in this type of file: (1) **Header**, where the name of the dataset, the attributes and its types are defined; (2) **Data**, where the data, forgive the redundancy, is stored. Also, the following tags (case insensitive) differentiate each of the previous components:

%: Comments within the file.

@RELATION: name of the dataset or set of relations.

@ATTRIBUTE: they are used to unambiguously declare the name of the attributes of the dataset, as well as their data type. The declaration order indicates its position in the columns of the dataset, with the first being the first column, and so on until the last, which occupies the last.

@DATA: used to indicate the beginning of the data.

Listing 16: ARFF file example.

```
% 1. Title: Colorectal Cancer Database
%
% 2. Sources:
%      (a) Creator: Hospital Universitario Reina Sofia (Cordoba, Espana)
%      (b) ...

@RELATION dataset

@ATTRIBUTE SEXO         {'Hombre','Mujer'}
@ATTRIBUTE P_EDAD       numeric
@ATTRIBUTE COLONOSCOPIA {'Completa','Incompleta','No Posible','No Realizada'}
@ATTRIBUTE ...          ...

@DATA
'Hombre',60,'Completa',...
'Mujer',45,'No Posible',...
'Mujer',72,'Incompleta',...
'Hombre',58,'Incompleta',...
...
```

More information about the definition of attributes and their data types, as well as the various ARFF files, can be found in the provided reference.

A.2. BNF format

BNF (Backus-Naur Form) is a metalanguage used to express context-free grammars, this is, a formal way of describing formal languages [59].

The general structure of BNF is as follows:

$$name ::= expansion \tag{26}$$

Where the symbol ::= means "may expand into" and "may be replaced with", in other words, a non-terminal symbol.

Names are placed inside angle brackets (< ... >), regardless of their position in the rulers (right or left). Expansions are expressions that contain either terminal symbols or non-terminal symbols. In the case of terminal symbols, these can be in the form of a literal ("1", "+", ...) or a category of literals (integer, float, ...). Last, vertical bars (|) indicates choice.

Listing 17: BNF file example.

```
1 <number> ::= <digit> | <number> <digit>
2 <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

In the previous listing you can see an example that shows how a "number" is a "digit", or any other "number" followed by a "digit". A "digit" can be one of the characters 0, 1, 2, ..., 9.

A.3. CSV format

CSV (Comma-Separated Values) is a simple open format document type for representing data in table-like, in which columns are separated by commas or semicolons, and the rows are separated by line breaks. An optional header can be also added, containing the same amount of fields than the records within the file.

Listing 18: CSV file example.

```
1 SEXO,P_EDAD,COLONOSCOPIA,...
2 'Hombre',60,'Completa',...
3 'Mujer',45,'No Posible',...
4 'Mujer',72,'Incompleta',...
5 'Hombre',58,'Incompleta',...
6 ...
```

It must be taken into account that there are some complications when using double quotes (“”), line breaks (CRLF) and even the commas themselves (,), inside the fields. You can see [60] where multiple of these complications can be seen, accompanied with some solutions.

A.4. TXT format for PonyGE2 parameters

Parameter files in PonyGE2 are in plain text format, adapting the **key-value** structure, in such a way that the **key** is placed on the **left**, separated by a colon (:) from the **value**, which is placed on the **right**. Since the file is read by a Python program, multiple values can be accepted through the use of Python lists, and subsequently accessed in the typical manner:

In parameters file:

Listing 19: PonyGE2 parameters file example: multiple values (1).

```
1 CROSS_OPERATORS:          [subtree, prune_if_else_tree]
```

Within the Python source code, `params['CROSS_OPERATORS']` is a `list`. You can access to its values by accessing to the corresponding index:

Listing 20: PonyGE2 parameters file example: multiple values (2).

```
1 params['CROSS_OPERATORS'][0] = subtree
2 params['CROSS_OPERATORS'][1] = prune_if_else_tree
```

The file that has been used to carry out the experiments is the following:

Listing 21: PonyGE2 parameters file example.

```
1  CACHE:                   True
2  CODON_SIZE:              100000
3  CROSSOVER:               multiple_x_operators
4  CROSS_OPERATORS:         [subtree, prune_if_else_tree]
5  CROSSOVER_PROBABILITY:   0.75
6  DATASET_TRAIN:           data.arff
7  DATASET_TEST:            data.arff
8  DEBUG:                   False
9  ELITE_SIZE:              0
10 ERROR_METRIC:            precision_and_recall_score
11 GENERATIONS:             100
12 MAX_GENOME_LENGTH:       500
13 GRAMMAR_FILE:            supervised_learning/
       if_else_classifier_heterogeneous_data.bnf
14 INITIALISATION:          PI_grow
15 INVALID_SELECTION:       False
16 MAX_INIT_TREE_DEPTH:     10
17 MAX_TREE_DEPTH:          17
18 MUTATION:                multiple_mut_operators
19 MUTATION_PROBABILITY     0.2
20 MUTATION_OPERATORS:      [subtree, prune_if_else_tree]
```

```
MIN_GINI_REDUCTION:      0.0
SIMPLIFY_MIN_PATTERNS:   0
POPULATION_SIZE:         100
FITNESS_FUNCTION:        supervised_learning.subclassification
REPLACEMENT:             generational
SELECTION:               tournament
TOURNAMENT_SIZE:         2
VERBOSE:                 False
SHARING_FITNESS:         True
SHARING_PROCEDURE:       importance disimilarity
```