UNIVERSIDAD DE GUANAJUATO

CAMPUS IRAPUATO - SALAMANCA
DIVISIÓN DE INGENIERÍAS

"Evolutionary Learning of Selection Hyper-Heuristics for Choosing the Right Method in Text Classification Problems"

Thesis

A thesis presented for the degree of:
Maestría en Ingeniería Eléctrica
(Instrumentación y Sistemas Digitales)

By:

Ing. Jonathán de Jesús Estrella Ramírez

Thesis Director:

Dr. Juan Carlos Gómez Carranza

Salamanca, Guanajuato September 2023

UNIVERSIDAD DE GUANAJUATO

CAMPUS IRAPUATO - SALAMANCA
DIVISIÓN DE INGENIERÍAS

"Aprendizaje Evolutivo de Hiperheurísticas de Selección para Elegir el Método Correcto en Problemas de Clasificación de Textos"

Tesis

Que para obtener el grado de:
Maestría en Ingeniería Eléctrica
(Instrumentación y Sistemas Digitales)

Presenta:

Ing. Jonathán de Jesús Estrella Ramírez

Director de Tesis:

Dr. Juan Carlos Gómez Carranza

Salamanca, Guanajuato Septiembre 2023

Abstract

Text classification is a common task in various areas of machine learning and has many applications. In this task, there are different problems such as email filtering, fake news detection, sentiment detection, etc. Different datasets can be used depending on the problem, but the optimal classification method can be specific for each one. Nevertheless, the process to find this optimal method is a complicated problem. In the scope of automated machine learning, different approaches have been developed to attack this problem; the most recent ones based on deep learning. In this thesis project, an evolutionary model, with the objective of generalizing the selection of methods in text classification problems through selection hyper-heuristics, is presented. Given a dataset, it is characterized by a group of 16 statistical meta-features that represent its data distribution. A hyper-heuristic consists of a set of rules of the if-then form, where each rule evaluates the group of meta-features to determine the appropriate classification method for that dataset. The evolutionary model begins with the creation of an initial population of hyper-heuristics, which, over the generations, is evolved through specific crossover and mutation operators. During each generation, the performance of the hyper-heuristics is calculated using a group of training datasets. In the last generation, the hyper-heuristic with the best performance is selected, and its final generalization is determined with a group of independent datasets. The results and analysis indicate that the best learned hyper-heuristic, in addition to having a good generalization, is more efficient than two state-of-the-art automated machine learning systems, with very similar overall performance.

Resumen

La clasificación de textos es una tarea común en diferentes áreas de aprendizaje de máquina y tiene muchas aplicaciones. Dentro de esta tarea, existen diversos problemas tales como filtrado de correo electrónico, detección de noticias falsas, detección de sentimientos, etc. Diferentes conjuntos de datos pueden ser usados dependiendo del problema, pero el método de clasificación óptimo puede ser específico para cada uno. Sin embargo, el proceso para encontrar este método óptimo es un problema complicado. En el ámbito de aprendizaje de máquina automatizado, diferentes enfoques han sido desarrollados para atacar este problema; los más recientes basados en aprendizaje profundo. En este proyecto de tesis, un modelo evolutivo, con el objetivo de generalizar la selección de métodos en problemas de clasificación de textos mediante hiper-heurísticas de selección, es presentado. Dado un conjunto de datos, este es caracterizado mediante un grupo de 16 meta-características estadísticas que representan su distribución de datos. Una hiper-heurística consta de un conjunto de reglas de la forma si-entonces, donde cada regla evalúa el grupo de meta-características para así determinar el método de clasificación adecuado para tal conjunto de datos. El modelo evolutivo parte de la creación de una población inicial de hiper-heurísticas, que con el paso de las generaciones, es evolucionada mediante operadores de cruza y mutación específicos. Durante cada generación, el desempeño de las hiper-heurísticas es evaluado mediante un grupo de conjuntos de datos de entrenamiento. En la última generación, la hiper-heurística con el mejor desempeño es seleccionada, y su generalización final es determinada con un grupo de conjuntos de datos independiente. Los resultados y análisis indican que la mejor hiper-heurística aprendida, además de contar con una buena generalización, es más eficiente que dos sistemas de aprendizaje de máquina automatizados del estado del arte, con desempeños generales muy similares.

Dedication

To my mother *María*

Acknowledgements

I want to thank my mother María, my sisters Brenda, Lucía, Juana, Laura and Yesenia, and my brothers Jesús and Gamaliel for being my support during this journey.

Thank you, Dr. Carranza, for giving me the opportunity to work on this project and also for your constant support during its development.

To all of you my greatest appreciation and gratitude.

Institutional Acknowledgements

Thanks to the University of Guanajuato for all the academic supports, and to CONACyT for the financial support under grant 806210.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, natural language processing (NLP), machine learning (ML), and data mining have experienced exponential growth in their respective areas of research and application, mainly due to the massive growth of data from Internet and other sources, and the involvement of large technology companies such as Google, Facebook, Amazon, Microsoft, among others [1]. And as a consequence, the interest of new researchers has been attracted in each of these fields. In the industry, the work that allows NLP and ML to be carried out has become very important, mainly because it allows analyzing large amounts of data and at the same time to extract information that may be relevant for decision-making. There is a wide variety of the different tasks covered by NLP, such as indexing and searching large texts, information retrieval, information extraction, speech recognition, automatic language translation, text classification, among others [1, 2]. Likewise, for each of these tasks, there is a wide variety of methods that could provide solutions to them.

One of the most popular tasks within these fields is text classification, due in large part to an exponential growth in data generation sources [2], such as social networks, news websites, e-commerce stores, etc. For example, in the case of the social network Twitter, in May 2022, it was estimated that there were around 867 million tweets sent per day [3]. Also, these sources have provided a large number of texts that have become more complex over the years, so various ML or deep learning (DL) methods have been developed to handle and analyze them more efficiently.

Text classification has a wide variety of applications and existing problems that can be solved with different methods and techniques that have been developed over the years [3, 4, 5]. Text classification is focused on different types of tasks such as sentiment analysis, fake news detection, news classification, email spam detection, authorship analysis, among others [6]. For each of the text classification tasks, there are different proposals that provide a solution to them. Commonly, in these pro-

[1] https://bit.ly/3UKhYqL
[2] https://bit.ly/3HgdcwX
[3] https://bit.ly/3Usc4cE

posals, different methodologies are used in each of the stages of the classification process, such as pre-processing, obtaining a textual representation, feature extraction, selection and configuration of classification methods [6, 7, 8], in such a way that a significant amount of time and computational resources are consumed to be able to determine which methodologies are the most appropriate for each of the stages, and thus achieve an adequate result for a given problem/dataset.

Thus, when addressing a specific problem of a text classification task, there is a wide search space, where different proposals provide different results. The main difference between these is in the classification method used (Naïve Bayes, support vector machines, k-Nearest neighbors, Random forest, etc.) because they work with different approaches (probabilistic, instance-based, rule-based, functions, ensembles, deep learning, etc.), which allows a wide diversity of configurations for each of them. This is the main problem, the selection of classification methods consumes the most time and computational resources since in this stage it is very common to try different classification methods with different configurations, in order to find the method and its configuration that allows obtaining the best performance for a given problem/dataset.

The abovementioned also occurs in other ML classification tasks, such as speech recognition or detection/recognition of people or objects in images or videos, and the only differences between these tasks are different data types, different distributions, and different features to represent them. In the case of text classification, it has been observed and investigated that a classification method can be optimal or near to it for those datasets that have similar meta-features [9, 10]. Meta-features are commonly based on statistics, and these provide insightful information about the data distribution in a dataset.

Various researchers have sought to attack the problem of the selection of classification methods with the aim of making this stage automated. In recent years, several works use the term "Automated Machine Learning" (AutoML) as a fundamental part, and they address different stages of the general classification process such as automated model selection, automated feature engineering, or something more specific as in the case of neural networks: automated neural architecture search [11, 12, 13]. It all started with the idea of developing a tool capable of automating the process of selecting classification methods and their adjustment, in problems that were classification or regression, from a given dataset (which may not have had pre-processing applied to it). And thus, removing the need for an ML expert, so that non-experts could apply ML to different problems and analyze their results for themselves. Generally, most tools today are capable of optimizing the method selection process for a single task for a single dataset [14, 15]. However, for the text classification task, most AutoML proposals have focused on pre-processing stages, in order to obtain adequate textual representations, and therefore have not addressed the method selection problem [16], without taking into account that developing an alternative to try to alleviate this problem would also achieve reductions

in the time and computational resources consumed in the search for the best classification method for a specific problem/dataset.

For this research, an evolutionary model is presented that seeks to generalize the method selection process for text classification tasks through evolutionary learning of hyper-heuristics, using a genetic algorithm approach, in such a way that the evolutionary model is capable of learning a set of hyper-heuristics that allow determining the most appropriate methods based on the meta-features based on statistics that describe the data distribution in each dataset. In addition, it is expected that the learned hyper-heuristics allow for optimal performance or close to it. The evolutionary model is tested with a total of 34 datasets, which correspond to different types of text classification tasks, and in turn, have different numbers of categories and documents. The evaluation of these hyper-heuristics is through the average macro F1 evaluation metric. The results obtained have shown that the proposed model allows the generation of hyper-heuristics that, when evaluated, result in a performance very close to the optimal value.

1.1 Motivation

Applying ML for the automation of tasks or processes has become very common and essential in recent years, mainly due to the continuous generation of large amounts of data in the digital world [4]. Therefore, it has also grown a lot in the alternatives it proposes to solve a wide variety of tasks, allowing the applications derived from it to act in an intelligent way, and be able to either hand in hand with humans or individually be more effective and efficient [5].

Despite the great advances that ML has had in recent years in the supervised learning part [17, 18] when dealing with classification or regression tasks, an expert in this area is still required to be able to successfully apply any of its methods and obtain a correct solution for a specific problem of a specific task. This is caused because although there is a wide variety of methods developed, none of these is capable of being superior to all the others in all possible scenarios. This has been demonstrated by theorem I of the theorems of "No free lunch" [19], which establishes the following: "Any two optimization algorithms are equivalent when their performance is averaged across all possible problems", endorsing the aforementioned.

AutoML tools have been developed to deliver solutions for these types of tasks, eliminating the need for an expert [14]. In any type of classification task, finding the best classification method for a single dataset is a complex problem due to the wide diversity of methods and configurations that can be tried. These problems have been addressed by AutoML tools with different methodologies. In addition, the use of these tools has become more common, mainly by non-experts in the area [20],

[4] https://bit.ly/3CgfwRT
[5] https://stanford.io/3LNTPM2

since these tools allow for a faster implementation of ML and sometimes better for the solution of a specific problem than that which could be provided by an expert, without the need for it.

Currently, most AutoML tools for any type of classification task search over a large space (in some cases, this space can be very complex), optimizing the method selection process according to a given dataset from a determined problem, for which there is no generalization of this. Furthermore, for the algorithm to converge and find the best solution for such a dataset, this optimization may be time-consuming if the search space is not bounded and also according to the data distribution of such a dataset. In the case of the text classification task, there are proposals that demonstrate that the meta-features of a dataset provide a lot of valuable information, which has allowed them to develop works that are more focused on the stages prior to the selection of methods, as is the optimization for obtaining textual representations. Although, there are also proposals that are focused on the method selection process, which mostly works with the same general approach mentioned in the previous paragraph.

The AutoML works developed for the task of text classification have allowed us to observe that it is still necessary to develop a framework that has a more general process for the selection of suitable classification methods. Therefore, this thesis project is aimed at research for the development of a framework of this type, involving the meta-features that allow a good representation of the data distribution of a dataset. This type of meta-features, based mainly on statistics, can be key to determining the optimal (or close to it) classification methods for different types of datasets belonging to different types of text classification tasks.

This thesis work is part of the line of research carried out in [9, 10], making a deeper analysis of classification methods, meta-features, hyper-heuristics, and in general. This will allow us to know if this type of methodology can be a promising path or alternative toward the automatic classification of texts in the selection process of classification methods.

1.2 Objectives

The general objective of this work is to create a framework that, through an evolutionary learning process, is capable of learning a set of hyper-heuristics capable of generalizing the selection process of classification methods based on meta-features extracted from a group of datasets and producing near-optimal performance. The specific objectives for the success of this work are the following:

- Define and extract a set of meta-features for a collection of datasets for text classification.
- Create a pool of different machine learning methods for text classification based on the following approaches: probabilistic (Naïve Bayes), instance-based

(k-Nearest Neighbors), rules (Decision Trees), and functions (Logistic Regression and Support Vector Machines).

- Design and create an evolutionary model based on genetic algorithms to learn hyper-heuristics, that applies evolutionary operators (selection, crossover, and mutation) to create and identify a set of general hyper-heuristics for method selection in text classification.

- Evaluate the performance of the learned hyper-heuristics using the macro F1 evaluation metric, over a group of test datasets.

1.3 Literature Review

In recent years, ML and NLP have significantly expanded text classification to analyze and solve several problems effectively [2, 18]. However, to address text classification as a task, it is necessary to have experience in the area of ML, in order to be able to apply the methods correctly and obtain results that can be considered acceptable. One of the biggest conflicts within this type of classification is determining which method will be used to classify the documents (samples), due to the large diversity of these. The methods belong to different approaches (probabilistic, instance-based, rule-based, functions, etc.), and most of them allow a large number of configurations in their hyperparameters (neighborhood number, distance metric, kernel, parameter regularization, etc.). To give a solution to a problem of some text classification task, what is usually done is to perform tests with various methods until finding the one that has the best classification performance based on the result of an evaluation metric such as accuracy or F1.

This conflict is found in a general way in any type of ML classification and has been named the CASH problem (Combined Algorithm Selection and Hyperparameter) [21]. Currently, CASH is still a problem where the skill of a human expert in ML intervenes a lot and, to find the best method and its respective configuration for a certain problem, the search is done exhaustively or guided based on trial and error.

AutoML arose in order to attack the selection of methods and their respective adjustment, and later it also focused on automating all the processes in which ML is involved, so it also has proposals that are aimed at automated neural architecture search [22, 23], automated feature engineering [24, 25], among others [14]. Regarding the automation of the selection of classification methods, some of the first works that started to use the term AutoML were Particle Swarm Model Selection [21], Auto-WEKA [26], and Hyperopt-sklearn [27].

During the early years, AutoML was not a popular term within ML, but its popularity grew thanks to the two ChaLearn AutoML Challenge [11] competitions. These competitions occurred between the years 2015 to 2018 and were focused on

supervised ML with the objective of training and testing various methods to solve different classification and regression problems without requiring human skill. The datasets/problems could contain data from various sources such as audio, images, speech, sensor data, text, and video, among others. In addition to being fully automated, the systems developed had to be computationally efficient in order to provide solutions to datasets with categorical values, irrelevant variables, missing values, and unbalanced classes, among others; regardless of whether it was binary or multi-class classification, and to be able to handle dimensionality of up to 300,000 features with sparse or full matrices.

The main challenge in the first competition (held between 2015 and 2016) consisted of 6 rounds that in turn had multiple phases (AutoML, Tweakathon, Final). For this competition, the computational resources were limited and when passing the round the difficulty had a considerable increase. In each round, the systems had a time limit of 20 minutes to provide a solution to each of the datasets provided (5 datasets in total), along with each dataset a scoring metric was provided so that the system would optimize the solution from this. The standings were determined by the average rank obtained and the average performance of the five datasets in each round. The team *AAD Freiburg* won the competition with the system called AutoSklearn [28]. This system creates an ensemble from machine learning methods that might work well with a given dataset, using a meta-model based on Bayesian optimization. The meta-model performs the comparison of datasets in a repository by extracting 38 meta-features, allowing you to select the possible machine-learning methods.

For the second competition (held in 2018), 5 datasets were provided to the competitors to develop and improve their systems (development phase). And for the final phase (AutoML blind test) the systems were tested with 5 datasets not disclosed to the competitors. The datasets for this competition were more complex than the previous one, resulting in lower overall performance. The winning system was an enhancement to AutoSklearn called PoSH (Portfolio Successive Halving) Auto-Sklearn [29] and developed by the *AAD Freiburg* team. This system implemented new improvements in terms of candidate configurations for every dataset; a new technique to remove methods that have poor performance; and the implementation of successive halving with Bayesian optimization to get efficient and robust results in a given time.

In recent years, the application of AutoML within text classification has had different approaches and methodologies, which have allowed certain stages within this classification to become more efficient. First, in [10] the authors developed a framework for method selection, through the use of genetic algorithms to learn meta-rules. A meta-rule selects a method for a given dataset based on the evaluation of the meta-features extracted from that dataset. In [30], the authors applied meta-learning in order to identify the type of task to which a dataset belongs (fake news identification, sentiment analysis, etc.), by extracting a group of 72 meta-

features including those used in [10]. Later, in [31] the authors developed a work focused on obtaining the most appropriate textual representation for a dataset by extracting the same group of meta-features used in [30]. The textual representation was determined from that representation that worked well for datasets of similar tasks, the associations of textual representations with tasks were learned in previous training. As the culmination and union of the works developed in [30, 31], the authors in [32] presented AutoText, a framework that was developed with the purpose of automating the entire classification process. This framework, based on a given dataset, searched for the most appropriate textual representation for that dataset. This representation was the input for the AutoSklearn [28] framework, which was in charge of finding, training, and evaluating the classification method more appropriate. In [33] the authors presented a new sequence for the classification process, with an extension in the pre-processing stage in order to obtain an optimal textual representation for a dataset from three steps. First, a new representation is obtained from the calculation of a set of meta-features based on distances (distance from a sample to the centroids of the classes, distance from a sample to its k nearest neighbors belonging to other classes, etc.) of a previously obtained *tf-idf* representation. Second, a sparsification is applied to the new representation in order to remove noise that can affect the quality of the samples. And as the last step, selective sampling is applied to rule out conflicting samples that could affect training. This new sequence allows classification methods to take better advantage of the information provided for their training.

In more recent years, works have been presented that have a larger complexity because they cover a large number of stages of the classification process, including different pre-processing and classification methods. In [34] the authors developed a system called AutoKeras, which performs a neural architecture search (NAS) in a much more efficient way than its predecessors. This system uses Bayesian optimization to drive network morphism, which is responsible for maintaining network functionality while its architecture is changing, thus avoiding the large computational cost of NAS methods back then. Although AutoKeras was not developed primarily to deal with text, it does have an internal embedding to deal with this type of data and make configurations regarding the maximum size of the vocabulary, the text vectorization method, or using a pre-trained word embedding such as *word2vec* [35], *GloVe* [36] or *fastText* [37], among others.

In [38] the authors presented AutoGluon, a system that is based on classifier ensembles and neural network architecture, instead of searching over a wide space for the best method and its best configuration. The system has a neural network architecture, in which each layer is made up of an assembly of methods, the outputs of these methods are concatenated and become the input of the next layer, this technique allows better use of the time of training. In terms of text classification, the system is only responsible for fine-tuning methods based on transformer neural networks like BERT [39], ALBERT [40] and ELECTRA [41].

In [42] the authors presented TextBrew, this system is in charge of finding the best transformer-based methods for a given dataset, and creates a soft voting ensemble with the union of these. The transformer-based methods that are considered are variants of ALBERT, BERT, and XLNET [43]. The number of methods that the ensemble can contain will depend on those that could be trained during a certain time given by the user. An internal meta-model is in charge of determining the best methods for a dataset, this meta-model is a soft voting ensemble, where the input data for it are the accuracies obtained by a Naïve Bayes Multinomial classifier and an ALBERT variant, and the number of documents of such a dataset.

The works presented above attack various stages of the general process of text classification, but in most of these, there is no generalization of the method selection process. These works allow an optimization that goes according to a single dataset that belongs to a single problem. In addition, these are works that are clearly more focused on pre-processing stages, such as obtaining adequate or optimal textual representations. In the case of the AutoKeras, AutoGluon, and TextBrew systems, they have the limitation of only contemplating methods based on DL, without taking into account that the ML methods are still superior in certain problems or tasks [44, 45].

In recent years, selection hyper-heuristics have been popularized to such an extent that they can be applied to various problems. For example, in [46], for tackling bi-objective 2D bin packing problems. In this work, a hyper-heuristic is represented by blocks of the condition-action form, where the condition represents the current state of the problem and the action determines which heuristics to apply. In this case, two evolutionary crossover operators were designed: the first to exchange complete blocks between two selected individuals (block-level crossover); the second to exchange internal parts of blocks (internal-level crossover), where the individuals, the blocks, and their internal parts are randomly selected. The mutation operators were also designed according to the problem: the block-level mutation operator allows removing or adding a new block from a randomly selected individual; the second operator (internal level mutation) allows the mutation of the conditions or actions of blocks that are randomly selected. Such methodology serves as the basis for the development of the evolutionary model presented in this work.

This thesis presents a more in-depth examination of this problem that has not been fully resolved, giving significance to the relationship between meta-features and the optimal (or near to it) classification method for a given dataset. Applying the selection hyper-heuristics approach could be crucial to the generalization of method selection for text classification problems.

Chapter 2

Theoretical Framework

2.1 Genetic Algorithms

Genetic algorithms are adaptive heuristic search methods, developed in order to apply them to optimization problems (e.g., the traveling salesman problem) and to study self-adaptation in biological processes [47]. A genetic algorithm (GA) is based on Darwin's theory of evolution, which allows it to have operators that exploit and explore the space of possible solutions for a given problem. Although a GA does not occupy a large configuration, it is capable of evolving highly complex individuals (possible solutions), and thus being able to find an optimal solution or close to it. An explanation of each of the components of a GA is provided below.

2.1.1 Basics

The general process of a GA begins with the creation of an initial population (parent population) in a random or heuristic way, where each individual within it is a possible solution to the problem being treated. The parent population is involved in an iterative process, where each iteration is known as a generation. Within each generation, certain methods are applied to the population. First, the individuals of the parent population are evaluated with respect to their ability to solve the problem, in order to know which are the fittest (the best solutions to the problem in that generation). Second, the evolutionary crossover operator is applied to the parent population. This operator allows the creation of a new population, which is called the child population. Third, the evolutionary mutation operator is applied to the child population, allowing further exploration of the solution space. Fourth, the fitness of the individuals of the created population (child population) is evaluated. Fifth, once the fitness of the individuals of the child population is obtained, the parent and child populations are mixed, and the fittest individuals are selected to form the new population, which will become the new parent population. This process returns to the second step until the termination criterion is met (e.g. number of

generations or no major improvements in fitness). When the termination criterion is met, the fittest individual or individuals of the last generation are returned (in some cases the entire final population may also be returned). This general process can be seen in Fig. 2.1.

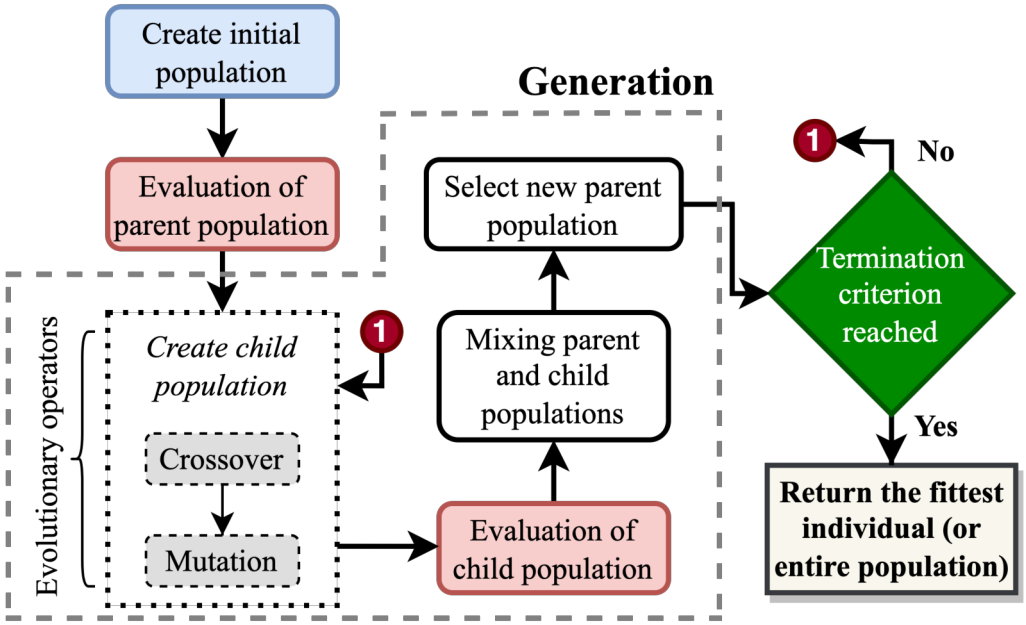

Figure 2.1: General process of a genetic algorithm.

2.1.2 Individuals

As mentioned above, GA is a computational method of biological evolution based on Darwin's theory of evolution. An individual or chromosome has two types of representations: phenotype and genotype. Within the genotype representation, each part of the chromosome is known as a gene. A gene has two properties: allele and locus. Allele is the corresponding value of the gene and locus is the location that the gene occupies within the chromosome. The phenotype representation is the observable properties of the chromosome.

To bring it to an explanation in computational terms, an individual can be represented as a string of bits (genotype representation). The genes would be each one of the bits of the string. Given a certain bit of the string, its corresponding value (0 or 1) would be the allele property and the position of that bit within the string would be the locus property. Its phenotype representation could be the conversion of the bit string to an integer or a float. This example can be seen in Fig. 2.2. Within GAs, genotype-phenotype mapping is not always required.

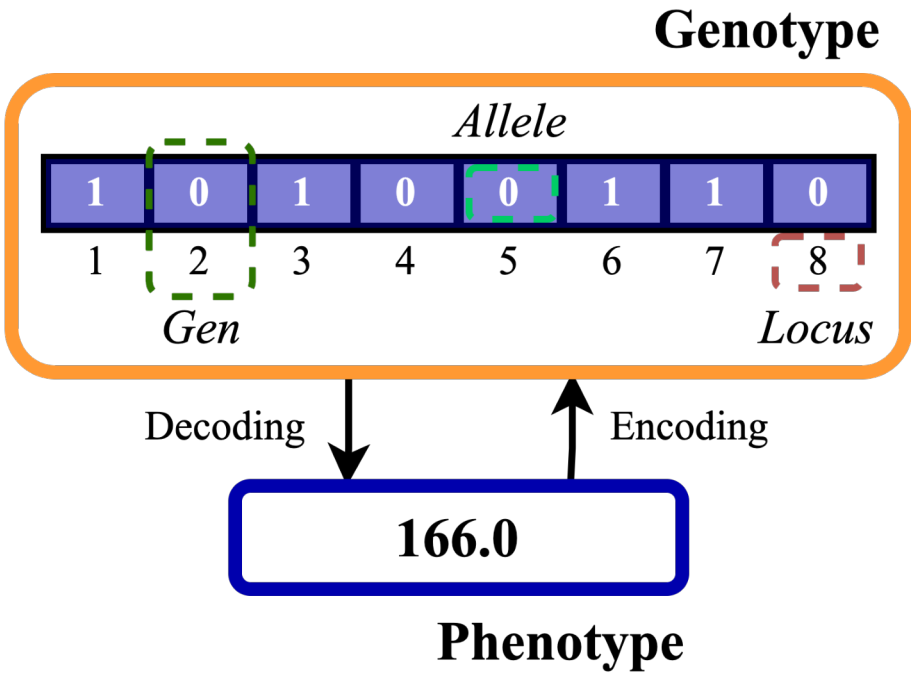

Figure 2.2: Representation of an individual.

2.1.3 Initial Population

Once the representation of the individuals has been defined, another crucial stage is the creation of the initial population. The size of this population is a parameter given by the user, and it will always be maintained across generations. There are different "simple" techniques for initializing this population, each of which causes the population to behave differently across generations and in turn, could result in different final solutions.

Some of these techniques are random initialization, random initialization with overpopulation, initialization with a single value, initialization defined by the user, and initialization with final individuals from another execution, among others. Although the initialization with final individuals from another execution allows a warm start of the GA, commonly the most used techniques are random or random initialization with over-population because the solution space can be covered widely.

2.1.4 Crossover

Crossover is an evolutionary operator that allows the genes of two or more individuals (parents) to be combined to create new individuals (children). In nature, the most common is that this process involves only two parents, in various applications

of the GAs the same thing occurs but this is not a restriction. Based on the representation of individuals in the form of bit strings, there are different operators such as single point, multiple point, or uniform crossover, examples of these can be seen in Fig. 2.3. Within the literature, according to the complexity of the problem, the adequate representation of individuals is sought. From the selected representation, the crossover operators are designed, so it is not limited to the above-mentioned crossover operators.

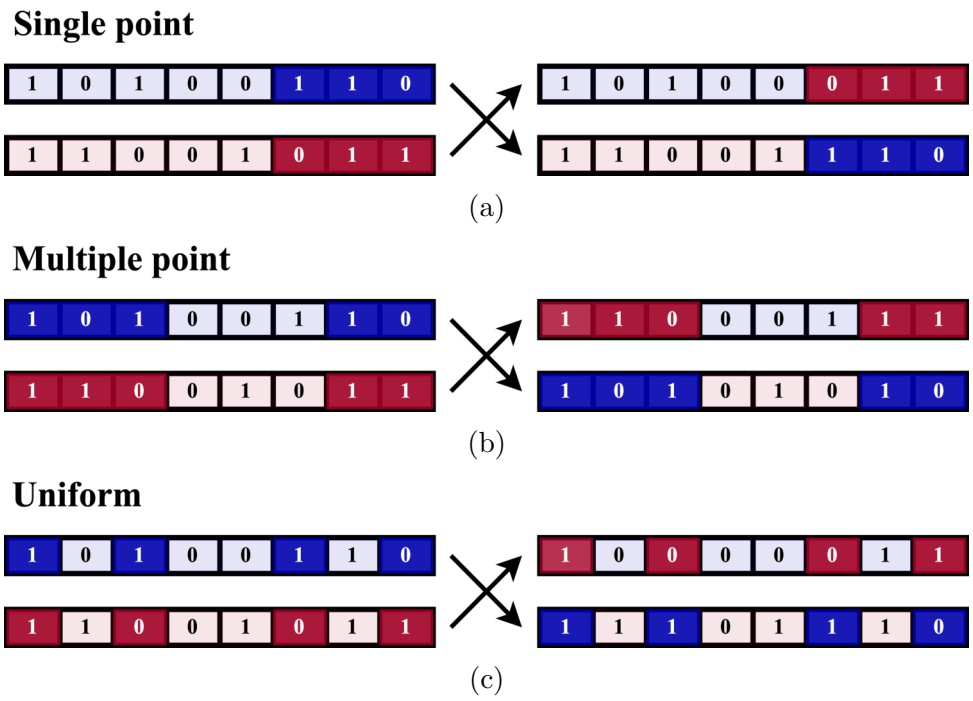

Figure 2.3: Some techniques used by an evolutionary crossover operator.

The use of this operator allows the newly created individuals (children) to be potentially fitter than their parents and thus be a better solution to the problem, since they may have inherited the best parts of them, which is why it is also known as the operator that exploits the different sectors of the solution space. Despite this, the fact that this operator is applied does not mean that the parents have to be replaced, since the children created may not be as suitable as these. This operator in some cases is accompanied by a probability, so it is not applied to the entire population.

2.1.5 Mutation

The evolutionary mutation operator is in charge of maintaining diversity in a population, this is achieved by disturbing some individuals in it. This operator is always

applied to the child population, which is created by applying the crossover operator to the parent population. In the case of representing individuals in the form of strings of bits, the evolutionary mutation operator changes the values of some of the bits of an individual, the way to select the bits that will be mutated is random in many cases, as shown in Fig. 2.4. As in the case of the crossover operator, the mutation operators are designed according to the representation of the individuals.

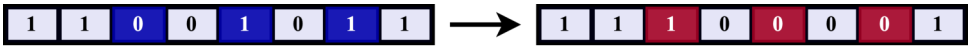

Figure 2.4: Mutation of an individual.

The selection of an individual for mutation is by probability, which is generally low. When applying this operator to an individual, the information prior to its mutation is not conserved, rather it is replaced by the new one. Unlike the crossover operator, this operator allows you to explore the wide space of solutions, where there are sectors that perhaps have not yet been known.

2.1.6 Fitness

A GA optimizes solutions according to the fitness function. In order to know how good an individual is as a solution to a problem, genotype-phenotype mapping is performed if necessary. The fitness function determines the quality of the solutions that the GA provides and also guides the search in the solution space. This function has to be designed in an efficient way so that the performance of the GA is not affected too much. Also, a very important factor is to reduce the number of calls to the fitness function when dealing with highly complex problems where each call to the fitness function is computationally expensive because the fitness function is applied to each individual of each of the populations obtained through each generation.

2.1.7 Selection

The selection operator is also known as survival selection. Since it determines which individuals survive and which die, where the fittest have a better chance of surviving. When having the father and child populations, it is necessary to make a selection of the individuals in order to direct the GA toward the optimal solution. This selection allows to preserve the fittest individuals of both populations and for this, there is a great diversity of methods, with some of them using randomness.

In its simplest form, the new population is composed of the entire child population. Some methods directly select the best individuals from the child population as parents of the next generation. Others only select a portion of the best individuals from the child population to form part of the next generation. Or in some cases, the best individuals from the child population are selected together with the individuals

from the parent population that generated them. Another method is to select the best individuals by mixing the parent and child populations.

The different methods of this operator are used in some cases to select the individuals that will function as parents in the crossover stage. For this selection of parents, there are some methods such as Proportional Selection (Roulette Wheel Selection), Linear-Rank Selection, and Tournament Selection, among others.

2.1.8 Termination Criterion

In order for the evolutionary process of a GA to end, a termination criterion must be met. This criterion is determined by the user. The most common is to use the number of generations as a criterion. There are also other criteria such as the convergence of the optimization process (when the fitness of individuals does not improve significantly), a specific time limit (not recommended when evaluating individuals is expensive), or maximum fitness achieved (the optimal solution has been found).

When the termination criterion is the convergence of the optimization process, it may be the case that the search is stuck in a local optimum and the evolutionary operators are not able to get out of that optimum. For this problem, there are "restart strategies", which help to avoid falling into this type of optimum.

2.2 Hyper-Heuristics

The term hyper-heuristic (hh, read as hyper-heuristic throughout the manuscript) is used to refer to a high-level approach, which searches over a space of low-level heuristics, allowing it to select and apply appropriate low-level heuristics to make decisions, which in turn allows giving a good solution to the problem that is being addressed [48]. There are two types of low-level heuristics: constructive and perturbative. Constructive heuristics are used in order to create an initial solution that serves as a starting point. And perturbative heuristics allow initial solutions to be improved regardless of whether they were created randomly or with a constructive heuristic.

Within various problems, a single low-level heuristic has a lower performance than a mixed and combined set of these, because the performance of each one is linked to a certain situation or instance of a problem, that is, a heuristic may provide a good or optimal solution to an a instance of the problem but perhaps not for a b instance. The hhs are in charge of finding and providing solutions (e.g. sets of heuristics), which allow a better generalization for a certain problem, these solutions may not allow for obtaining the optimal performance but they are capable of obtaining performances very close to it for a large number of instances of the problem than using a single heuristic. In short, when using the concept of hh its equivalent would be a heuristic to choose heuristics.

2.2.1 Selection Hyper-Heuristics

Hhs can be used in order to create or select low-level heuristics. Therefore, they can be categorized into four types (according to the type of low-level heuristic): selection constructive, selection perturbative, generation constructive, and generation perturbative [49]. Selection hhs (constructive or perturbative) have different methods to choose low-level heuristics such as case-based reasoning, local search methods, population-based methods, adaptive methods, and others. Although selection hhs can also be classified according to feedback method (online, offline, mixed or without learning); low-level heuristics (type, set, grouping method); the type of solution (single-point, multi-point or mixed); the type of objective (single or multi); the type of move acceptance (stochastic or non-stochastic); and parameter settings (static, dynamic, adaptive or self-adaptive) [50].

In the case of constructive selection hhs, starting from a given problem and a set of low-level constructive heuristics (depending on the problem), a constructive selection hh is responsible for providing a solution for that problem, by selecting and applying a low-level heuristic at each of the different stages of the problem.

Using a population-based method (such as GAs) for the selection of low-level heuristics, has the advantage that it allows exploring a large number of solutions at the same time, which normally have a great diversity; therefore, the space of solutions can be further explored. GAs are very flexible to be applied to various problems, and the case of selection hhs is no exception. The GAs methodology allows the conservation of the most suitable individuals (the best solutions), so that convergence can be achieved to what could be an optimal solution or close to it for a given problem.

By applying this method, each individual in the population is a set of low-level constructive heuristics selected by a selection hh; each of these individuals represents a possible solution to the problem. The performance obtained by the hhs will depend on the representation used, this representation will also determine the design of the evolutionary crossover and mutation operators. During the process of evolution, individuals are evaluated in each of the generations to know their fitness for solving one or more instances of the problem, and those that are the fittest have a better chance of surviving and moving on to the next generation. To obtain individuals that allow better generalization, it is best to evaluate them with multiple instances of the problem, since by using only one instance, the individuals will be developed specifically to provide a solution to it.

When an individual's fitness is determined by their ability to solve multiple instances of a problem, it is most common to divide these instances into two parts: training and test. The training part contains those instances of the problem with which the individuals will be evaluated and evolved during the evolutionary process. On the other hand, the instances of the test part are used to know the fitness of the individuals to solve unseen instances of the problem.

2.3 Methods for Data Pre-Processing

The data collected by the various sources of generation and storage cannot be sent directly to the classification methods, because within these data there may be values that can seriously affect the performance of the methods (e.g. irrelevant variables, missing values, among others). The pre-processing methods allow discarding this type of conflicting values, with which the quality of the datasets increases and the data within it becomes more consistent. This has a high impact on classification methods since it allows them to speed up the reading, use, and interpretation of datasets.

2.3.1 Cleaning Process

Due to the fact that a large number of sites for the interaction among users, the use of the Internet has become more common within society, which has also caused the abundant generation of data. In turn, different types of data generated on these sites can be found and collected, such as audio, images, text, and video, among others. For this work, the cleaning process is focused solely on the generated textual content. The majority of the most popular sites are developed by big companies, in which a correct analysis of their data can provide a lot of information for decision-making. Also, much of this data can be collected by scientists for research into the development of new techniques. Therefore, the cleaning step is very important as it can significantly improve the results obtained and the interpretation of these, as well as get the best out of the data provided. Within ML, this process is very relevant since the performance of the classification methods (execution time, accuracy, etc.) will depend on the quality of the data. If the data is of poor quality, rarely any classification method will provide good results, regardless of its focus.

Today various techniques developed in different programming languages allow this process to be carried out, and in general they are capable of eliminating irrelevant data, fixing structural errors, dealing with missing data, and eliminating atypical data, among others. In Fig. 2.5, a general example of the cleaning process is shown with a sample of a news item. Prior to the cleaning process (left side), it can be observed that this sample is composed mainly of English words with uppercase or lowercase letters, punctuation marks, and also alphanumeric strings, in addition, in some cases they may contain links to websites, emails, phone numbers, etc. After the cleaning process (right side), all words are kept, and all letters are lowercase, in some cases, words of too short or too long length (depending on the language) are also removed. This process is adapted according to the interest that one has in preserving the textual content depending on the task that is going to be carried out, for example in certain problems it might be necessary to preserve the capital letters or punctuations.

As mentioned above, various techniques or a set of techniques can be applied according to the problem, some of these are: convert all letters to lowercase, apply

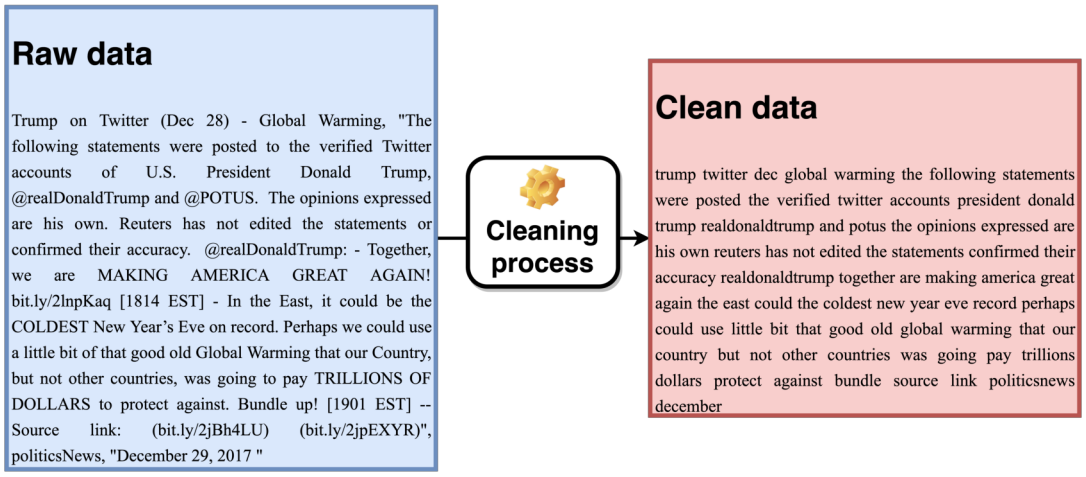

Figure 2.5: Example of the cleaning process.

tokenization, remove stop words, or remove short and large words. Converting all the letters of a text to lower case is not a technique used in many cases, since it can cause certain problems due to the different meanings that words can have when beginning with upper or lower case (capitonym), for example, August (the eight month of the year) or august (majestic or venerable). Tokenization is a technique that consists of dividing a text (or part of it) into tokens, where each token represents a sequence of characters (e.g. a word, a phrase, etc.). These tokens are created from a separator character (blank spaces, line breaks, tabs, etc.), that is, the one that allows identifying where a token begins and ends. For example, if you have the sentence "i will try to remember that" and the separator character is a white space, the resulting tokens would be ['i', 'will', 'try', 'remember', 'that']. Normally this tokenization process is done with the help of regular expressions since they allow powerful, efficient, and flexible text processing [51]. Depending on the regular expression, it could be possible to extract different features such as words, links, phone numbers, email addresses, etc.

Stop word removal is typically used when tokenization and regular expressions are not enough. Stop words are a set of meaningless words on their own, such as conjunctions, articles, prepositions, and adverbs [52]. Despite the fact that these words are the most used in the texts, the fact of removing them does not substantially affect the meaning of the sentences. Clearly, the set of stop words depends on the language, some examples of stop words from the English language are 'me', 'my', 'our', 'being', and 'having'. The NLTK (Natural Language Toolkit) [53] library allows access to more than 25 sets of stop words from different languages, which facilitates the process of removing them. In addition, NLTK has tools for tokenization, lemmatization, labeling, parsing, and semantic reasoning, among others.

2.3.2 Transformation Process

Once the cleaning process has been applied to the data, it is necessary to perform a transformation process on it, in order to convert it into a suitable representation for ML classification methods. In general, any of these classification methods (such as k-Nearest Neighbors, Support Vector Machines, Decision Trees, etc.), work with arrays of numbers as input data, that is, tabular data. Since this work is focused on working with textual content (non-tabular data), it is necessary to convert the data to a suitable representation for the classification methods. To achieve this transformation, there are various methods such as *tf-idf* [54], fastText, word2Vec, gloVe, etc. Some of these may have advantages over others, always depending more on the type of problem than on the classification method. For this work, only the tf-idf method is used.

tf-idf

Despite being proposed many years ago with respect to fastText, word2Vec, and gloVe, the *term frequency-inverse document frequency* or just tf-idf method is still very popular and used in various problems, allowing to obtain superior or competitive results against others. There are documents in which the frequencies of words of little significance are very high, and if this count goes directly to the classification methods, the words with the least frequency (which may be the most significant and interesting for the distinction of documents) would be overshadowed, and would have no relevance to the methods. Tf-idf is a method that allows calculating how relevant a word turns out to be within a document in a collection of documents, that is, words with very high frequencies and that appear in multiple documents do not turn out to be very relevant compared to those that they appear one or multiple times in a single document. This method is summarized in Equation 2.1.

$$tf\text{-}idf(t,d) = tf(t,d) \times idf(t) \tag{2.1}$$

The first factor $tf(t,d)$ calculates the frequency of occurrence of a term t in a document d for all documents in a collection, although an adjustment is commonly made due to the variability that may exist between the lengths of the documents in the collection. For example, this adjustment can be done by dividing the frequency $tf(t,d)$ by the length of the document d or by the frequency of the largest term found in the document d. There are also established methods for normalizing these frequencies such as 'boosted normalized term frequency' [54].

The second factor called inverse document-frequency or $idf(t)$ helps with the problem of large frequencies of terms across multiple documents by giving more relevance to terms found in fewer documents in the collection. This factor directly depends on the collection of documents, as can be seen in Equation 2.2.

$$idf(t) = \log \frac{N}{df(t)} \tag{2.2}$$

where N is the total number of documents in the collection, and $df(t)$ is the number of documents that contain the term t.

The result of this method is a matrix of type term-document. An illustration of how this method works can be seen in Fig. 2.6. First, a term-document matrix is calculated according to the $tf(t,d)$ method, then the $idf(t)$ part is calculated, in order to know the importance of each term within the entire collection of documents. The result can be represented as a diagonal matrix, where each of the elements on the diagonal represents the importance of a term. Finally, the product of tf by idf is computed. This results in a term-document matrix, but now each element of the matrix represents a tf-idf value. The rows represent each of the documents and the columns the terms.

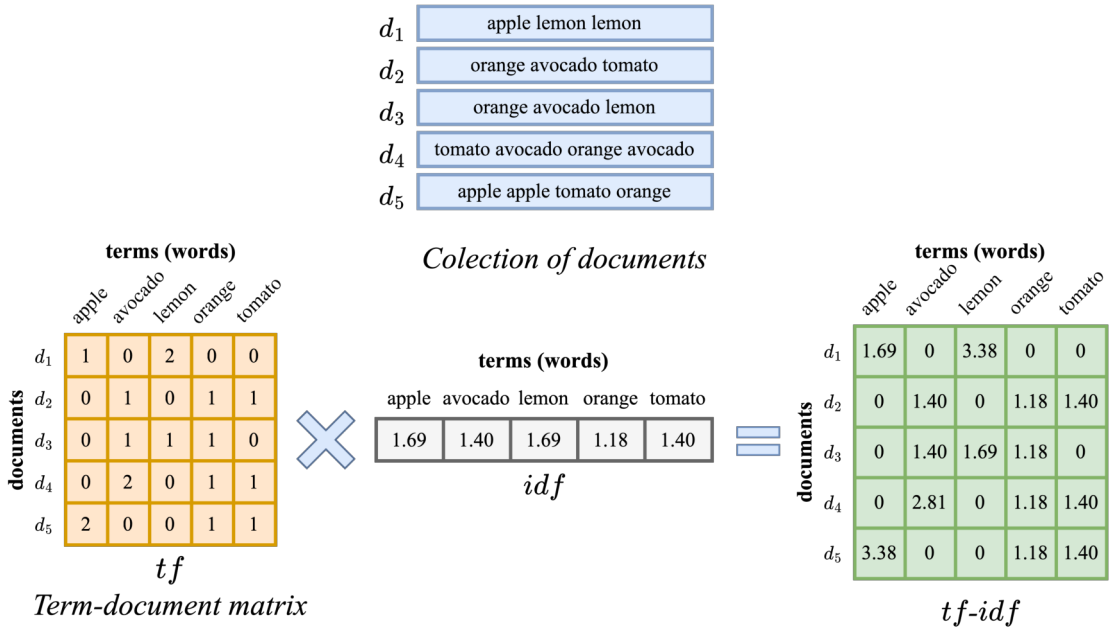

Figure 2.6: Representation of obtaining a term-document matrix with the tf-idf method.

2.3.3 Normalization Process

Normalization is a process that should always be applied prior to training the classification methods, since this allows the characteristics to be in the same range of

values, avoiding problems with scales and units, and considered with the same importance by the classification method. Furthermore, the performance and stability of the classification methods during training are generally improved.

There are various techniques for data normalization, and some of the most common are Z-Score, Log Scaling, Min-Max, Feature Clipping, Max-Norm, L1-Norm, and L2-Norm. Normalization using L2-Norm allows transforming a vector $\mathbf{x}$ into a unit vector $\hat{\mathbf{x}}$, that is, the sum of the squares of the n-components $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n)$ is equal to 1. This normalization is also known as Euclidean Norm because, within the normalization process of a vector, it is necessary to calculate its magnitude, which is the same equation as in the calculation of a Euclidean distance from a point p to the origin, as can be seen in Eq. 2.3.

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{2.3}$$

Therefore, the L2 or Euclidean normalization for a vector $\mathbf{x}$ is defined by Eq. 2.4

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|_2} \tag{2.4}$$

where $\hat{\mathbf{x}}$ is the normalized vector. Within text classification, this process is applied to a term-document matrix, where each row of the matrix is a vector and the value associated with each column are the components of that vector.

2.4 Machine Learning and Deep Learning Methods

ML is an area that is focused on the ability to learn through experience. There are four main categories within ML, which are: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Within each of these categories there are various learning algorithms, which are fed with experience in the form of data, in order to build computational models that are capable of recognizing the different patterns found in the data provided, and therefore be able to make correct predictions. Being so that many applications today work in an intelligent way thanks to the extensive advances in ML.

For its part, DL is a broad field of study within ML, which has been popularized in recent years. The various DL classification methods for supervised learning are based on artificial neural networks. Neural networks are able to identify various patterns that some cases are too complex, which allows extracting a large number of features, but at a large computational cost unlike other more simple ML methods, in addition to requiring a large amount of data to perhaps generate good learning.

Supervised learning is a technique used in order to be able to learn a mapping function or a set of weights and thus assign an output value (category) to an input

(sample) based on a training set made up of input-output pairs (sample-category). It is during the learning or training process that such a function or set of weights is learned. Later, the model is tested with samples not seen during the training process, in order to know its generalization capacity. When a model performs very well within a set of unseen samples it is known as a well-generalized model.

The classification methods used in this work belong to the category of supervised learning. Likewise, the datasets used consist of samples in document-category form. A brief description of the classification methods used is given below.

2.4.1 Multinomial Naïve Bayes

Bayes' classification methods are methods based on Bayes' probabilistic Theorem. This theorem allows us to know the probability of an event, taking into account the previous knowledge related to such event. This theorem is expressed in Eq. 2.5, where $P(A)$ and $P(B)$ are the probabilities that the events A and B occur without any conditions; and the conditional probabilities $P(A|B)$ and $P(B|A)$ are those that determine the probabilities of an event A occurring given an event B and for an event B to occur given an event A, respectively.

$$P(A|B) = \frac{P(A|B)\,P(A)}{P(B)} \tag{2.5}$$

In terms of classification it can be expressed as in Eq. 2.6, where there is a sample $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ in the form of an n-dimensional vector, and a set of k-classes $\mathbf{C} = \{c_1, c_2, \ldots, c_k\}$. $P(c_i|\mathbf{x})$ determines the probability that the sample $\mathbf{x}$ belongs to a class c_i; $P(x_j|c_i)$ is the probability of occurrence of a feature x_j in a sample belonging to a class c_i; and $P(c_i)$ is the prior probability of the class c_i. The denominator found in the right-hand factor in Eq. 2.5 is omitted in this case, since it only represents a constant.

$$P(c_i|\mathbf{x}) \propto P(c_i) \prod_{j=1}^{n} P(x_j|c_i) \tag{2.6}$$

In such a way that to determine to which class a sample belongs $\mathbf{x}$ is calculated:

$$\arg\max_i P(c_i|\mathbf{x}) \tag{2.7}$$

Multinomial Naïve Bayes (MNB) is a very popular method within text classification. From a term-document matrix representation, a document (sample) is represented by a vector with numeric values, this method of classification treats the values of such a vector as a multinomial distribution. Thus, it allows converting to a linear classification method by passing to a logarithm space as shown in Eq. 2.6, which also changes the way in which likelihood is calculated $P(x_j|c_i)$ of a document, as seen in Eq. 2.8. Where N_{ij} is the number of times that the feature

j appears in the documents of a class i; N_i is the number of occurrences of class i; α_j is a smoothing prior for a feature j and α is the sum of α_i; $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$; $\mathbf{w}_i = \{w_{i1}, w_{i2}, \ldots, w_{in}\}$. See [55, 56] for a more detailed definition of this method.

$$\begin{aligned} \log P(c_i|\mathbf{x}) &\propto \log\left(P(c_i)\prod_{j=1}^{n} P(x_j|c_i)\right) \\ &= \log P(c_i) + \sum_{j=1}^{n} x_j \log\frac{N_{ij}+\alpha_j}{N_i+\alpha} \\ &= \log P(c_i) + \mathbf{w}_i^T\mathbf{x} \end{aligned} \tag{2.8}$$

where $w_{ij} = \log\frac{N_{ij}+\alpha_j}{N_i+\alpha}$. So the MNB classification method summarizes the maximum argument of Eq. 2.8, as shown in Eq. 2.9.

$$\arg\max_i \left[\log P(c_i) + \mathbf{w}_i^T\mathbf{x}\right] \tag{2.9}$$

2.4.2 Complement Naïve Bayes

Complement Naïve Bayes (CNB) [55] is a classification method developed to address the systematic errors of MNB that prevent better performance. This method has a different way to calculate the weights unlike MNB (Eq. 2.8), since instead of using the data of a c_i class, it uses the data of the other classes except for this one. This allows for dealing with problems where there is an imbalance in the classes. In such a way that taking Eq. 2.8 as a starting point, the second logarithm on the right side and the sign associated with it are rewritten as shown in Eq. 2.10.

$$\log P(c_i|\mathbf{x}) = \log P(c_i) - \sum_{j=1}^{n} x_j \log\frac{N_{\tilde{i}j}+\alpha_j}{N_{\tilde{i}}+\alpha} \tag{2.10}$$

where $N_{\tilde{i}j}$ corresponds to the number of times a feature j occurs in documents of all classes except class i, and $N_{\tilde{i}}$ is the number of occurrences of all classes except class i, α and α_j are calculated in the same way as in Eq. 2.8.

Another conflict within MNB is the erroneous production of different magnitude classification weights, which are mainly caused by the assumption of independence between features, which causes the weights to bias towards a particular class. The way in which CNB solves these problems is through normalization of these weights $w_i = \frac{w_{ij}}{\sum_j w_{ij}}$. Finally, CNB can be represented by the classification rule shown in Eq. 2.11. x_j represents the value of feature j of a vector $\mathbf{x}$, and w_{ij} is the weight of class i for feature j.

$$\arg\min_i \left[\sum_j^n x_j w_{ij}\right] \tag{2.11}$$

2.4.3 Bernoulli Naïve Bayes

The Bernoulli Naïve Bayes (BNB) classification method, as its name suggests, uses Bernoulli multivariate distributions when making use of the data for training or classification. That is, a document is represented by a vector made up of ones and 0s, in which a 1 represents the presence of the feature in the document and a 0 the absence of it. Therefore, BNB does not consider the number of occurrences of a feature within a document, unlike MNB and CNB. Thus, the probability of a document $\mathbf{x}$ given a class c_i or (likelihood) is determined differently from MNB and CNB as can be seen in Eq. 2.12.

$$P\left(c_i|\mathbf{x}\right) \propto P\left(c_i\right) \prod_{j=1}^{n} x_j P\left(x_j|c_i\right) + \left(1 - x_j\right)\left(1 - P\left(x_j|c_i\right)\right) \tag{2.12}$$

Similar to the other Naïve Bayes classification methods, logarithms are applied in order to deal with problems where the dimensionality is very large. Likewise, BNB uses a Laplacian prior to compute each $P\left(x_j|c_i\right)$. Therefore, the BNB classification rule is shown in Eq. 2.13. A further breakdown of this classification method is presented in [55, 57].

$$\arg\max_i \left[\log P\left(c_i\right) + \sum_j^n \log\left(x_j \frac{1 + N_{ij}}{2 + N_i} + \left(1 - x_j\right)\left(\frac{1 + N_{ij}}{2 + N_i}\right)\right)\right] \tag{2.13}$$

2.4.4 *k*-Nearest Neighbors

In the instance-based classification method approach, the one that always appears first is the k-Nearest Neighbors (KNN) [58] classification method. KNN is an instance-based method since it does not create a model, rather, it stores instances (to be used as neighbors) of the documents given in the document-category form to later make use of this information during the stage of tests. In an n-dimensional space, a test document $\mathbf{x}$ is classified by a vote of its k-nearest neighbors (k is a user-supplied parameter, usually of odd value), that is, the category assigned to this document will be the one that is in the majority among these neighbors, as can be seen in Fig. 2.7.

There are different methods to find the k-neighbors, some of them optimize the search when dealing with large numbers of instances, but there are also different metrics for calculating the distances that will allow to identify which are the closest, such as: *Minkowski*, *Manhattan*, *Euclidean*, *Cosine Similarity*, *Mahalanobis*, among others. In each of these metrics, the way to calculate the distances varies with respect to the others, this is exemplified in Fig. 2.8. Thus, each one can provide different results for the same problem.

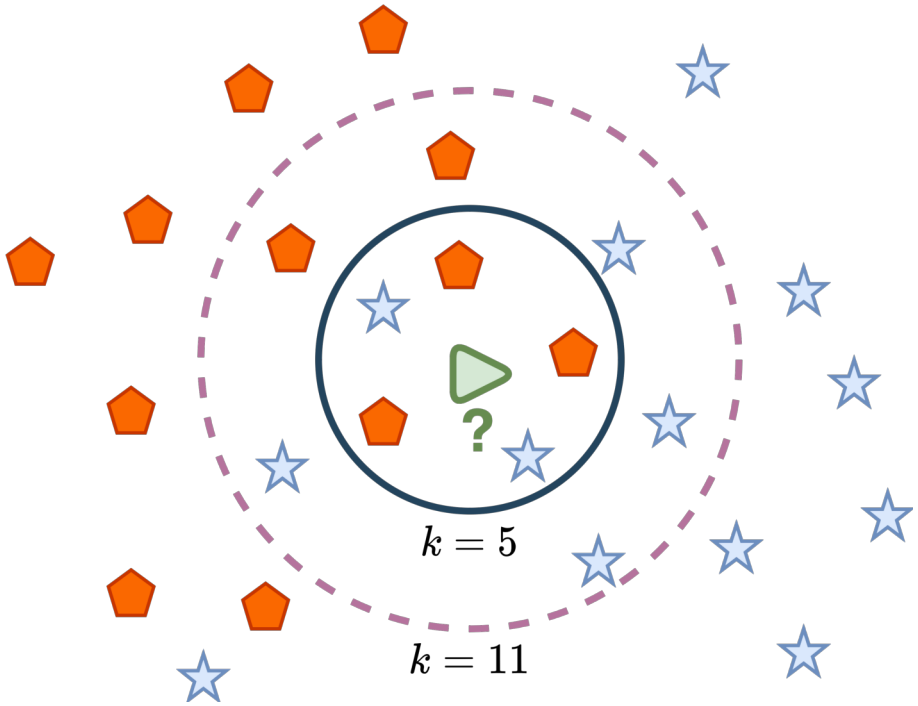

Figure 2.7: k-Nearest Neighbors classification example for $k = 5$ (the assigned category will be 'pentagon') and $k = 11$ (the assigned category will be 'star').

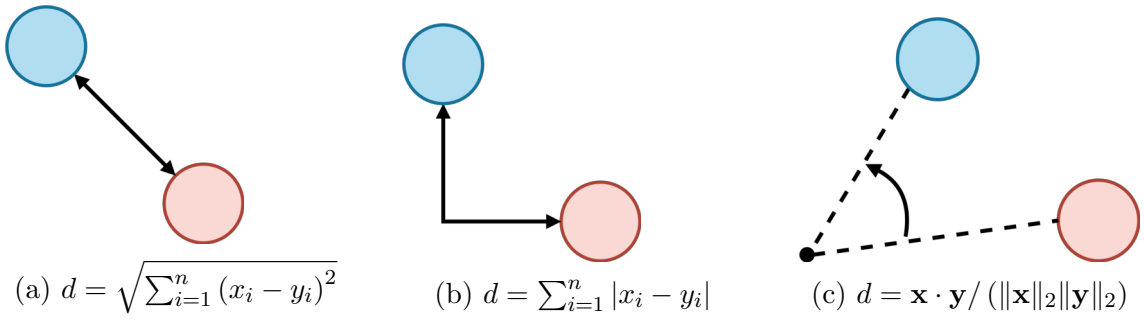

Figure 2.8: Distance between two points ($\mathbf{x}$, $\mathbf{y}$) using different metrics: (a) *Euclidean*, (b) *Manhattan* and (c) *Cosine Similarity*

2.4.5 Decision Tree

A Decision Tree (DT) is a non-parametric classification method, which, during the training stage, seeks to create a set of decision rules based on the features of the input documents. Thus, DT can decompose complex decisions into much simpler ones, and thus determine the difference between documents of a class A and documents of a class B by means of a subset of features. Although there are also cases in which

a DT can develop a very complex structure, which will not allow it to have a good generalization when classifying unseen documents.

In general, a DT is composed of a root node, internal nodes and leaves (terminal nodes), and has the structure of a binary or non-binary tree, see Fig. 2.9. Each of the internal nodes, as well as the root node, are defined by a node-specific subset of classes $c(i)$, feature subset $f(i)$, and decision rule [59, 60] $d(i)$. Therefore, the process of classifying a document can be understood as a hierarchical classification. This means that, when classifying a document, it goes through stages in which the number of possible categories decreases as it advances on the DT until it reaches a terminal node, which only contains the category to which the document belongs. When a tree is fully developed, the predicted probability for a document that can be classified into k-categories is 1 for one category and 0 for the others, since all terminal nodes are pure. When it is the opposite, the predicted probability for each category can be a number between 1 and 0, and the document is classified with the category with the highest probability.

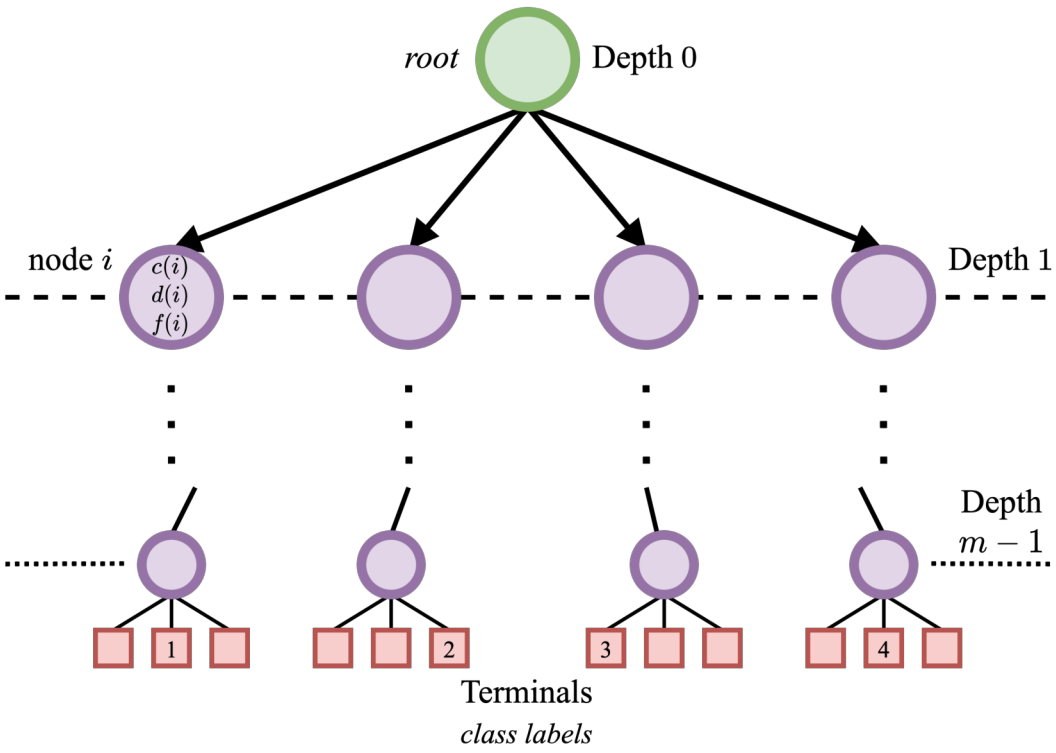

Figure 2.9: Representation of the structure of a DT.

2.4.6 Logistic Regression

The Logistic Regression (LR) classification method is a method that allows describing the probability that a document $\mathbf{x}$ belongs to a category c_1 through the use of the logistic function. This probability can be defined as a conditional probability, as shown in Eq. 2.14. The parameters α and β are calculated with the data provided during the training stage. Therefore, $1-P_{c_1}(\mathbf{x})$ is the probability that the document $\mathbf{x}$ belongs to a category c_2.

$$P_{c_1}(\mathbf{x}) = \frac{1}{1 + e^{-(\alpha+\beta\mathbf{x})}} \tag{2.14}$$

When dealing with a non-binary classification problem, a methodology known as Multinomial Logistic Regression [61] is used, and the probability that a document $\mathbf{x}$ belongs to a class c_i, where there are K possible classes, is determined by Eq. 2.15. Thus, each $\mathbf{w_i}$ is a vector of weights for a category i, and b_i is a bias.

$$P_{c_i}(\mathbf{x}) = \frac{\exp(\mathbf{w}_i \cdot \mathbf{x} + b_i)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)} \tag{2.15}$$

In order to find the optimal weight vectors for each category, the problem is approached as an optimization problem given an objective function. And also, in order to avoid overfitting, the addition of a regularization term would be the solution. This is represented in Eq 2.16, where m is the number of samples provided for training, $\mathbf{w}$ is the vector of weights that allows maximizing the log probability, and $R(\mathbf{w})$ is the regularization term.

$$\arg\max_{\mathbf{w}} \sum_{i=1}^{m} \log P\left(y^{(i)}|\mathbf{x}^{(i)}\right) - \alpha R(\mathbf{w}) \tag{2.16}$$

Commonly, the term $R(\mathbf{w})$ is computed by L1 or L2 regularizations. In case of applying L1 regularization, the term would look like:

$$R(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{j=1}^{n} |w_j|$$

and for L2 regularization, the term is computed as follows:

$$R(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{j=1}^{n} w_j^2$$

In both regularizations, n corresponds to the size of the vector $\mathbf{w}$.

2.4.7 Support Vector Machines

Like LR, Support Vector Machines (SVM) is a feature-focused classification method, but in this case, SVM tries to find the best hyperplane in feature space that allows it to separate instances according to their category [62]. An instance (document) is represented as a point in an n-dimensional space (n corresponds to the number of features). In such a way that the hyperplane learned through the documents provided in a training stage, allows dividing the space into two subspaces, so that in each subspace a single category dominates. The way the feature space is partitioned is defined by Eq. 2.17. Therefore, $\phi(\mathbf{x})$ denotes a feature space transformation applied to the document $\mathbf{x}$, which can be categorized into one of two categories. $\mathbf{y} \in \{1, -1\}$. The transformation is commonly performed when there is no hyperplane that can admissibly separate the documents in the original space. The parameters $\mathbf{w}$ (normal vector, which controls the direction of the hyperplane) and b (bias, the distance from the origin to the hyperplane), are adjusted during the learning process. The category assigned to an unseen document will depend on the sign resulting from applying Eq. 2.17, allowing to know if it is above or below the hyperplane.

$$\mathbf{w}^T \phi(\mathbf{x}) + b = 0 \tag{2.17}$$

The closest points to the hyperplane are known as the support vectors, and the smallest distance from this hyperplane to the support vectors is called *margin*. Therefore, it is vital to find the maximum-margin hyperplane, since with it the components $\mathbf{w}$ and b can be obtained, which in turn achieve the minimum generalization error of all possible hyperplanes. The way to solve this optimization problem can be in its primal or dual [63] form, the latter represented by Eq. 2.18. Where each document $\mathbf{x}_i$ corresponds to a category $\mathbf{y}_i$ of a set of m documents. Such an equation is subject to: $\sum_{i=1}^{m} \alpha_i \mathbf{y_i} = 0$, $\alpha_i \geqslant 0$, $i = 1, 2, \ldots, m$.

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \mathbf{y}_i \mathbf{y}_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \tag{2.18}$$

Since space cannot be perfectly separated, either linearly or using a kernel, the SVM can be allowed to make some minimal errors (see Figure 2.10), this method is known as *soft margin*. This method performs an addition of a regularization parameter C and a loss function (e.g. hinge loss, exponential loss, or logistic loss). Using the loss function *hinge*, the dual problem (Eq. 2.18) becomes bound by: $\sum_{i=1}^{m} \alpha_i \mathbf{y_i} = 0$, $0 \leqslant \alpha_i \leqslant C$, $i = 1, 2, \ldots, m$. Where the only difference is the limits set for α_i.

2.4.8 BERT

Bidirectional Encoder Representations from Transformers (BERT) is one of the most popular DL methods for the text classification task, based primarily on one of the Transformers components. A Transformer is based on self-attention mechanisms,

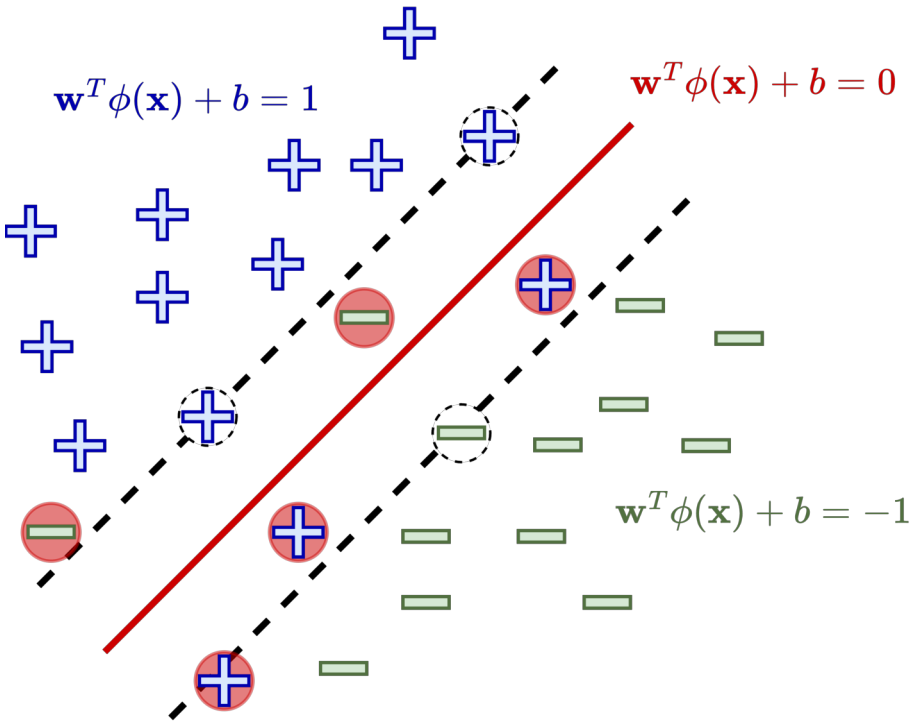

Figure 2.10: Representation of a Soft Margin Support Vector Machine.

which allows it to give greater importance to the context according to the input information. In general, a Transformer is made up of two components *encoding* and *decoding*. Both the *encoding* and *decoding* components consist of stacks of *encoders* and *decoders*, respectively. Each *encoder* is composed of a self-attention layer and a Feed Forward Neural Network. Instead, each *decoder* is composed of a self-attention layer, an encoder-decoder attention layer, and a Feed Forward Neural Network. The middle layer of the *decoders* allows them to give more importance to the parts that are relevant.

BERT is a method that only uses the *encoding* component of a Transformer, as shown in Fig. 2.11. This stack consists of *n-encoders* (12 for the Base version and 24 for the Large version). When this method is used to perform text classification tasks, it is most common to use methods that have been pre-trained with large amounts of datasets. The input data consists first of a special *CLS* token (to specify use as a classifier) and followed by a sequence of words, and this entire sequence is passed through each of the *encoders*, where each apply two processes: first, a layer of self-attention; the output of such a layer is passed to a feed-forward neural network, and these results from each *encoder* are passed to the next in the stack. Finally, the first position (a numeric vector) of the top encoder is passed as input data to a feed-forward neural network of a layer plus softmax, thus completing the classifier. With such an architecture, BERT positioned itself as one of the state-of-the-art

methods in various NLP tasks, surpassing performances in multiple benchmarks such as GLUE [64], SQuAD v1.1 [65], SQuAD v2.0 (extension of SQuAD v1.1) and SWAG [66].

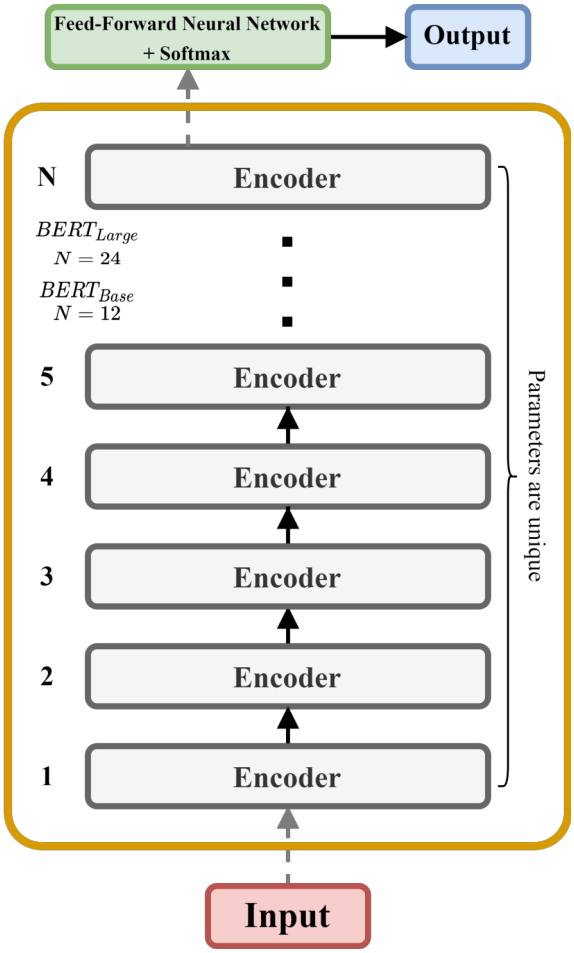

Figure 2.11: BERT's architecture.

2.4.9 ALBERT

A Lite BERT (ALBERT) was developed in order to reduce the computational costs during the training of its predecessor BERT, this was achieved by using different techniques to reduce the number of parameters, which allowed it to achieve lower consumption of memory and a considerable increase in training speed. In summary, there were three improvements: 1) Factorized embedding parameterization, 2) Cross-layer parameter sharing, and 3) Inter-sentence coherence loss.

First, a *Factorization of the Embedding matrix* was performed, so that the size of the input embedding layer and the hidden layer were different (in BERT they

are identical) since the input embedding layers learn from context-independent information, and the hidden embedding layers from context-dependent information. This allows a great reduction of about 80% of the original number of parameters. Second, to improve the efficiency of the parameters, the *Cross-layer parameter sharing* method was proposed, which shares all the parameters in the *n-encoders*, which, in addition to reducing the number of parameters, increases the regularization of ALBERT. Finally, both BERT and ALBERT use the loss of masked language modeling during training. But in addition, each one additionally uses different losses, next-sentence prediction for BERT and sentence-order prediction for ALBERT, this change was made because next-sentence prediction presented certain inefficiencies when predicting topics, and instead, sentence-order prediction worked with consistency between sentences.

With these improvements, it was able to surpass the performances of BERT and similar to it in benchmarks such as GLUE, and SQuAD. The architecture of ALBERT is very similar to that of BERT (Fig. 2.11), with the difference that the parameters are the same in each *encoder* of ALBERT, as mentioned above and as is specified in Fig. 2.12.

In general, DL methods such as BERT and ALBERT have become too popular, obtaining great results in different types of tasks, but one of their main disadvantages is the excessive use of computational resources. Although there are pre-trained models, they still require large resources for their respective fine-tuning. In general, both classic ML and DL methods have their respective advantages in computing costs, time, performance, etc. This will always depend on the problem being addressed, as a simple example: when it comes to text classification and the number of documents in a dataset is small, a ML method may be the most appropriate. If not, a DL method could be superior, as long as only classification performance matters and not the computational cost.

2.5 Evaluation Setup

2.5.1 Dataset Split

The most adequate way to validate the generalization of a classification method that has been previously trained with a dataset is to evaluate it with a set of documents that have not been seen during training. There are various validation methods that allow this to be carried out, where the use of each will depend mainly on the number of documents contained in the dataset being used.

The method used in this work is train-test split, which allows dividing a dataset into two subsets, where the one with the largest number of documents is used for the training stage of the classification method and the second for the validation of the method. Normally, the proportions of documents are handled in percentages of 60%-80% for the training subset, and the rest for the test subset. Under this

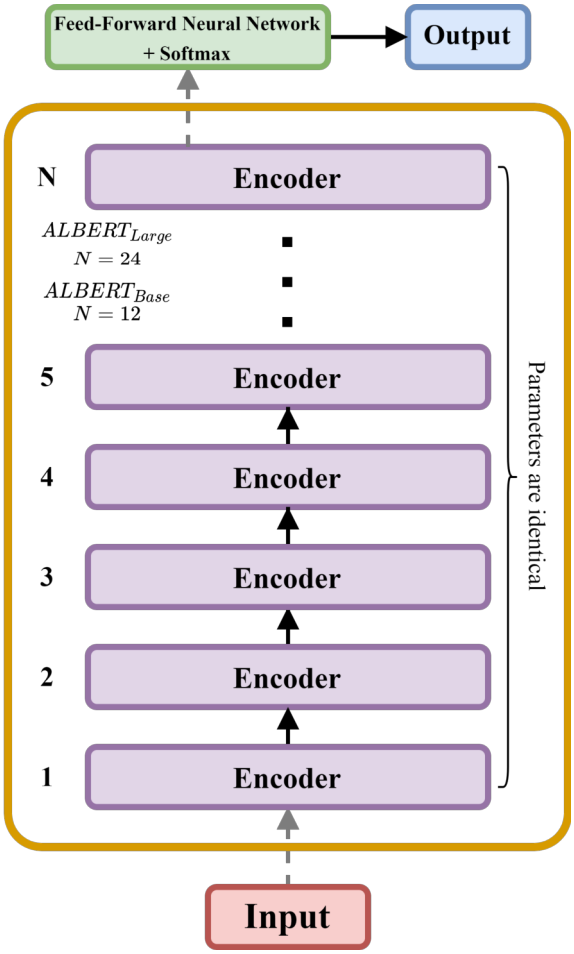

Figure 2.12: ALBERT's architecture.

method, there are cases in which three subsets training, validation, and test are created, in order to use the training subset to train various classification methods with different values in the hyperparameters and evaluate their performances with the subset of validation, in order to find the best values of its hyperparameters, and finally be tested with the test subset. This allows knowing a brief estimate of how the classification method would perform when tested with real-world problems, with any of the two techniques (train-test, train-validation-test).

Other types of methods are those of cross-validation, which are used with the objective of analyzing the multiple versions of a classification method, which are trained with different training sets, allowing finding a set that allows obtaining a classification method with a good generalization. One of the most popular cross-validation methods is k-fold, this method divides the original dataset into k subsets, of which $k-1$ are selected to train a classification method, and the remaining subset to validate such a method. This process is performed k times, in which the subset

to validate the classification method is different each time. The performance of the classification method when using k-fold is averaging the sum of the performances obtained on each occasion. The value of k is determined by the user, and typically values like 5, 10, or 20 are used.

In all validation methods, care must be taken when performing the splits when dealing with datasets with multiple categories and in some cases such categories are unbalanced (many documents for some categories and few for others). Since it may happen that some of the subsets created may not contain documents from any of the categories that appear in the original dataset. This problem can be avoided by applying stratification which makes sure that documents from all the categories of the original dataset will appear in each subset.

2.5.2 Evaluation Metrics

When a classification method is tested with a set of documents not seen during the training stage, it is necessary to use an evaluation metric to know how well the method performs when categorizing such documents. Each of the metrics provides a score that allows knowing the generalization that the classification method has achieved, evaluating it from different perspectives. Therefore, they can also be used in order to select the best classification method within a set of methods, or other types of cases [67].

A large number of the evaluation metrics are defined from a confusion matrix, which simplifies the understanding of the performance of a classification method by visualizing the number of occasions in which the classification method predicted the category correctly (true positives and negatives) or incorrectly (false positives and negatives). This matrix is built from the actual categories and those predicted by the method, as shown in Fig. 2.13. The true positives (TP) and the true negatives (TN) are the counts of occasions in which the classification method assigned the correct category to a certain number of documents. Conversely, false positives (FP) and false negatives (FN) are the counts in which the classification method did not assign the correct categories.

Two of the most common evaluation metrics are *Accuracy* and *F1*. Being one of the most used metrics in practice due to its ease of calculation and understanding, accuracy is defined as the percentage of hits that the classification method had in the entire test set, and it is defined as in Eq. 2.19. The problem with this metric is when the test set suffers from unbalanced categories, since the results provided may not correctly reveal the performance of the classification method in each of the categories. For example, there is a dataset made up of 90 documents of category A and 10 documents of category B; and the classification method is capable of correctly categorizing all documents from category A but none from category B, an accuracy of 0.90 would be obtained, which is a very good value, but it is provided because there is a category that dominates in the dataset, and the classification

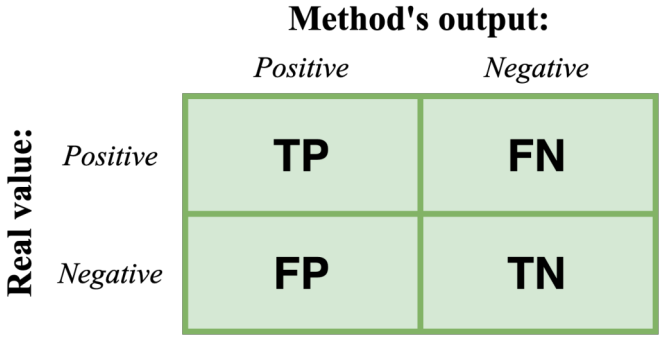

Figure 2.13: A confusion matrix for a binary classification problem.

method really does not work.

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{2.19}$$

The F1 evaluation metric becomes a great alternative to avoid the problem of imbalanced datasets, as it is made up of the *Precision* and *Recall* metrics. Precision is a metric that allows knowing the percentage of everything that was categorized as positive by the classification method and that really is (Eq. 2.20). And the Recall metric instead calculates from the total number of documents of the positive class, how many hits the classification method had (Eq. 2.21).

$$Precision = \frac{TP}{TP + FP} \tag{2.20}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.21}$$

F1 determines the average between Precision and Recall, as shown in Eq. 2.22. Therefore, a high value of F1 means that the Precision and Recall are high, and vice versa.

$$F1 = \frac{2 \cdot Precison \cdot Recall}{Precision + Recall} \tag{2.22}$$

Both evaluation metrics provide values between 0 (completely incorrect classification) and 1 (perfect classification). In addition, F1 has two forms, *macro* and *micro*, which are used when the datasets have more than two categories. In summary, the macro form calculates each metric for each class, that is, it creates a confusion matrix for each of the classes, and at the end, it averages the results obtained. And the micro form could be said to join all the confusion matrices to calculate the corresponding metric.

Chapter 3

Methodology

A general description of the methodology used in this work is presented in Fig. 3.1. In the following subsections, each component is described more in-depth, except the component Results, which is described in Chapter 4.

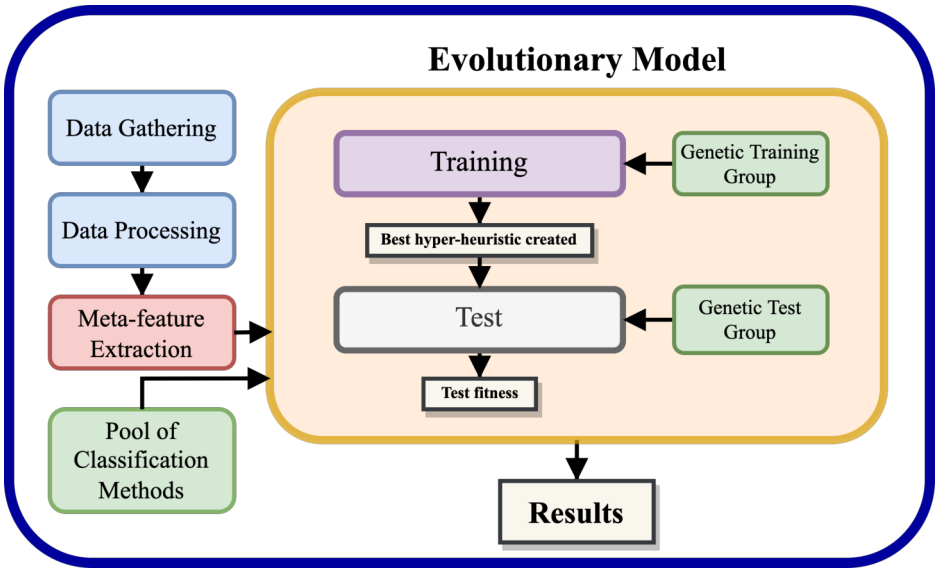

Figure 3.1: Schematic representation of the methodology process.

3.1 Data Gathering

For the creation of the group of datasets, the Kaggle[1] site was used as the main source, where there are large amounts of datasets for different ML tasks. The objective was to create a group of datasets that encompassed different types of text

[1]https://www.kaggle.com

classification tasks (e.g. news classification, email filtering, sentiment detection, etc.). Although, some other sites were also used to collect datasets that were very popular in the literature.

Thus, a total of 34 datasets were collected, with different types of classification tasks such as news classification, sentiment detection, email filtering, cyberbullying detection, hierarchical document classification, disaster detection, political preference, age identification, fake job posting detection, and research articles classification. The textual content of each of the datasets is in English, as was intended at the time of collection. Table 3.1 provides the names of the collected datasets and the text classification task they are associated with. Datasets such as 20ng, movies, r52, wipo_l1, and wipo_l2 have been widely used in various works focused on text classification. Therefore, they are considered benchmarks to test different pre-processing techniques, textual representations, classification methods, complete methodologies, etc.

3.1.1 Dataset Description

The information contained in each of the 34 collected datasets comes from various sources such as CNN (CNNAC, CNNAS), Twitter (CorTws, CybTws, DisTws), Reddit (SuiDect), IMDB (imdbs, IMDBR), The New York Times (NYTAND, NYTATM), Rotten Tomatoes (Rotten, sen_pol), TripAdvisor (TripAd), among others. Within this group of datasets there are a total of 11 types of text classification tasks, each one composed of a different number of datasets, as can be seen in Fig. 3.2. Some tasks have more datasets than others, both News classification and Sentiment detection are those that cover a greater number of datasets, 11 and 10 respectively, and the others with fewer such as 3 (Research articles classification), 2 (Age identification, Hierarchical document classification) or 1 (Email filtering, Cyberbullying detection, Disaster detection, Political preference, Automated Moderation, and Fake job posting detection). Despite not covering all types of tasks and the number of datasets for such tasks are not equivalent, all this content is enough to carry out the different experiments in this work.

As previously mentioned, the textual content within the datasets was intended to be in English so that the data processing was the same for each dataset. Clearly, the largest amount of information within each dataset is textual content, but some datasets provide other types of information that were collected during their construction. For example, some of the datasets of the News classification type provide information about the author who wrote the news, headline, keywords, publication date, and links to sites that feed the news written, among other features. Another example is those datasets that come from sources such as Twitter, where an instance is composed by a *tweet* (textual content), username, date and time posted, and location. This type of information in some works can be very valuable, but in this work, only the textual content is kept.

Dataset	Task	Source
20 Newsgroups (20ng)	News	https://bit.ly/3JrdgL7
AG News (AGNews)	News	https://bit.ly/3ZyV9IR
CNN Articles By Category (CNNAC)	News	https://bit.ly/3mIC9ZW
CNN Articles By Section (CNNAS)	News	https://bit.ly/3mIC9ZW
Coronavirus Tweets (CorTws)	Sentiment	https://bit.ly/3F6YHtG
CSDMC 2010 Spam Corpus (csdmc)	Email Filt.	https://bit.ly/3ZRB14q
Cyberbullying Tweets (CybTws) [68]	Bullying	https://bit.ly/3ZS01Iz
Disaster Tweets (DisTws)	Disaster	https://bit.ly/3ZxmKKg
Fake and real news dataset (F&RNS) [69]	News	https://bit.ly/421qVzy
GOP Debate Sentiment (gopds)	Sentiment	https://bit.ly/3JqUtiP
Highly Rated Children Books (HRCB)	Age	https://bit.ly/41WwvU2
Highly Rated Children Stories (HRCS)	Age	https://bit.ly/41WwvU2
IMDB Sentiment (imdbs) [70]	Sentiment	https://bit.ly/3mGPBx4
IMDB 50K Movie Reviews (IMDBR) [71]	Sentiment	https://bit.ly/3ZUUUYd
Liberals vs Conservatives on Reddit (LvsC)	Political	https://bit.ly/3yqQyfD
Movie Review Dataset (movies) [72]	Sentiment	https://bit.ly/3F70QFK
News Category Dataset (NewsCat) [73]	News	https://bit.ly/3T6tL2u
New York Times Articles By New Desk (NYTAND)	News	https://bit.ly/3ZBFF6J
New York Times Articles By Type of Material (NYTATM)	News	https://bit.ly/3ZBFF6J
oh	News	https://bit.ly/3FcOjjT
r8	News	https://bit.ly/3FcOjjT
r52	News	https://bit.ly/3FcOjjT
Real/Fake Job Posting Detection (RFJob)	Fake job	https://bit.ly/3Ld8pi0
Rotten Tomatoes Reviews Dataset (Rotten)	Sentiment	https://bit.ly/3J5dVQV
Sentence Polarity (sen_pol)	Sentiment	https://bit.ly/3Lbh3xf
Stack Overflow Questions with Quality Rating (StOvQR) [74]	Moderation	https://bit.ly/3kYYWjs
Suicide and Depression Detection (SuiDect)	Sentiment	https://bit.ly/3ysdISS
Topic Modelling For Research Articles By Computer Science (TMACS)	Res. Art.	https://bit.ly/3ZUOpoo
Topic Modelling For Research Articles By Mathematics (TMAMt)	Res. Art.	https://bit.ly/3ZUOpoo
Topic Modelling For Research Articles By Statistics (TMASt)	Res. Art.	https://bit.ly/3ZUOpoo
TripAdvisor Hotel Reviews (TripAd) [75]	Sentiment	https://bit.ly/3JrfZUE
WIPO Level 1 (wipo_l1)	Hier. Doc.	https://bit.ly/3T2aT4s
WIPO (wipo_l2)	Hier. Doc.	https://bit.ly/3T2aT4s
YELP Sentiment (yelp) [70]	Sentiment	https://bit.ly/3mGPBx4

Table 3.1: Datasets used for the experiments. News: News classification. Sentiment: Sentiment detection. Email filt.: Email filtering. Bullying: Cyberbullying detection. Hier. Doc.: Hierarchical document classification. Disaster: Disaster detection. Political: Political preference. Age: Age identification. Fake job: Fake job posting detection. Res. Art.: Research Articles classification. Moderation: Automated Moderation.

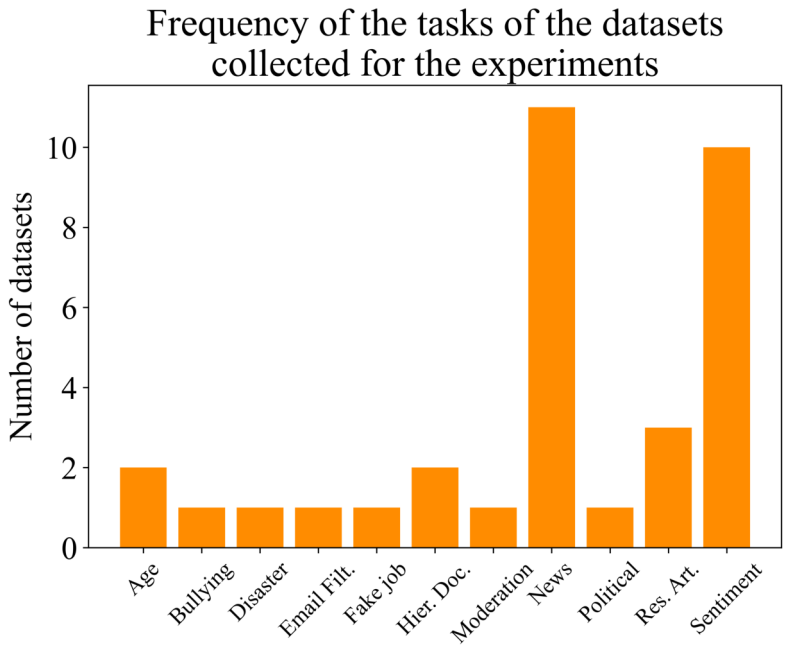

Figure 3.2: Frequency of text classification tasks within the group of datasets.

The two datasets that make up the Age identification task (HRCB and HRCS) are used to determine the interest age of a document (book or story, depending on the dataset) from a text that contains a brief description of it. Both datasets are composed of books or stories for children with an age range of 0-12 years. Therefore, there are certain features within the textual content that make it possible to differentiate, for example, a book recommended for 3-year-olds against a book with a recommended age of 10 years (such as difficult words).

Regarding the Cyberbullying detection task, it was a task that had great growth due to the COVID-19 pandemic because it caused a great increase in the use of social networks as the main sources of communication but also increased the harassment within them. As there is massive interaction between users on a social network, a group of people may not be able to moderate the posts, comments, responses, etc., from a large number of users. This is where classification methods play an important role since these models could be capable of moderating social networks in order to reduce the great harassment that is experienced within them.

The Disaster Detection task is also related to social networks, this task aims to identify if a post is really about a real disaster or not. This would allow news agencies to quickly publicize such disasters or disaster relief agencies to act more quickly, all with the main objective of helping the population.

Email filtering has been a very popular type of task for many years, its name clearly says it, filtering true information from that which is not. This task has been

approached with different methodologies, which have been improving over the years. The fact that it is still a very popular task is due to the fact that email is still a widely used means of communication in different sectors, and today on the Internet there are a wide variety of ways in which they seek to scam people, stealing personal information, phishing, etc.

Fake job posting detection, is another task that focuses on identifying true information in this case of job postings, since this is another technique used by cybercriminals to defraud people. This was also a task that gained popularity in the years of the COVID-19 pandemic, as many companies from different fields were looking for people to work remotely. As it is a false publication, cybercriminals request certain money for the person's application to be accepted, in addition to requesting personal data.

Hierarchical document classification is a task that consists of assigning categories within a hierarchy in such a way that the assigned categories at lower levels are more specific. This task can be applied to different documents such as web pages, news, etc. As the years go by, attempts have been made to train classification methods that allow this process to be automated since new data is generated every day, which requires a lot of personnel to carry out such classification, and the results that have been obtained when applying ML techniques have not been good enough to declare that the problem has already been solved. The two datasets collected that belong to this type of task (wipo_l1 and wipo_l2) are a series of documents that contain the abstract part of patents with categories at a certain level of hierarchy.

With the great rise of the internet and its use as a means of communication, the task of Fake News classification emerged in order to prevent people from being misinformed about news that are not real. This problem has been tried to be solved for some years since this type of news has caused serious problems in the political and social fields because it can easily influence a person to change their mind. On the other hand, News classification is focused on classifying the news according to the field to which they are directed, such as sports, social, political, etc. The datasets collected on this type of task are directed toward both problems.

Today, social networks are used as the main means of communication between a political party and its supporters. Likewise, a person can publish their political tastes or ideas related to this field with great ease. Political preference is a type of task focused on identifying the political preferences of an individual, group, or sector of a population. The dataset collected from this type of task (LvsC) comes from the Reddit website, which contains posts on two types of trends: liberal and conservative.

Research articles classification is a type of task similar to hierarchical document classification, being a problem of determining one or multiple categories to a document, only research articles classification addresses a big problem within the scientific field. Because there are hundreds of thousands of research articles published each year that address problems in one or multiple specific areas, the problem

of determining which articles are relevant when searching for a particular topic has arisen. As there is a lot of information available over the years, the correct identification of what area a certain article belongs to is essential for various systems such as search engines, citation indexes, and digital libraries, among others. For this type of task, a total of three datasets were collected (TMACS, TMAMt, TMASt), the documents that make up the datasets contain only the title and abstract of an article, and its respective category that allows identifying its research area.

Sentiment detection is a task in which the political and business fields have placed great interest in recent years by providing a polarity (e.g. positive, neutral, or negative) or emotional state. (e.g. sad, happy, worried, etc.) to a text created by a person. A large amount of this information is generated on websites where people can easily exchange ideas, points of view, opinions, reviews, etc. The reason these fields are involved is that sentiment detection can provide very valuable information for decision-making. An example from the business field is that it allows them to know if a product is well received by people or not, and thus determine what to do to improve its sales. The datasets collected on this type of task come from various sources such as Twitter, IMDB, Rotten Tomatoes, TripAdvisor, and YELP. There are three types of category groups that can be found in these datasets: two (negative and positive); three (negative, neutral, and positive); and five (extremely negative, negative, neutral, positive, and extremely positive).

Finally, automated moderation is focused on the moderation of question-and-answer (Q&A) websites, where the content of these is mainly generated by various users. These sites are very popular because they are used as learning tools. In such a way that the moderation of the content within these is vital, keeping only content of good or high quality (e.g. non-duplicate content, spam-free, understandable, etc.). Currently, many of these sites depend on the community of users to preserve their quality, but it is a process that consumes too much time since to determine if content should be preserved, it goes through different stages (according to the site) to determine a final decision, within which the community is always involved, either to provide a rating, comment, flag spam or duplicate question, among others. Therefore, providing an automated solution for these types of sites can increase the quality of the content as well as serve as a tool for new users by giving them information on how to answer, rate, or post questions. StOvQR is a dataset with questions asked during the years 2016-2020 on the Stack Overflow [2] site, classified according to their quality (High-quality, low-quality still open, and low-quality closed).

Some of the collected datasets are divided into subsets, in different ways such as training-validation-test, training-test, and by categories (e.g. *Fake* and *True* datasets). In some other cases, multiple files where each one contains only one document, or subsets according to the year of extraction (e.g. CNN News), and in the best of cases a single dataset containing all the documents and their respective categories. In order for the datasets to share the same format, a methodology was

[2]https://stackoverflow.com

proposed, with which each dataset only had two files, the first corresponding to the documents, and the second to the labels of such documents. Once applied to all the datasets in the group, a series of statistics were calculated with the purpose of knowing the composition of each one of the datasets, that is, given a dataset, to which type of task it belongs, and how many categories and documents were found within it. These results are presented in Table 3.2.

Dataset	**Task**	**#Cat**	**#Docs**	**Dataset**	**Task**	**#Cat**	**#Docs**
20ng	News	20	7528	NYTAND	News	43	9335
AGNews	News	4	127600	NYTATM	News	13	9335
CNNAC	News	9	37949	oh	News	23	7399
CNNAS	News	49	37949	r8	News	8	7673
CorTws	Sentiment	5	44955	r52	News	51	9099
csdmc	Email Filt.	2	4327	RFJob	Fake job	2	17880
CybTws	Bullying	6	47692	Rotten	Sentiment	5	156060
DisTws	Disaster	2	11370	sen_pol	Sentiment	2	10662
F&RNS	News	18	44919	StOvQR	Moderation	3	60000
gopds	Sentiment	3	13871	SuiDect	Sentiment	2	232074
HRCB	Age	10	3269	TMACS	Res. Art.	2	20972
HRCS	Age	48	430	TMAMt	Res. Art.	2	20972
imdbs	Sentiment	2	1000	TMASt	Res. Art.	2	20972
IMDBR	Sentiment	2	50000	TripAd	Sentiment	5	20491
LvsC	Political	2	12854	wipo_l1	Hier. Doc.	114	75249
movies	Sentiment	2	2000	wipo_l2	Hier. Doc.	922	75249
NewsCat	News	41	200853	yelp	Sentiment	2	1000

Table 3.2: Distribution of tasks, number of categories and documents for the group of datasets.

When analyzing the content of Table 3.2, it can be seen that in addition to the variability of the types of tasks, this also exists with respect to the number of categories and documents. Regarding the tasks that have a greater number of datasets: News classification has a range of documents from a minimum of 7,399 to a maximum of 200,853 and for categories a minimum of 4 and a maximum of 51; the second task, Sentiment detection has values from 1,000 to 232,074 for the number of documents and from 2 to 5 for the number of categories. For Research articles classification, the number of categories and documents is the same for the 3 collected datasets. Both age identification and hierarchical document classification count a

couple of datasets, which have different numbers of categories, and different numbers of documents in the case of age identification datasets. Finally, the remaining tasks only have a single dataset, for which in general, the number of categories ranges from 2 to 6, and from 4,327 to 47,692 documents. Within the entire group of datasets, SuiDect is the dataset with the largest number of documents (232,074) and HCRS has the fewest (430), wipo_l2 is the dataset with the largest number of categories (922), whereas the smallest number of categories (2) can be found in various datasets such as csdmc, DisTws, imdbs, among others. By having great variability in various aspects, this group of datasets will allow a better analysis of the evolutionary model to be carried out in order to verify its generalization capacity.

3.2 Data Processing

At the end of the collection of the datasets, and after applying a process so that they are only composed of two files (documents and categories), it is necessary to apply processing on the part of the documents, since the data within these still are in a *raw* form. This process is focused on dividing each of the datasets into subsets, to later extract superficial features, which for this work are words, leaving aside other features such as emojis, links, etc. All the following processes were done using Python using several libraries.

First, each of the datasets of the collected group was divided into two subsets: the first with 50% of the data for the training part of the evolutionary model (genetic training group), and the remaining 50% of the data for carrying out the tests of this (genetic test group); the subsets were created in a stratified way and the documents belonging to each subset were randomly selected. Later, having 34 datasets in each genetic group (training and test), these were divided in a stratified way into 80% to train and 20% to test the different classification methods that are described below. The following stages of data processing are applied independently to each of the subsets (training and test) of each genetic group.

Once the splitting process was completed, the extraction of surface features continued. First, given a document, all text within it is converted to lowercase. Then, through the use of regular expressions, only words were extracted, as sequences of alphabetical characters plus the '-' hyphen char, to capture compound words that are common in English. This process was done with the *re* library.

After word extraction, stop words were eliminated, which are words that do not provide relevant information and appear very frequently within texts. The precompiled list of stopwords in English from the NLTK library was used. In addition, to stop word removal, words of short length (less than 3 characters) and long length (greater than 30 characters) were also eliminated. And as a last step, words that appeared only once in the entire dataset were also eliminated.

Once the surface features were extracted, each dataset was transformed using the *tf-idf* method to be used with the classification methods. As previously mentioned,

within each genetic group there are a total of 34 datasets, which in turn are divided into a training set and a test set. For each dataset, the vocabulary is extracted from the training set, and for each term within it, the *idf* is calculated following Eq. 2.2. And then, the vectorization of each document in the training set is performed according to Eq. 2.1, allowing to obtain a matrix within which each element represents the importance of the t term in the document d. Having vectorized the training set, both the vocabulary and the idfs obtained are reused to vectorize the test set using the same Eq. 2.1. The vectors of both sets are normalized by applying the L2-Norm.

3.3 Classification Methods

Once the data processing was carried out, a pool of various ML classification methods was defined, addressing different approaches and configurations of these, so that the evolutionary model could have wide options to choose the most appropriate method for each of the different datasets used.

Methods with various approaches (probabilistic, instance-based, and functions, among others) and different configurations were used according to the hyperparameters available for each method, as follows. Bernoulli (BNB), Complement (CNB), and Multinomial (MNB) from Naïve Bayes; turning normalization on and off in the case of CNB. K-Nearest Neighbors (KNN) combining k (neighbors) with values of 1, 5, 10, and 20; with the distance metrics Manhattan, Euclidean, and cosine similarity. Decision Trees (DT) alternating the quality criteria of entropy and gini; with max_features as None, auto, sqrt, and log2. Logistic Regression (LR) with a regularization parameter C of 0.1, 1, or 10; and using different types of solvers such as lbfgs, newton-cg, liblinear, sag, or saga; in addition to limiting the number of iterations to 10,000. Linear Support Vector Machine (LSVM) in its dual form; the regularization parameter C with values of 0.1, 1, or 10; hinge or squared_hinge as loss functions; and a maximum of 20,000 iterations. Support Vector Machine (SVM) with gamma as scale; the kernels of rbf, sigmoid, and polynomial; and using the degrees of 2, 3, and 4 for the polynomial kernel. These classification methods and their various configurations make it possible to obtain a set of 60 methods, which are described in Table 3.3.

One of the objectives of this work is not to search for the best hyperparameters of a classification method, but rather to determine the most appropriate method for a given dataset based on its meta-features. In this case, the method is selected from the pool of 60 previously defined methods. For each dataset of each genetic group, the training part is used to train each pool method, and the test part of the dataset is used to test their respective classification performance.

The evaluation metric macro F1 was used to measure the performance of the classification methods on each dataset. This metric provides values between 0 and 1; a value close to 1 means good performance and vice versa. The ML methods were implemented in Python, using the libraries *scikit-learn* [76], and *NumPy* [77]. For

#	Base	Hyperparameters	#	Base	Hyperparameters	#	Base	Hyperparameters
1	MNB		21	DT	cr='entropy', mf=None	41	LSVM	C=0.1, l='hinge'
2	CNB	norm=True	22	DT	cr='entropy', mf='auto'	42	LSVM	C=1, l='s_hinge'
3	CNB	norm=False	23	DT	cr='entropy', mf='sqrt'	43	LSVM	C=1, l='hinge'
4	BNB		24	DT	cr='entropy', mf='log2'	44	LSVM	C=10, l='s_hinge'
5	KNN	k=1, m='manhattan'	25	LR	C=0.1, s='lbfgs'	45	LSVM	C=10, l='hinge'
6	KNN	k=1, m='euclidean'	26	LR	C=0.1, s='newton-cg'	46	SVM	C=0.1, ke='rbf'
7	KNN	k=1, m='cosine'	27	LR	C=0.1, s='liblinear'	47	SVM	C=0.1, ke='poly', de=2
8	KNN	k=5, m='manhattan'	28	LR	C=0.1, s='sag'	48	SVM	C=0.1, ke='poly', de=3
9	KNN	k=5, m='euclidean'	29	LR	C=0.1, s='saga'	49	SVM	C=0.1, ke='poly', de=4
10	KNN	k=5, m='cosine'	30	LR	C=1, s='lbfgs'	50	SVM	C=0.1, ke='sigmoid'
11	KNN	k=10, m='manhattan'	31	LR	C=1, s='newton-cg'	51	SVM	C=1, ke='rbf'
12	KNN	k=10, m='euclidean'	32	LR	C=1, s='liblinear'	52	SVM	C=1, ke='poly', de=2
13	KNN	k=10, m='cosine'	33	LR	C=1, s='sag'	53	SVM	C=1, ke='poly', de=3
14	KNN	k=20, m='manhattan'	34	LR	C=1, s='saga'	54	SVM	C=1, ke='poly', de=4
15	KNN	k=20, m='euclidean'	35	LR	C=10, s='lbfgs'	55	SVM	C=1, ke='sigmoid'
16	KNN	k=20, m='cosine'	36	LR	C=10, s='newton-cg'	56	SVM	C=10, ke='rbf'
17	DT	cr='gini', mf=None	37	LR	C=10, s='liblinear'	57	SVM	C=10, ke='poly', de=2
18	DT	cr='gini', mf='auto'	38	LR	C=10, s='sag'	58	SVM	C=10, ke='poly', de=3
19	DT	cr='gini', mf='sqrt'	39	LR	C=10, s='saga'	59	SVM	C=10, ke='poly', de=4
20	DT	cr='gini', mf='log2'	40	LSVM	C=0.1, l='s_hinge'	60	SVM	C=10, ke='sigmoid'

Table 3.3: Group of ML classification methods. norm: Apply second normalization. k: Neighbors. m: Distance metric. cr: Quality criteria. mf: Maximum number of features. C: Regularization parameter. s: Solver. l: Loss function. ke: Kernel. de: Degree for the polynomial kernel.

the training and testing of the ML classification methods, a PC with an Intel Xeon Silver processor @2.1 GHz, 128 GB of RAM, and Windows 10 Pro was used.

3.4 Meta-Feature Extraction

In addition to defining the classification methods (actions) available for the evolutionary model, it is also necessary to extract certain meta-features from each dataset. These meta-features are focused on representing the distribution of the data within each dataset in the best possible way, and thus knowing the properties of a dataset that favor the performance of a certain classification method. Therefore, each of the defined meta-features can provide valuable information for the selection of an appropriate method for a given dataset. For example, if a certain classification method A provides better classification performance than a method B when the categories within a dataset are unbalanced, a meta-feature that allows knowing the standard deviation of the documents by category could be essential to make the decision of

which classification method to apply. For each training part of the datasets of each genetic group, a total of 16 meta-features defined in Table 3.4 were calculated, based mainly on statistics that are calculated at a document, category, and dataset level. Both their definition and the way they are calculated are explained in the following paragraphs.

Abreviation	**Description**
nDocs	Number of documents
nTops	Number of categories
dptAvg	Mean of documents per category
dptMed	Median of documents per category
dptStd	Standard deviation of documents per category
dptStdAvg	Ratio bewteen dptStd and dptAvg
dptEnt	Shannon entropy of documents per category
wpdAvg	Mean of words per document
wpdMed	Median of words per document
wpdStd	Standard deviation of words per document
wpdStdAvg	Ratio between wpdStd and wpdAvg
wpdEnt	Shannon entropy of words per document
pca10	Variance captured by the first 10 components of PCA
pca20	Variance captured by the first 20 components of PCA
pca30	Variance captured by the first 30 components of PCA
cmxWds	Percentage of difficult words in the entire dataset

Table 3.4: Set of meta-features used to represent the data distribution of a dataset.

- **Number of documents.** nDocs corresponds to the number of documents found in the training part of a dataset, which will be used to train each classification method. nDocs can allow the evolutionary model to select classification methods that are capable of performing well with a given number of documents.
- **Number of categories.** Similar to nDocs, nTops is the number of categories found in the training part of a dataset. In turn, this meta-feature can be crucial to identify methods that perform well on binary or multi-class classification datasets.

- **Documents per category: mean.** dptAvg corresponds to the average number of documents per category, which is calculated as the division of nDocs/nTops. This meta-feature measures the centrality of the number of documents per category but is sensitive to extreme values.

- **Documents per category: median.** A meta-feature that also measures the centrality of the number of documents per category, but is not sensitive to extreme values is dptMed, which corresponds to the median of the documents per category. Given the training part of a dataset, this meta-feature is calculated as the middle value of the ordered list of the number of documents that exist in each category.

- **Documents per category: standard deviation.** dptStd measures the dispersion of the documents per category with respect to their mean, through the standard deviation. This meta-feature is calculated by Eq. 3.1.

$$\text{dptStd} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(dpt_i - \mu\right)^2} \tag{3.1}$$

  where N is nTops, dpt_i is the number of documents in category i, and μ is dptAvg. A value of 0 or close to it means that the categories are balanced (same number of documents per category), on the other hand, a large value means that there is an imbalance between the categories. This meta-feature could allow the evolutionary model to select classification methods that are not affected in their performance due to unbalanced categories (a very common problem in the literature).

- **Documents per category: the ratio between standard deviation and mean.** The objective of calculating the ratio between the standard deviation and the mean of the documents per category is also to calculate the dispersion of the documents. However, dptStdAvg allows comparison between datasets that have widely different dptAvg values, since this meta-feature is dimensionless. Its calculation is simple since it is a value corresponding to dptStd/dptAvg.

- **Documents per category: Shannon entropy.** Shannon entropy or information entropy is a measure that allows knowing the uncertainty of a source of information, and thus also knowing the measure of information necessary to deal with such uncertainty. In this case, dptEnt is the average amount of information found in each category of a given dataset, which is calculated using Eq. 3.2.

$$\text{dptEnt} = \sum_{i=1}^{N} P\left(dpt_i\right) \log_e\left(\frac{1}{P\left(dpt_i\right)}\right) \tag{3.2}$$

where N is equal to nTops; $P\left(dpt_i\right)$ is the probability that a category has a total of dpt_i documents. dpt_i is an element of **dpt**, which is a list of size N containing the number of documents that exist in each category. For reasons of ease in computational terms, the base of the logarithm was taken as e; in information theory, it is taken as base 2 because it works in terms of bits (0 or 1).

- **Words per document: mean.** This meta-feature is the average size of the documents within a given dataset, with which we can generally identify how large or small the documents are. Due to the diversity of types of text classification tasks, the documents in a cyberbullying detection dataset are not the same length as those in a research articles classification dataset. wpdAvg can allow the evolutionary model to identify classification methods that are capable of handling long, short, or both types of documents. The way to be calculated within Python is by reading each document (a sequence of characters) and dividing it by the spaces that exist within it, which allows knowing the total number of words within the document. This summation divided by nDocs is the value assigned to wpdAvg.

- **Words per document: median.** wpdMed is a meta-feature that is similar to dptMed, only that it is calculated at a document level. wpdMed takes the middle value of the ordered list of word counts for each document in the training part of a given dataset. By also sharing a feature of wpdAvg, which is measuring data centrality, this meta-feature can perhaps be key to identifying methods that are very sensitive to large feature spaces.

- **Words per document: standard deviation.** Taking Eq. 3.1 as a reference, to apply it at a document level, it results in Eq. 3.3.

$$\text{wpdStd} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(wpd_i - \mu\right)^2} \tag{3.3}$$

  where N is nDocs, wpd_i is the number of words in document i, and μ is wpdAvg. When wpdStd is a large value, it means that there is a large variance between document lengths from the mean, whether they contain many or few words. Datasets with high values in wpdStd are commonly found, such as product reviews, movies, etc., where there is no control over the text size. But there is also the opposite case, for example, where the documents of a dataset represent the abstracts of articles, which rarely exceed the amount of around 150 words.

- **Words per document: the ratio between standard deviation and mean.** Like dptStdAvg, wpdStdAvg is a meta-feature for comparing datasets

where there is a wide difference in means, in this case, the number of words per document. This can allow identifying classification methods that perform well on this type of distribution, regardless of any other meta-characteristic of the dataset (e.g. nDocs, nTops, etc.).

- **Words per document: Shannon entropy.** Using Eq. 3.4, the Shannon entropy is calculated with respect to words per document, that is, the average amount of information in a document when its number of words is observed. In such an equation, N equals nDocs, $P\left(wpd_i\right)$ is the probability that a document has a total of wpd_i words, and $\mathbf{wpd} = [wpd_1, wpd_2, \ldots, wpd_N]$.

$$\text{wpdEnt} = \sum_{i=1}^{N} P\left(wpd_i\right) \log_e \left(\frac{1}{P\left(wpd_i\right)}\right) \tag{3.4}$$

- **Principal Component Analysis.** Principal Component Analysis (PCA) is a process widely used within ML, which, from a dataset consisting of multiple variables, passes it to a space of uncorrelated latent components, which allows obtaining a new set where each feature is an orthogonal component and uncorrelated to all others. These new features or components are known as principal components, and they are ordered in such a way that the first ones explain a greater amount of variance than those that are last. Typically, the components that explain the least amount of variance can be removed because there is no significant loss of information, thus lowering the dimensionality, which in turn may allow for better classification.

  The principal components can be calculated from the singular value decomposition (SVD) defined by Eq. 3.5, where X corresponds to the input data in a space of n dimensions, U and V are unitary matrices, whose columns are known left and right singular vectors respectively, and S is a diagonal matrix containing the singular values of X. The columns of V (right singular vectors) correspond to the eigenvectors. The principal components correspond to the columns of $U \cdot S$ as long as if X is centered.

$$X = U \cdot S \cdot V^T \tag{3.5}$$

  The percentage of information provided by a component c_j is calculated using Eq. 3.6, where λ_i corresponds to the variance of the component i, which is related to the singular value s_i by means of $\lambda_i = s_i^2 / (n-1)$.

$$E_{c_j} = \frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i} \tag{3.6}$$

  In this work, SVD is applied to the tf-idf representations of the training parts of each dataset, which is why it can also be called Latent Semantic Analysis.

As mentioned above, the first components are the ones that provide the most information to the system, in such a way that they are also the ones that capture the greatest variance. Equation 3.7 is used to compute the percentage of variance explained by the first k components.

$$PCA_k = \frac{\sum_{j=1}^{k} \lambda_j}{\sum_{i=1}^{n} \lambda_i} \tag{3.7}$$

Three meta-features were defined from PCA, which correspond to the percentage of explained variance contained by the first 10 (pca10), 20 (pca20), and 30 (pca30) components.

- **Difficult words.** cmxWds represents the percentage of complex or difficult words within the training part of each dataset. This meta-feature is based on the Gunning fog index [78], which is a test that determines the academic degree that a person should have to understand a certain text in the first reading of it.

  A word is called a complex word when it consists of 3 or more syllables, in addition to not being a proper noun, familiar jargon, or compound word. The cmxWds computation was done using the *textstat* [3] library. This meta-feature is calculated by Eq. 3.8.

$$cmxWds = \frac{N_{cw}}{N_{tw}} \tag{3.8}$$

  where N_{cw} is the number of difficult words, and N_{tw} is the total number of words for a given dataset.

The values obtained by the set of 16 meta-features when applied to the training parts of each dataset in the training and test genetic groups are shown in Tables 3.5 to 3.8. Looking at these values, we can notice that there is great variability between the meta-features between datasets.

[3] Available at: https://pypi.org/project/textstat

	20ng	**AGNews**	**CNNAC**	**CNNAS**	**CorTws**	**csdmc**	**CybTws**	**DisTws**	**F&RNS**
				Genetic training group					
nDocs	3011	51040	15179	15179	17981	1730	19076	4548	17967
nTops	20	4	9	49	5	2	6	2	18
dptAvg	150.5	12760	1686.5	309.7	3596.2	865	3179.3	2274	998.1
dptMed	152.5	12748	222	31	3259	865	3182	2274	5.5
dptStd	13.22	57.27	2724.4	738.4	919.1	311	32.85	1410	1546.9
dptStdAvg	0.087	0.004	1.615	2.383	0.255	0.359	0.010	0.620	1.549
dptEnt	2.787	1.386	2.197	3.337	1.609	0.693	1.791	0.693	2.135
wpdAvg	143.7	24.37	528.5	528.5	18.92	171.3	13.11	10.21	234.2
wpdMed	86	24	391	391	20	88	12	11	210
wpdStd	315.2	6.793	566.6	566.6	6.458	403.1	7.528	3.243	185.7
wpdStdAvg	2.192	0.278	1.072	1.072	0.341	2.352	0.573	0.317	0.793
wpdEnt	5.570	3.212	6.917	6.917	3.251	5.599	3.276	2.541	6.236
pca10	0.029	0.023	0.059	0.059	0.028	0.102	0.048	0.020	0.047
pca20	0.048	0.037	0.085	0.085	0.045	0.149	0.069	0.037	0.073
pca30	0.064	0.049	0.103	0.103	0.058	0.177	0.086	0.051	0.093
cmxWds	0.181	0.189	0.199	0.199	0.209	0.200	0.139	0.161	0.234
				Genetic test group					
nDocs	3011	51040	15180	15180	17982	1731	19076	4548	17968
nTops	20	4	8	46	5	2	6	2	13
dptAvg	150.5	12760	1897.5	330	3596.4	865.5	3179.3	2274	1382.1
dptMed	157	12795	293	41	3353	865.5	3189.5	2274	302
dptStd	19.28	86.52	2803.2	748.4	950.3	308.5	48.76	1452	1671.3
dptStdAvg	0.128	0.006	1.477	2.268	0.264	0.356	0.015	0.638	1.209
dptEnt	2.510	1.386	2.079	3.491	1.609	0.693	1.791	0.693	2.311
wpdAvg	138.5	24.40	539.7	539.7	18.93	165.2	13.07	10.28	236.9
wpdMed	85	24	395	395	20	90	12	11	212
wpdStd	257.3	6.794	559.3	559.3	6.424	404.2	8.388	3.143	191.5
wpdStdAvg	1.857	0.278	1.036	1.036	0.339	2.447	0.641	0.305	0.808
wpdEnt	5.569	3.208	6.941	6.941	3.249	5.590	3.280	2.504	6.232
pca10	0.028	0.023	0.061	0.061	0.028	0.101	0.049	0.020	0.047
pca20	0.048	0.037	0.087	0.087	0.045	0.145	0.071	0.035	0.072
pca30	0.064	0.049	0.105	0.105	0.058	0.172	0.087	0.048	0.092
cmxWds	0.184	0.190	0.197	0.197	0.208	0.193	0.140	0.159	0.234

Table 3.5: Values of the meta-features for the datasets.

	gopds	HRCB	HRCS	imdbs	IMDBR	LvsC	movies	NewsCat	NYTAND
					Genetic training group				
nDocs	5548	1307	172	400	20000	5141	800	80340	3733
nTops	3	10	48	2	2	2	2	41	42
dptAvg	1849.3	130.7	3.583	200	10000	2570.5	400	1959.5	88.88
dptMed	1288	123.5	2	200	10000	2570.5	400	1332	35
dptStd	1076	84.51	3.593	3	65	764.5	1	2276.9	121.5
dptStdAvg	0.581	0.646	1.002	0.015	0.006	0.297	0.002	1.162	1.367
dptEnt	1.098	2.302	1.837	0.693	0.693	0.693	0.693	3.713	3.494
wpdAvg	10.28	90.78	55.54	7.112	116.3	7.855	351.9	16.94	27.41
wpdMed	11	81	55	6	87	7	323.5	17	27
wpdStd	2.851	40.48	25.23	5.396	88.42	4.950	157.6	7.068	10.84
wpdStdAvg	0.277	0.446	0.454	0.758	0.759	0.630	0.447	0.417	0.395
wpdEnt	2.443	3.364	4.171	2.771	5.470	2.793	5.87	3.301	3.765
pca10	0.101	0.297	0.096	0.092	0.021	0.028	0.035	0.017	0.049
pca20	0.158	0.589	0.177	0.155	0.032	0.048	0.065	0.029	0.077
pca30	0.194	0.845	0.249	0.208	0.042	0.065	0.090	0.038	0.097
cmxWds	0.184	0.195	0.193	0.200	0.175	0.232	0.189	0.181	0.238
					Genetic test group				
nDocs	5548	1308	172	400	20000	5141	800	80341	3734
nTops	3	10	43	2	2	2	2	41	43
dptAvg	1849.3	130.8	4	200	10000	2570.5	400	1959.5	86.83
dptMed	1235	124	2	200	10000	2570.5	400	1369	39
dptStd	1114.8	83.61	3.965	5	75	779.5	1	2298.1	118.9
dptStdAvg	0.602	0.639	0.991	0.025	0.007	0.303	0.002	1.172	1.369
dptEnt	1.098	2.302	2.042	0.693	0.693	0.693	0.693	3.645	3.491
wpdAvg	10.30	91.39	55.36	7.497	115.8	7.880	345.6	17	27.25
wpdMed	11	81	57	6	86	7	323	17	27
wpdStd	2.831	42.58	26.50	5.229	86.99	4.895	149.6	7.063	10.69
wpdStdAvg	0.274	0.465	0.478	0.697	0.751	0.621	0.432	0.415	0.392
wpdEnt	2.439	3.354	4.257	2.814	5.458	2.797	5.836	3.299	3.762
pca10	0.096	0.300	0.105	0.088	0.021	0.027	0.039	0.017	0.047
pca20	0.152	0.587	0.187	0.150	0.032	0.047	0.070	0.029	0.074
pca30	0.191	0.839	0.260	0.202	0.042	0.063	0.097	0.038	0.094
cmxWds	0.182	0.197	0.190	0.204	0.174	0.232	0.187	0.182	0.236

Table 3.6: Values of the meta-features for the datasets.

	NYTATM	oh	r8	r52	RFJob	Rotten	sen_pol	StOvQR
				Genetic training group				
nDocs	3733	2959	3068	3639	7152	62424	4264	24000
nTops	13	23	8	51	2	5	2	3
dptAvg	287.1	128.6	383.5	71.35	3576	12484	2132	8000
dptMed	17	94	123.5	10	3576	10869	2132	8025
dptStd	661.9	120.6	521	247.8	3197	10504	21	36.77
dptStdAvg	2.305	0.937	1.358	3.474	0.894	0.841	0.009	0.004
dptEnt	2.458	3.075	2.079	3.190	0.693	1.609	0.693	1.098
wpdAvg	27.41	109.7	56.82	63.07	76.33	3.878	10.30	85.30
wpdMed	27	108	39	43	65	3	10	55
wpdStd	10.84	41.77	59.28	65.24	67.90	3.434	4.692	113.3
wpdStdAvg	0.395	0.380	1.043	1.034	0.889	0.885	0.455	1.329
wpdEnt	3.765	5.077	4.845	4.961	5.129	2.199	2.924	5.207
pca10	0.049	0.038	0.157	0.146	0.119	0.025	0.024	0.047
pca20	0.077	0.063	0.204	0.192	0.155	0.042	0.042	0.072
pca30	0.097	0.083	0.233	0.222	0.182	0.054	0.056	0.090
cmxWds	0.238	0.368	0.080	0.083	0.377	0.242	0.235	0.174
				Genetic test group				
nDocs	3734	2960	3069	3640	7152	62424	4264	24000
nTops	11	23	8	51	2	5	2	3
dptAvg	339.4	128.6	383.6	71.37	3576	12484	2132	8000
dptMed	78	90	121	10	3576	11066	2132	7957
dptStd	699	122.4	528.3	247.9	3260	10436	5	71.67
dptStdAvg	2.059	0.951	1.377	3.473	0.911	0.835	0.002	0.009
dptEnt	2.271	3.075	2.079	3.198	0.693	1.609	0.693	1.098
wpdAvg	27.25	108.9	59.93	61.60	76.12	3.884	10.25	84.42
wpdMed	27	106	41	44	65	3	10	55
wpdStd	10.69	42.07	62.95	62.6	66.85	3.451	4.663	111.4
wpdStdAvg	0.392	0.386	1.050	1.016	0.878	0.888	0.454	1.320
wpdEnt	3.762	5.085	4.895	4.938	5.129	2.201	2.920	5.196
pca10	0.047	0.037	0.157	0.147	0.117	0.025	0.024	0.047
pca20	0.074	0.061	0.202	0.193	0.154	0.041	0.041	0.072
pca30	0.094	0.081	0.232	0.223	0.181	0.054	0.056	0.090
cmxWds	0.236	0.369	0.083	0.081	0.376	0.241	0.236	0.174

Table 3.7: Values of the meta-features for the datasets.

	SuiDect	**TMACS**	**TMAMt**	**TMASt**	**TripAd**	**wipo_l1**	**wipo_l2**	**yelp**
				Genetic training group				
nDocs	92829	8388	8388	8388	8196	30099	30099	400
nTops	2	2	2	2	5	114	922	2
dptAvg	46414	4194	4194	4194	1639.2	264	32.64	200
dptMed	46414	4194	4194	4194	884	123.5	10	200
dptStd	17.5	788	1910	2087	1189	404.4	62.64	3
dptStdAvg	0.000	0.187	0.455	0.497	0.725	1.531	1.918	0.015
dptEnt	0.693	0.693	0.693	0.693	1.609	4.573	3.936	0.693
wpdAvg	58.62	95.08	95.08	95.08	97.20	63.94	63.94	5.432
wpdMed	28	93	93	93	72	61	61	5
wpdStd	98.05	37.22	37.22	37.22	94.16	29.78	29.78	3.170
wpdStdAvg	1.672	0.391	0.391	0.391	0.968	0.465	0.465	0.583
wpdEnt	4.924	5.001	5.001	5.001	5.330	4.757	4.757	2.423
pca10	0.044	0.026	0.026	0.026	0.037	0.033	0.033	0.116
pca20	0.064	0.042	0.042	0.042	0.057	0.053	0.053	0.184
pca30	0.081	0.055	0.055	0.055	0.074	0.068	0.068	0.238
cmxWds	0.125	0.384	0.384	0.384	0.152	0.320	0.320	0.126
				Genetic test group				
nDocs	92829	8388	8388	8388	8196	30100	30100	400
nTops	2	2	2	2	5	114	911	2
dptAvg	46414	4194	4194	4194	1639.2	264	33.04	200
dptMed	46414	4194	4194	4194	865	123	10	200
dptStd	71.5	735	1972	2131	1187.6	406.1	63.14	6
dptStdAvg	0.001	0.175	0.470	0.508	0.724	1.538	1.911	0.03
dptEnt	0.693	0.693	0.693	0.693	1.609	4.593	3.961	0.693
wpdAvg	58.66	95.18	95.18	95.18	99.05	63.90	63.90	5.535
wpdMed	27	93	93	93	73	61	61	5
wpdStd	98.71	36.71	36.71	36.71	93.05	29.76	29.76	3.013
wpdStdAvg	1.682	0.385	0.385	0.385	0.939	0.465	0.465	0.544
wpdEnt	4.921	4.988	4.988	4.988	5.359	4.758	4.758	2.387
pca10	0.043	0.027	0.027	0.027	0.038	0.033	0.033	0.117
pca20	0.063	0.043	0.043	0.043	0.057	0.052	0.052	0.185
pca30	0.080	0.055	0.055	0.055	0.074	0.068	0.068	0.240
cmxWds	0.125	0.384	0.384	0.384	0.152	0.321	0.321	0.138

Table 3.8: Values of the meta-features for the datasets.

3.5 Evolutionary Model

GAs are optimization algorithms that can be applied to various problems because they are not problem dependent. One of the best features of GAs is the ability to find solutions mainly due to its evolutionary operators. The evolutionary operators of a GA allow to get out of local optima by exploring and exploiting the space of solutions, for which a better solution can be provided.

The evolutionary model designed for this work is schematically represented in Fig. 3.3, where the first phase (training) is designed according to a GA. The second phase of the model (test) consists of testing the best hh that was obtained during the training phase. The remaining content of this section corresponds to the explanation of how our problem was approached with a GA.

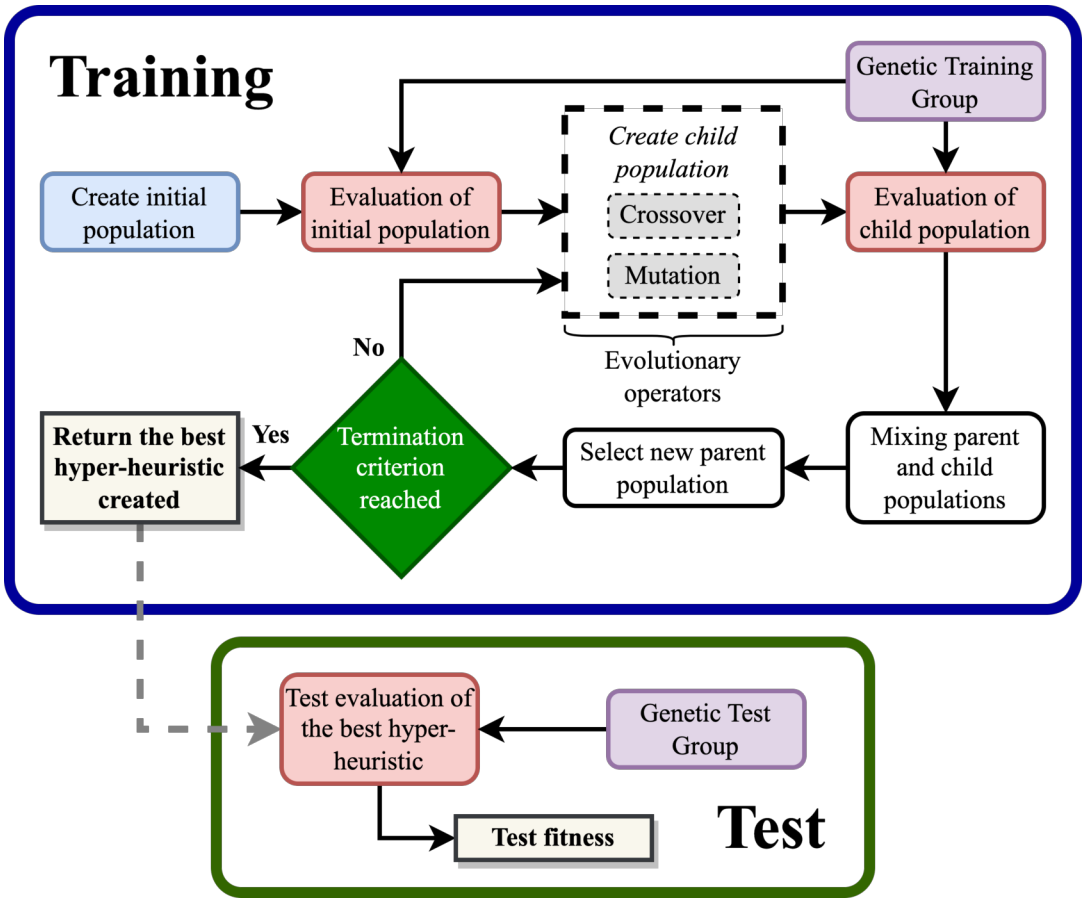

Figure 3.3: General process of the evolutionary model to learn and evaluate hhs.

3.5.1 Individuals

The definition of individuals is one of the most essential parts of a GA since an individual can be represented/designed in multiple ways, which allows for generating different possible solutions to the problem being addressed.

In this work, an individual is a hh, and a hh is defined as a set of m if-then rules, plus an else-rule. Each one of the rules of a hh is made up of a set of conditions and an action, with the exception of the else-rule, the latter only has one action, and it is applied in case none of the other rules are applied, as shown in Fig. 3.4, $C_{x,y}$ denotes the y condition for the x rule. The conditions inside the rules are joined by the AND logical operator, it has been decided to be that way because the OR logical operator is implemented explicitly at the rule level, or in other words, the conditions of rule A are met OR the conditions of rule B are met. In a way, each hh functions as a meta-classifier, which according to each rule provides a specific classification method for a single dataset.

IF $C_{1,1}$ AND $C_{1,2}$ AND ... AND C_{1,n_1} THEN **ACTION**$_1$
IF $C_{2,1}$ AND $C_{2,2}$ AND ... AND C_{2,n_2} THEN **ACTION**$_2$
...
IF $C_{m,1}$ AND $C_{m,2}$ AND ... AND C_{m,n_m} THEN **ACTION**$_m$
ELSE **ACTION**$_{m+1}$

Figure 3.4: Representation of the rules that compose a single hh.

The way in which a hh determines the classification method for a dataset is demonstrated in Algorithm 1. The following are received as arguments: the set of meta-features extracted from the training part of the dataset, and the hh that will be used. The evaluation is carried out sequentially, that is, it starts with rule 1, then with rule 2, and so on until the rule is found in which all its conditions are satisfactorily fulfilled. But there is also the case that the conditions of any rule are not met, for this case, the else rule will be applied. Once the rule is found, the action associated with it is taken, which is the classification method that is expected to be optimal or one close to it for that dataset.

Due to the fact that the design of the individuals for this problem is somewhat complex, in the following paragraphs the composition of these is broken down, providing a concise definition of each one of its components and the way in which they are created.

Algorithm 1 Using a hh to determine a classification method for a single dataset

Require: hh, mf_D ▷ Hyper-heuristic, set of meta-features of a dataset D

```
1: function EvaluateDataset(hh, mf_D)
2:     for each rule ∈ hh do
3:         if rule ≠ else_hh then
4:             if CheckConditions(rule, mf_D) then
5:                 return action_rule
6:         else
7:             return action_else_hh

8: function CheckConditions(rule, mf_D)
9:     for each condition ∈ rule do
10:        if not Satisfies(condition, mf_D) then
11:            return False
12:    return True
```

Rule set

As mentioned above, a set of rules is an individual/hh, where each of its corresponding rules can also be named low-level heuristics. In general, the hh is responsible for selecting a rule to provide a solution that is expected to be the most appropriate for an instance (dataset) of the general problem (group of datasets). In Python, the list data type is used to create each set of rules.

Rule

In this work, a low-level rule or heuristic is composed of a set of conditions and an action. In programming, it can be seen as an if-then statement, since certain conditions must be met (it depends on the use of logical operators) to apply a certain action. In this case, all the conditions of a rule have to be met in order for an action to be returned. Due to this composition, it is necessary to create a final rule (else statement) without a set of conditions to be applied in case none of the if-then rules are met, this rule is added to the end of the set. These rules can also be created using the list data type.

Condition

Each condition of a rule is designed to evaluate only one meta-feature from the set previously defined in Table 3.4. This process is done by means of a comparison operator, the respective value of the meta-feature, and a reference value. In this way, the value of the meta-feature and the reference value have the role of operands, the condition is fulfilled as long as the result of the operation is true. A condition

has the form of a 3-tuple (meta-feature, comparison operator, reference value).

The comparison operators were limited to two: less than (<) and greater than (>). The equal comparison operator (=) was omitted due to the fact that it is a criterion with a very low probability of being met. Although the reference value is randomly generated at the beginning, it is necessary to establish the ranges in which such a value can be generated. For example, given a meta-feature whose value only varies between 0-10 regardless of the dataset, a reference value outside these ranges would make the condition created with such a meta-feature always true, therefore, the condition and the meta-feature would become irrelevant within the rule. Added to the above, for the design of the rules it was contemplated that a meta-feature could be evaluated on more than one occasion within a rule, this allows performing range checks such as (nDocs $> 52,893$ AND nDocs <79,463).

The rules work as a conjunction of multiple conditions (the conditions are joined by means of the logical AND operator), that is, for the action associated with a rule to be applied, it is necessary that all the conditions of the rule are fulfilled. Although disjunctions (logical OR operator) were not contemplated to be implemented within the rules, as was mentioned at the beginning of this subsection, they exist at a rule level.

Action

The action associated with each rule means the classification method that will be used to train and evaluate a certain dataset as long as all the conditions of the rule are met. The Algorithm 1 function EvaluateDataset provides such a classification method. Thus, the action assigned to a rule is one of the 60 classification methods from the pool of classification methods (ML classification methods) defined in Table 3.3.

As can be seen, within the pool of classification methods there are some methods that have a higher frequency (multiple variants due to multiple configurations of their hyperparameters) such as KNN, LR, LSVM, and SVM, the latter two use the same approach (functions by support vector machines). If the classification method assigned to a rule is taken randomly directly from the pool, the probability that one of these methods will be selected is very high, which would not allow knowing the performance of methods such as Naïve Bayes (BNB, CNB and MNB), DT, etc. To address this problem, it was decided that any classification method has the same probability of being selected, but the selected configuration of that method can be randomly selected.

The methodology for selecting a classification method and its respective hyperparameter configuration is demonstrated in Algorithm 2. According to this methodology, for example, a KNN method and a Naïve Bayes method have the same probability of being selected despite having different numbers of hyperparameter configurations within the pool.

Finally, an example of a hh created by the evolutionary model is shown in Fig. 3.5,

Algorithm 2 Select a classification method and its hyperparameter configuration from a pool of classification methods

Require: *pool* ▷ Pool of classification methods
1: **function** SELECTMETHOD(*pool*)
2: *methods* ← [NB, KNN, DT, LR, SVM]
3: *ms* ← RANDOMMETHODFROM(*methods*)
4: *method* ← GETRANDOMMETHODFROM(*pool*, *ms*)
5: **return** *method*

in which it can be seen that there are rules with one or more conditions, rules whose conditions create a range check, different actions that can apply the rules, among other aspects.

Rule 1: **if** *pca30* < 0.2366 **and** *pca30* > 0.1380 **then**
LSVM C=0.1, l='hinge'
Rule 2: **if** *dptEnt* > 1.3587 **and** *pca30* > 0.1380 **and** *dptAvg* < 13680.480 **then**
KNN k=1, m='euclidean'
Rule 3: **if** *pca30* < 0.2366 **and** *dptEnt* > 2.1107 **then**
LSVM C=0.1, l='hinge'
Rule 4: **if** *nDocs* > 6139.839 **and** *pca10* < 0.2474 **and** *pca20* < 0.1928 **then**
MNB
Rule 5: **else**
SVM C=1, ke='sigmoid'

Figure 3.5: Example of a hh created by the evolutionary model.

3.5.2 Initialization

A GA starts with an initial population of individuals. In this work, the initial population of hhs is created at random. The size of such a population has to be constant throughout the evolutionary process. To avoid the creation of complex hhs (a large number of rules and/or conditions), it is necessary to establish a minimum and maximum number of rules that can be in one hh and the same for the number of conditions in a rule. The limits for the generation of the random reference value against which a meta-feature will be compared can be set as 0 and the maximum value found in the genetic training group of such meta-feature.

The process of creating the initial population is shown in the Algorithm 3, taking into account the several aspects mentioned above. The limit values for the meta-features can change according to the genetic training group, therefore $limits_f$ must always be a parameter that the function receives. Both $size_p$ and *pool* are constant in all generations of the evolutionary model and are defined in a previous stage.

The maximum and minimum values for the rules and conditions are defined within the CreateInitialPopulation function, whereby the number of rules for an individual is a randomly generated number within these limits, and the same goes for the number of conditions within a rule. The set $features$ corresponds to the set of 16 meta-features (see Table 3.4), and *operators* is a set containing only the comparison operators (<) and (>). Finally, SelectMethod is the same function that has been introduced in Algorithm 2.

Algorithm 3 Creation of an initial population of hhs

Require: $size_p, pool$ ▷ Size of population, pool of classification methods
Require: max_r, min_r ▷ Maximum and minimum number of rules for a hh
Require: max_c, min_c ▷ Maximum and minimum number of conditions for a rule
Require: $features, operators$ ▷ Set of meta-features, and set of operators
Require: $limits_f$ ▷ List of maximum values for each meta-feature extracted from a genetic training group

function CreateInitialPopulation($size_p$, $limits_f$, $pool$)
    $population \leftarrow [\,]$
    for $i \leftarrow 1$, $size_p$ **do**
        $individual \leftarrow [\,]$
        $n_r \leftarrow$ RandomInt(min_r, max_r)
        for $j \leftarrow 1$, n_r **do**
            $n_c \leftarrow$ RandomInt(min_c, max_c)
            $conditions \leftarrow$ CreateConditions(n_c, $limits_f$)
            $action \leftarrow$ SelectMethod($pool$)
            $rule \leftarrow$ CreateRule($conditions$, $action$)
            $individual\,[j] \leftarrow rule$
        $action_e \leftarrow$ SelectMethod($pool$) ▷ Action for the else rule
        $individual\,[n_r + 1] \leftarrow action_e$
        $population\,[i] \leftarrow individual$
    return $population$

function CreateConditions(n_c, $limits_f$)
    $conditions \leftarrow [\,]$
    for $i \leftarrow 1, n_c$ **do**
        $feature \leftarrow$ SelectFeature($features$)
        $operator \leftarrow$ SelectOperator($operators$)
        $limit \leftarrow$ LimitValueFeature($limits_f$, $feature$)
        $value \leftarrow$ RandomNumber(0, $limit$) ▷ Reference value
        $condition \leftarrow$ CreateCondition($feature$, $operator$, $value$)
        $conditions\,[i] \leftarrow condition$
    return $conditions$

3.5.3 Fitness Evaluation

The fitness evaluation of the hhs population helps to identify those hhs that have a better performance for the solution of the problem. This also makes it possible to identify and select the hhs that could take the role of parents of the population of the next generation, and at the same time discard those hhs that have low fitness. Likewise, this evaluation can identify the best hh that was developed during the entire evolutionary process of a GA execution.

The fitness evaluation of a hh begins with the extraction of the 16 meta-features from the training part of a given dataset, which corresponds to 80% of the dataset data. The hh and the values of the meta-features are provided to the function EvaluateDataset (shown in the Algorithm 1), which is responsible for providing the action (classification method) to be applied to such a dataset. Both the training part and the test part (remaining 20% of the data) of the dataset are transformed according to the tf-idf method, these new representations are used to train and test the assigned classification method. The performance of the classification method is measured through the macro F1 metric. A hh has to be tested with more than one dataset to get a more accurate fitness to reality. During the evolutionary process, the total fitness of a hh is determined with the use of the genetic training group; and it is calculated as the average of the F1 macros obtained by each of the methods assigned to each dataset of the group. Within each population, it is very likely that there is a wide diversity of fitness due to the way the GA is designed, which means a wide diversity of possible solutions to the problem.

Normally in GAs, the fitness evaluation of individuals is the process that requires the greatest amount of time and sometimes computational resources as well. This problem can occur in this work, since in order to know the fitness of a hh it is necessary to train multiple classification methods (one method for each dataset of the group), in addition to consider the computational time of extraction of the 16 meta-features for each dataset. The evaluation is carried out on a population of multiple hhs in multiple generations, therefore, it would become a too-demanding process. To avoid this problem, there are two options. The first one is a stage prior to the evolutionary process, which consists of extracting and storing the 16 meta-features of the datasets of both genetic groups, as well as training and testing all the classification methods in the pool to know their F1 macro with each of the datasets of such groups. And the second one only needs to compute the 16 meta-features on the first generation, to train and test each method only a single time and store the results. In both cases, once these values are stored, the fitness evaluation process of a hh only requires reading the corresponding meta-features of the dataset that is being evaluated, and the same to know the F1 macro value when a certain classification method has to be applied to such dataset. This fitness evaluation process for a population of hhs is shown in Algorithm 4.

Algorithm 4 Evaluation of a population of hhs

Require: $population$, $features_T$, $F1_T$ ▷ Population of hhs, values of the meta-features and the macro F1 for each dataset of the genetic training group T.

```
1: function EvaluatePopulation(population, features_T, F1_T)
2:     fitness ← [ ]
3:     N ← NumberDatasets(features_T)
4:     for each hh ∈ population do
5:         score ← 0
6:         for each features_D ∈ features_T do
7:             dataset ← GetDataset(features_D)
8:             action ← EvaluateDataset(hh, features_D)
9:             f1 ← GetF1(F1_T, dataset, action)
10:            score += f1
11:        score ← score/N
12:        fitness[hh] ← score
```

3.5.4 Crossover Operator

In order to achieve a better explanation of how the evolutionary operators were designed and how they are applied, a hh could also be represented as a set of blocks, as shown in Fig. 3.6. As can be seen, the components of a block (rule) are the conditions and the action that compose it, with the exception of the last block (rule $m+1$), which corresponds to the action associated with the else-rule. Due to the way the creation of the initial population and hhs has been designed (see Algorithm 3), the size of the blocks/rules is not constant, so there may be blocks with few or many cells (multiple conditions).

The objective of the evolutionary operators is to allow wider exploration of the space of solutions, each of these operators can provide new solutions from some previous ones by applying different methodologies. In the case of the evolutionary crossover operator, we have designed different versions that allow the creation of new child hhs from two parent hhs by combining different components of these. Taking the block representation as a reference, the operator is designed to work at the level of blocks, rules, and conditions.

At block level, the crossover operator is applied by implementing the single point [79] technique. At this level, the operator starts with two hhs randomly selected from a population and then finds the length of the hh with the fewest number of rules. This length is taken as a delimiter to calculate a random division point that does not exceed such hh. Then, each of the selected hhs is divided into two rule subsets according to the split point. The union of subgroup 1 of hh 1 with subgroup 2 of hh 2 corresponds to the first new hh. And the second new hh is created with subgroup 1 of hh 2 and subgroup 2 of hh 1. This process is represented

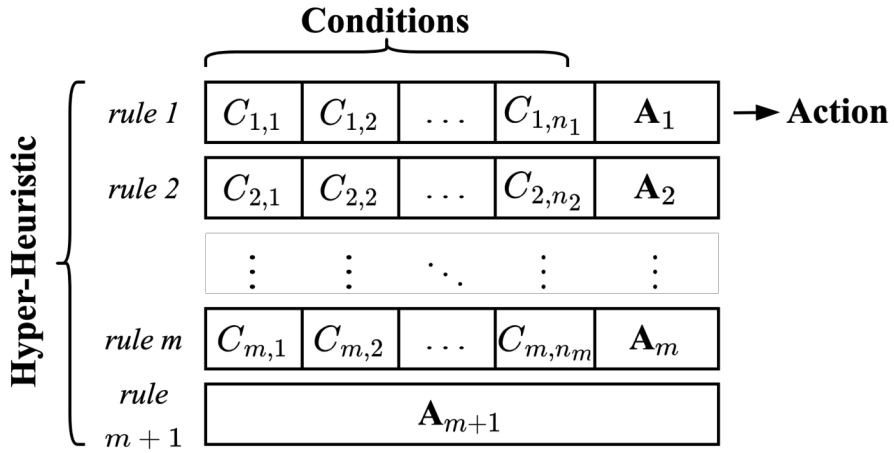

Figure 3.6: Block representation of a hh.

in Fig. 3.7 and described in Algorithm 5.

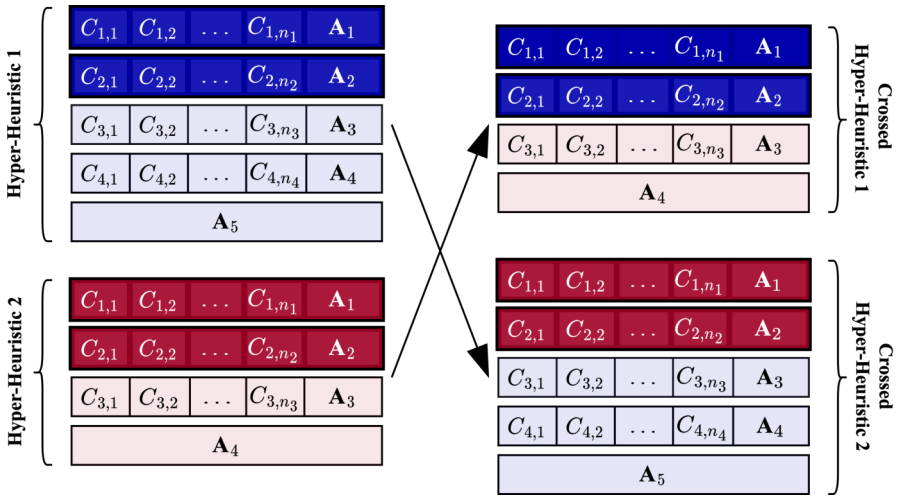

Figure 3.7: Crossover operator at block level.

The methodology designed to apply the crossover operator at rule level is the following. First, it is necessary to randomly select two hhs from a population P, to then identify the hh with the least number of rules. Once such a hh is identified, a rule is selected at random, the else-rule is also considered. Then, the rule of the other hh that is in the same position as the previously selected rule is selected. If the else rule was selected then the selected rule of the other hh also has to be the else rule. Finally, the two new hhs are created by exchanging the selected rules of the chosen hhs. The process is shown in Fig. 3.8 and Algorithm 6.

Algorithm 5 Crossover operator at block level

Require: P ▷ Population of hhs

1: **function** CROSSOVERBLOCK(P)
2: $P_c \leftarrow [\,]$ ▷ Population of new hhs
3: $N \leftarrow$ SIZE(P)
4: **for** $i \leftarrow 1,\ N/2$ **do**
5: $hh_1 \leftarrow$ RANDOMHHFROM(P)
6: $hh_2 \leftarrow$ RANDOMHHFROM(P)
7: $M \leftarrow$ SIZESMALLESTHH($hh_1,\ hh_2$)
8: $cp \leftarrow$ RANDOMINT(1,$M-1$)
9: $new\_hh_1 \leftarrow$ JOINHHSSEGMENTS($hh_1\,[1:cp],\ hh_2\,[cp+1:\text{end}]$)
10: $new\_hh_2 \leftarrow$ JOINHHSSEGMENTS($hh_2\,[1:cp],\ hh_1\,[cp+1:\text{end}]$)
11: $P_c\,[2i-1] \leftarrow new\_hh_1$
12: $P_c\,[2i] \leftarrow new\_hh_2$
13: **return** P_c

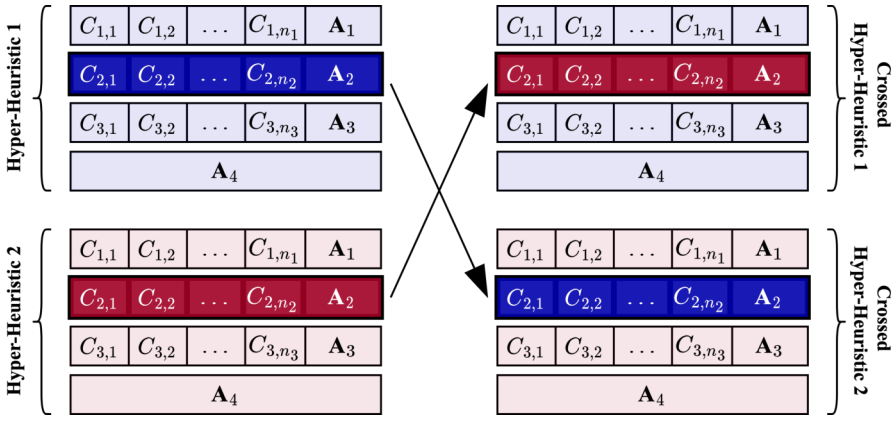

Figure 3.8: Crossover operator at rule level.

Finally, at condition level, the procedure starts from two randomly selected hhs, from which two pairs of rules are selected in the same way. Once the two pairs of rules have been selected, the new hhs are created by exchanging conditions and/or actions between the previously selected rules, therefore, in this case, the else rule cannot be contemplated since it does not have conditions. Exchanges only occur between two rules, that is, the first selected rule of hh 1 exchanges conditions/action with the first selected rule of hh 2, creating two new rules; the other two new rules are created by exchanging the components of the second selected rules from the hhs. The number of conditions (including the action) that can be exchanged between two rules is randomly determined by the operator. This process is shown in Fig. 3.9 and Algorithm 7.

Algorithm 6 Crossover operator at rule level

Require: P ▷ Population of hhs

function CROSSOVERRULES(P)
  $P_c \leftarrow [\,]$
  $N \leftarrow$ SIZE(P)
  for $i \leftarrow 1, N/2$ **do**
    $hh_1 \leftarrow$ RANDOMHHFROM(P)
    $hh_2 \leftarrow$ RANDOMHHFROM(P)
    if $hh_1 > hh_2$ **then**
      $hh_t \leftarrow hh_1$
      $hh_1 \leftarrow hh_2$
      $hh_2 \leftarrow hh_t$
    $r_{hh_1} \leftarrow$ RANDOMRULEFROM(hh_1)
    $p \leftarrow$ POSITION(hh_1, r_{hh_1})
    $r_{hh_2} \leftarrow hh_2\,[p]$
    if $r_{hh_1} = else\_rule_{hh_1}$ **then**
      $r_{hh_2} \leftarrow else\_rule_{hh_2}$
    $new\_hh_1 \leftarrow$ REPLACERULEHH(hh_1, r_{hh_1}, r_{hh_2})
    $new\_hh_2 \leftarrow$ REPLACERULEHH(hh_2, r_{hh_2}, r_{hh_1})
    $P_c\,[2i-1] \leftarrow new\_hh_1$
    $P_c\,[2i] \leftarrow new\_hh_2$
  return P_c

As can be seen in Algorithms 5 to 7, the three operators are designed to create a new population P_c of the same size as population P.

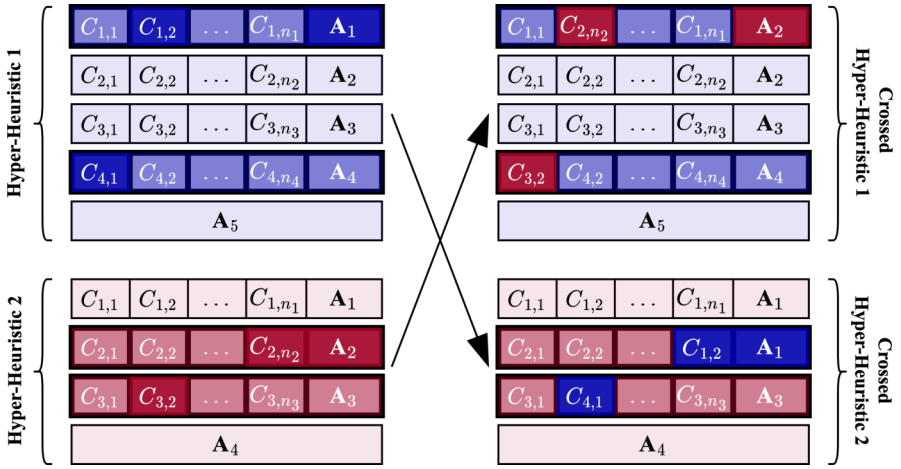

Figure 3.9: Crossover operator at condition level.

Algorithm 7 Crossover operator at condition level

Require: P ▷ A population of hhs

1: **function** CrossoverCondtions(P)
2: $P_c \leftarrow [\,]$
3: $N \leftarrow$ Size(P)
4: **for** $i \leftarrow 1,\ N/2$ **do**
5: $hh_1 \leftarrow$ RandomHHFrom(P)
6: $hh_2 \leftarrow$ RandomHHFrom(P)
7: $r1_{hh_1},\ r2_{hh_1} \leftarrow$ TwoRandomRulesFrom(hh_1)
8: $r1_{hh_2},\ r2_{hh_2} \leftarrow$ TwoRandomRulesFrom(hh_2)
9: $n\_r1_{hh_1}, n\_r1_{hh_2} \leftarrow$ CreateRules($r1_{hh_1},\ r1_{hh_2}$)
10: $n\_r2_{hh_1}, n\_r2_{hh_2} \leftarrow$ CreateRules($r2_{hh_1},\ r2_{hh_2}$)
11: $new\_hh_1 \leftarrow$ ReplaceRulesHH($hh_1,\ r1_{hh_1},\ n\_r1_{hh_1},\ r2_{hh_1},\ n\_r2_{hh_1}$)
12: $new\_hh_2 \leftarrow$ ReplaceRulesHH($hh_2,\ r1_{hh_2},\ n\_r1_{hh_2},\ r2_{hh_2},\ n\_r2_{hh_2}$)
13: $P_c\,[2i-1] \leftarrow new\_hh_1$
14: $P_c\,[2i] \leftarrow new\_hh_2$
15: **return** P_c

16: **function** CreateRules($r_1,\ r_2$)
17: $M \leftarrow$ SizeSmallestRule($r_1,\ r_2$)
18: $k \leftarrow$ RandomInt($1,\ \frac{M}{2}$)
19: $n\_r_1 \leftarrow r_1$
20: $n\_r_2 \leftarrow r_2$
21: $components_{r_1} \leftarrow$ kRandomComponentsFrom($r_1,\ k$)
22: $components_{r_2} \leftarrow$ kRandomComponentsFrom($r_2,\ k$)
23: **for** $i \leftarrow 1,\ k$ **do**
24: $c_{r_1} \leftarrow components_{r_1}\,[i]$
25: $c_{r_2} \leftarrow components_{r_2}\,[i]$
26: **if** $c_{r_1} = action_{r_1}$ **or** $c_{r_2} = action_{r_2}$ **then**
27: $c_{r_1} \leftarrow action_{r_1}$
28: $c_{r_2} \leftarrow action_{r_2}$
29: $n\_r_1 \leftarrow$ ReplaceComponent($n\_r_1,\ c_{r_1},\ c_{r_2}$)
30: $n\_r_2 \leftarrow$ ReplaceComponent($n\_r_2,\ c_{r_2},\ c_{r_1}$)
31: **return** $n\_r_1,\ n\_r_2$

3.5.5 Mutation Operator

The second evolutionary operator is in charge of mutating the hhs, which allows exploiting small sectors of the space of solutions and also has the ability to get out of local maxima, expanding the possibilities of finding better hhs. In this work, the mutation operator addresses four levels: block, rule, condition, and operator. This operator is applied over the hhs of the child populations (created by the crossover operator) with a defined probability. When a hh is selected by this operator, it must also be determined at which level the mutation will be applied, in such a way that the four levels have the same probability of being selected.

Block-level mutation consists of removing or adding a new rule to a given hh. In the case of adding a new rule, the rule is created randomly, similar to how the rules of the hhs of the initial population were created. The way to establish if a rule has to be removed or added is by means of a value p between 0 and 1 obtained randomly, and the number of rules in the hh. Two conditions were designed for a rule to be added to the hh: 1) p is less than 0.5 and the number of rules in the hh is less than the maximum number of rules allowed or 2) p is greater or equals 0.5 and the number of rules is equal to the minimum number of rules allowed. The rule is added at a position that is also calculated at random, but it can never come after the else rule. Therefore, if neither of the two previous conditions can be fulfilled, a randomly selected rule is eliminated. It is worth mentioning that for the mutation at this level, the else rule is not contemplated. This process can also be visualized in Fig. 3.10 and Algorithm 8.

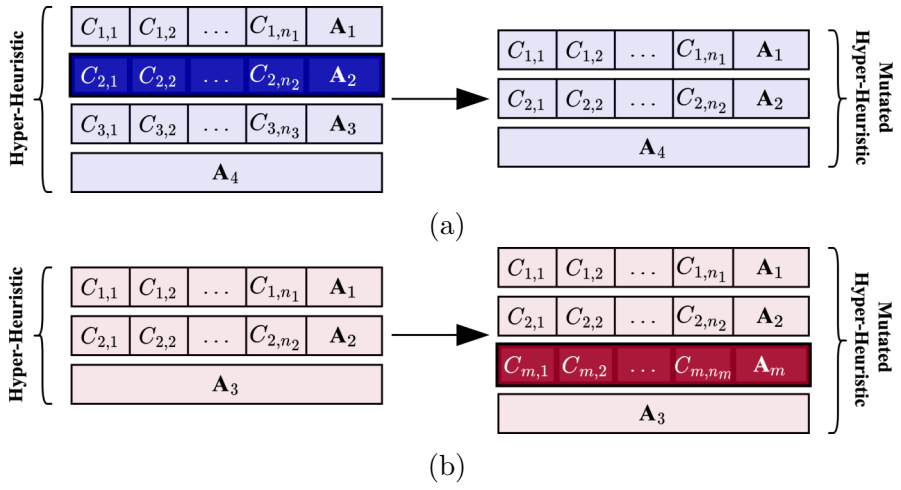

Figure 3.10: Mutation operator at block level: (a) Delete a rule and (b) Create and add a new rule.

At rule level, given a hh, the mutation operator selects a random number of rules no greater than half the number of rules in the hh. These selected rules will

Algorithm 8 Mutation operator at block level

Require: hh, max_r, min_r ▷ Selected hh, maximum and minimum number of rules allowed

```
1: function MutationBlock(hh, max_r, min_r)
2:     p ← RandomFloat(0,1)
3:     M ← Size(hh)
4:     if (p < 0.5 and M < max_r) or (p ≥ 0.5 and M = min_r) then
5:         rule ← CreateRandomRule( )
6:         hh ← InsertRule(hh, rule)
7:     else
8:         hh ← RemoveRandomRule(hh)
9:     return hh
```

be replaced by new rules that are created randomly (similar to how it is done at the block level). At this level, the else rule is also considered a possible rule that can be replaced by a new else rule. This process is illustrated in Fig. 3.11 and Algorithm 9.

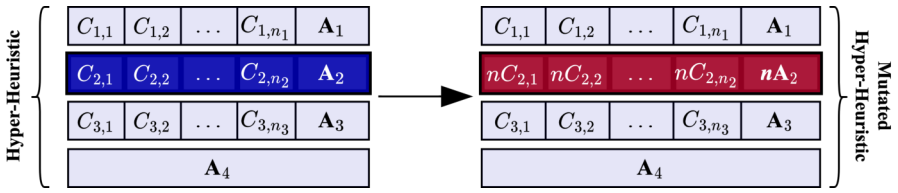

Figure 3.11: Mutation operator at rule level.

Algorithm 9 Mutation operator at rule level

Require: hh ▷ Hyper-heuristic to be mutated

```
1: function MutationRules(hh)
2:     M ← Size(hh)
3:     k ← RandomInt(1, M/2)
4:     rules ← kRandomRulesFrom(hh, k)
5:     for each rule ∈ rules do
6:         if rule ≠ else_rule_hh then
7:             new_rule ← CreateRule( )
8:         else
9:             new_rule ← CreateElseRule( )
10:        hh ← ReplaceRule(hh, rule, new_rule)
11:    return hh
```

The mutation operator, at condition level, performs a mutation at a deeper level with the possibility of not overriding the rules of a hh entirely. Given a hh, the

operator first selects a random number of rules, this number of rules will always be less than or equal to half the number of rules that exist in the hh. Once the rules to be mutated have been identified, the operator proceeds to replace a random number of conditions (the action may also be included) with new conditions and/or actions created randomly for each of these rules. The number of conditions in a rule that can be substituted is less than or equal to half the number of conditions in the rule. Because the operator at this level is mainly focused on condition substitution, the else rules are not considered. This process is shown in Fig. 3.12 and Algorithm 10.

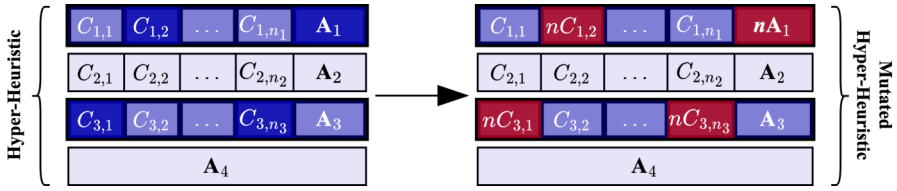

Figure 3.12: Mutation operator at condition level.

Algorithm 10 Mutation operator at condition level

Require: *hh* ▷ Hyper-heuristic to be mutated

```
1:  function MutationConditions(hh)
2:      M ← Size(hh)
3:      k ← RandomInt(1, M/2)
4:      rules ← kRandomRulesFrom(hh, k)
5:      for each rule ∈ rules do
6:          rule_m ← rule
7:          N ← Size(rule)
8:          k ← RandomInt(1, N/2)
9:          components ← kRandomComponentsFrom(rule, k)
10:         for each c ∈ components do
11:             if c ≠ action_rule then
12:                 new_c ← CreateCondition( )
13:             else
14:                 new_c ← CreateAction( )
15:             rule_m ← ReplaceCondition(rule_m, c, new_c)
16:         hh ← ReplaceRule(hh, rule, rule_m)
17:     return hh
```

Being the last type of mutation in this work, operator mutation is a process similar to condition mutation. The main difference between the mutation operators at these two levels is that the mutation operators do not replace a condition of a rule, but rather the comparison operator and/or the reference value of the condition.

The operator at this level only works with the conditions of the rules excluding the actions and therefore also the else-rules. The process of selecting the rules and, in turn, the conditions of these that will be mutated is somewhat similar to the process of the previous level (see Algorithm 10, lines 2 to 9). To determine if the comparison operator and/or the reference value of a condition will be mutated, a value p between 0 and 1 is calculated randomly. The comparison operator is changed when p is less than or equal to 0.33 or greater than or equal to 0.66. And the reference value is only changed when p is greater than 0.33. This process can also be visualized in Fig. 3.13 and Algorithm 11.

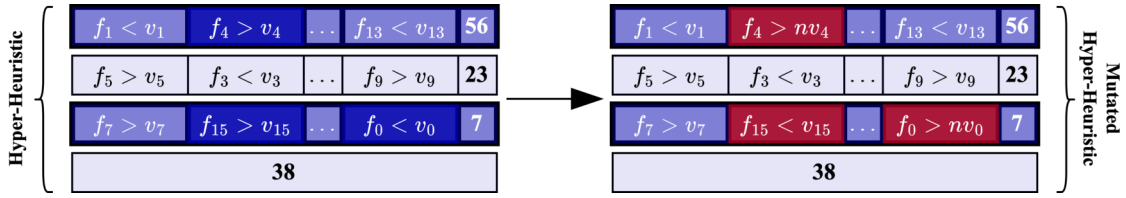

Figure 3.13: Mutation operator at operator level.

Algorithm 11 Mutation operator at operator level

Require: hh ▷ Hyper-heuristic to be mutated

1: **function** MutationOperators(hh)
2: $M \leftarrow$ Size(hh)
3: $k \leftarrow$ RandomInt(1, $M/2$)
4: $rules \leftarrow$ kRandomRulesFrom(hh, k)
5: **for each** $rule \in rules$ **do**
6: $rule_m \leftarrow rule$
7: $N \leftarrow$ Size($rule$)
8: $k \leftarrow$ RandomInt(1, $\frac{N-1}{2}$)
9: $conditions \leftarrow$ kRandomConditionsFrom($rule$, k)
10: **for each** $c \in conditions$ **do**
11: $p \leftarrow$ RandomFloat(0, 1)
12: **if** $p \leq 0.33$ **or** $p \geq 0.66$ **then**
13: $new\_c \leftarrow$ ChangeOperator(c)
14: **if** $p > 0.33$ **then**
15: $new\_c \leftarrow$ ChangeComparisonValue(c)
16: $rule_m \leftarrow$ ReplaceCondition($rule_m$, c, new_c)
17: $hh \leftarrow$ ReplaceRule(hh, $rule$, $rule_m$)
18: **return** hh

3.5.6 Selection

There are different methodologies for the selection of individuals (from a parent or child population) to determine the parent population of the next generation [79]. In any GA, the goal of individual selection is to lead the GA toward some optimal solution.

In this work, at the beginning of the evolutionary process, an initial population of n hhs is created and evaluated (Algorithms 3 and 4). This population enters a cycle of g generations, in the first generation this population acts as the parent population. Then, when applying the evolutionary crossover operator to this population, three hhs populations are created, one for each level to which it is applied (block, rules, conditions), and the union of these is called the child population, which has a size of $3n$. The mutation operator is applied over this population. For both the crossover operator and the mutation operator, the selection of hhs is done randomly. Then the hhs of the child population are also evaluated.

Once the evaluation is complete, the parent and child populations are mixed creating a temporary population of hhs. Then, the hhs within this population are sorted according to the fitness obtained with the genetic training group. Finally, the best n hhs from the temporary population are selected to be the new parent population. At this point, one generation has been completed, and the process repeats itself until a user-defined termination criterion is met.

When the termination criterion is met, the final solution is selected as the hh with the best fitness found in the last population. In such a way that, at this point, it is expected that the rules that make up such hh may be able to identify the optimal classification methods or close to them for unseen text datasets. This process is illustrated in the training phase of Fig. 3.3.

3.5.7 Termination

There are multiple ways to determine the termination criteria of a GA. Depending on the way in which we have decided to attack our problem, a simple criterion may be enough to find optimal solutions. In this work, it was decided to establish the number of generations as the termination criterion, since this criterion allows to always find this type of solution, and also provides enough time to inspect the solution space. Since the meta-features and the classification performances of the methods of the pool for each dataset of the genetic groups were calculated prior to the evolution process, the number of generations as the termination criteria is not capable of causing the GA to consume too much time.

3.5.8 Evaluation of the Best Individual

At the end of the evolutionary process, a hh is provided, which had the best fitness when evaluated with the genetic training group. At this point, it is necessary to know

how well it behaves with datasets that were not seen during the evolutionary process. The evaluation of the hh on an unseen dataset follows the same procedure shown in Algorithm 1. The result obtained with a single dataset is not enough to conclude that hh has very good generalization abilities, therefore, hh is evaluated with the genetic test group, a total of 34 datasets. For practicality, the F1 macro values of the pool classification methods and the meta-features for each of the datasets in the group were also calculated during the stage prior to the evolutionary process. In the end, an F1 macro value will be obtained for each dataset of the group, these results are averaged to know the general performance of the hh. If the hh has good generalization abilities, this value will be very close to the F1 macro values provided by the optimal classifiers for each dataset.

All the components of the evolutionary model were implemented in Python, using the math, nltk, numpy, random, sklearn, spicy, statistics, and textstat libraries.

3.6 Analysis

Within the evolutionary process, when evaluating each hh of a population in a certain generation, there may be rules in which their conditions will never be met, that is, unused rules. One option is to remove these rules during the evolutionary process because they may never be used in future generations, but there is no basis for saying that this will always be true, and it is also unknown how much this removal might interfere with the process. In these cases, the evolutionary operators play a very important role because of the way they were designed. When applying an evolutionary operator on a hh that has unused rules, there is a probability that these rules will be crossed or mutated, which, could turn them into relevant rules for the hh. Therefore, it was decided to keep the unused rules, since although, at one time they do not provide anything to the hh, over the generations they could become rules that provide useful information.

The evolutionary process or a single execution of the GA only outputs the hh with the best fitness within the population of the last generation. This leads to the following dilemma, a single execution of the GA may not be enough to give the verdict of a final solution to the problem, and also perhaps with this, it is not possible to define a relationship between the meta-features of a dataset and a specific classification method. It is not entirely correct to conclude that the result of a single execution is the final solution to the problem, since a GA is based on randomness, which allows the GA to have different starting points for each of its executions, and there is not something that ensures a convergence towards the same solution. Besides that, the creation of the initial population and its respective individuals, and the application of the evolutionary operators are also based on randomness.

The hh creation methodology allows for the possibility of one rule being specific to one dataset, while another rule may be essential for multiple datasets. This is possible because each of the rules in a hh can contain one or more conditions. A

rule with a single condition is very likely to be applied to multiple datasets since this is not as specific compared to a rule with two or three conditions. The objective of allowing more than one condition for each rule is to give the GA more freedom in the large space of solutions, which in turn allows better rules for the hhs. Going back to the example mentioned above, a rule with a single condition has a high probability of being applied to multiple datasets, but the action associated with this rule may not be able to obtain good performance for all of those datasets. On the other hand, rules with a larger number of conditions may only be applied to a smaller number of datasets, but the action associated with it may be able to provide optimal or close to optimal classification performance since they are more specific rules. This freedom that a rule can be defined by one or more conditions, in addition to allowing range checks (e.g dptAvg > 70 AND dptAvg $< 11,618$) allow the GA to find relationships between meta-features, for example, that there is a correlation between the meta-features dptMed and cmxWds.

This is also achieved in large part to the large number of datasets that are being used to address this problem, since otherwise we could have encountered various circumstances such as those mentioned below. First, in addition to using a small number of generations, a small training group of datasets would not work properly enough to define the reference value(s) of the conditions of a rule, which would not allow for discriminating the datasets very well at the moment of being evaluated. Second, a small number of datasets means very few instances of the general problem, so in addition to being biased, the creation of hhs with overfitting would be very likely (a single rule for a single dataset). As well as the case in which a single classification method predominates as the most optimal for most of the datasets. A classification method that predominates would avoid finding hhs with good generalization capabilities, since when handling randomness there may be a case in which none of the if rules of a hh are fulfilled, and therefore, the action associated with the else-rule is the one that has to be applied in all the datasets. If this action is the predominant classification method in the small group, the hh will be taken as a good hh without knowing that it is actually a bad hh in which the action of the else rule is always applied. In conclusion, using a small group of datasets would be skewing the problem, it would avoid exploiting the capabilities of a GA, and also, such a group would not be useful to determine if a hh has a good generalization to solve the general problem.

We conclude that taking the best hh obtained from a single execution of the GA as the final solution is not the best decision, at least if a deep analysis of it is not carried out. To achieve a more precise approach to determine the final solution to the problem, a statistical analysis of one or multiple runs is carried out. This also allows us to understand the internal functioning of the GA in terms of the development of the hhs, as well as their behavior. Therefore, the behaviors of the hhs populations and the final hh can be better explained. This analysis can be applied to one or multiple independent executions of the evolutionary model.

3.6.1 Single Run

For a single execution of the evolutionary model, the analysis is focused on visualizing the behavior of the evolution of the hhs. With an analysis of this type, information can be obtained about the moment in which the population converges, that is, that both the worst hh and the best hh have similar behavior. In some cases, this type of analysis is used to determine the number of generations in which a GA will be working, because with it it is possible to see the moment in which the fitness of the individuals does not achieve improvements.

According to the methodology designed for the evolutionary process, the hhs of all populations are evaluated after applying the evolutionary operators (except for the initial population that is evaluated at the beginning). Thus, the fitness values of each new parent population can be accessed. From these values, the maximum, average, and minimum aptitudes of such a population in each of the generations are determined. These three values are stored in a text file to later be consulted and plotted with the matplotlib module in Python. In this way, the general behavior of the population in each generation can be analyzed. This analysis is also included by default when doing multiple runs.

3.6.2 Multiple Runs

A run of the evolutionary model does not consume too much time due to the stage before the evolutionary process, in which the F1 macro values were calculated for each of the datasets of the training and test genetic groups with each of the classification methods of the pool. In addition to the extraction of the set of 16 meta-features for the training parts of such datasets. This allows various analyzes of multiple runs to be carried out without much delay.

If n runs of the GA are carried out, n hhs are obtained, and also n text files, which contain the behavior of the population during g generations, this belongs to the analysis of a single run but is added by default when dealing with multiple runs. Added to this, the frequency of the use of each of the rules is calculated and, in turn, the frequency of the actions associated with them for each of the final hhs. Due to the fact that the unused rules are not eliminated in the evolutionary process, there may be the possibility of finding this type of rules at the end of such a process, they will have a frequency of use of 0, and therefore, the classification methods associated with them they are not considered when calculating the frequency of actions.

In addition to the above, the set of the fitness of the last population of each run of the evolutionary model is used to calculate the five-number summary, and thus obtain: the minimum, first quartile, median, third quartile, and maximum of the aptitudes of such population. The mean value and standard deviation are also calculated for each of these populations. These values allow for a more detailed analysis of each of the hhs, in such a way that it can be observed when a hh has an outlier fitness value. This also allows knowing how each of the datasets are being

associated with a specific rule. With the calculation of the frequency of the actions, it is easy to determine which classification methods stand out the most, that is, if one approach is really superior to another in certain datasets.

Certain statistics are calculated from the multiple hhs obtained and from the frequencies mentioned above. For a total of n executions of the evolutionary model, the frequency of the actions associated with the rules of each hh is obtained (rules without use are not contemplated). Similarly, the frequency of occurrences of each of the 16 meta-features in rules that are used by hhs is also calculated. Finally, for each meta-feature, it is calculated how many times it is associated with each of the comparison operators ($<$ and $>$). And also, the mean, median, and standard deviation values are obtained from the list of reference values that are extracted according to the comparison operator. These latter statistics are calculated separately for when the rules are used and when they are not.

3.6.3 Computational Time

In a run of the evolutionary model, various components are involved with respect to the computational time required to find a final hh. Within this set, the evaluations of hhs have the greatest impact, since to know the fitness of a hh it is necessary to train and test a particular classification method for each of the datasets of the genetic group. In order to know the theoretical limits (in terms of time) for a hh to provide a solution to a group of datasets or even the limits of the computational time that a run of the evolutionary model can consume to provide a final hh, first, the training and test times were added for each of the classification methods with each of the datasets in the group, in the form (classification method, dataset). In such a way that the theoretical limits are established according to the sums of the pairs (classification method, dataset) when a single classification method is applied to all the datasets of the group, that is, the solutions of a hh for a group of datasets is always to apply the same classification method. In the case of an execution of the evolutionary model, the theoretical times are the same, since most of the training and testing of classification methods would take place in the evaluation of hhs of the first population, that is, when a hh needs to train and test a method A with a dataset X, the result could be stored and used in case another hh needs to perform the same operation.

The comparison of a final hh with other systems can be done using the classification performance, but also the computational time. This comparison consists of the computational times consumed (in minutes) by a final hh and the systems to provide a solution and classify each of the datasets in a genetic test group. The computational time consumed for each dataset is defined as the sum of three times: processing, training, and testing. Each of these times is relevant since both the final hh and each of the systems use a different approach to classify each dataset. For example, for processing, a hh uses a tf-idf transformation, on the other hand, other

systems can use various techniques to process and transform the information, and in some cases these techniques maybe involve DL, thus, the difference between the times consumed within this stage can be significant.

Chapter 4

Results

This chapter presents the results obtained from this work. The developed codes were executed using the Spyder environment [1]. The configuration of the evolutionary model was determined experimentally since the GAs have the characteristic of being very robust to changes in the values of its parameters, that is, small changes in its parameters do not seriously affect its performance. Later, it will be observed that the evolutionary model presents very interesting results. The configuration of the evolutionary model is shown in Table 4.1, in which the parameters and their respective assigned values are shown. The results that are presented later correspond to this configuration of the evolutionary model. The mutation probability is high compared to what is commonly used in the literature [79]. The management of a relatively high probability is in order that the four types of mutation have a greater chance of being applied and, therefore, there is a greater diversity of hhs within each population.

Parameter	**Value**
Population size	50
Number of generations	100
Maximum number of rules	16
Minimum number of rules	2
Maximum number of conditions	4
Minimum number of conditions	1
Mutation probability	20%

Table 4.1: Values of model's parameters for experimenting.

[1] Available at: https://www.spyder-ide.org/

With these parameters, the initialization of the first population of the evolutionary model is performed at random. Initialization is based on a probability distribution. A hh can be made up of a number of rules between 2 and 16, any number of rules within this range has the same probability of being selected. The number of conditions of a rule can be 1, 2, 3, or 4, this value is chosen at random, in such a way that any of these has the same chance of being chosen. When creating a condition of a rule, any of the 16 meta-features have the same probability of being selected, and this also happens when choosing one of the two comparison operators defined in Chapter 3.

Under this design of the evolutionary model, there is no preference for large or small hhs, as there is no preference for complex (greater number of conditions) or simple (one or two conditions) rules. If for the solution of the problem it is necessary to have a hh with many or few rules, as well as rules with few or many conditions, the same evolutionary model will be in charge of developing these solutions. Regarding the evolutionary operators, the three strategies of the crossover operator (at block, rule, and condition levels) are always applied, that is, there is no selection probability to decide which strategy to apply. In contrast, all four variants of the mutation operator have an equal chance of being chosen to apply to a given hh.

In order to carry out the analysis for one or multiple runs of the evolutionary process (see Chapter 3), we have divided the overall experiments into two phases. The first part corresponds to an analysis of 100 independent runs of the evolutionary process using the genetic training group, which results in a set of 100 hhs, which were the best in each of the corresponding runs. In the second part, based on the 100 hhs obtained, the best hh within this set was selected as the final solution, in such a way that this hh was evaluated with the genetic test group to determine its generalization capabilities. Finally, the results of this hh are compared against the optimum obtained by the best classification methods for each of the datasets in the group. And also against those obtained by two state-of-the-art AutoML systems for text classification using the same group.

For the first part of the experiments, because they are independent executions, the evolutionary process starts from a different point for each execution, although it is expected that when the termination criterion is reached, similar solutions will be reached in terms of fitness. In Fig. 4.1, section (a), the fitness of the best hhs found in each of the executions of the evolutionary process can be observed. The fitness of the best hh within the set is highlighted enclosed in a black circle, this hh was obtained in run number 56 with a fitness of 0.6790. This fitness is the average macro F1 value obtained for each of the datasets of the genetic training group. From the obtained set of 100 hhs, an average fitness of 0.6739 was found, a standard deviation of 0.0020, a median of 0.6739, a minimum of 0.6639, and a maximum of 0.6790, this can also be visualized in Fig. 4.1, in graph (b), which corresponds to a boxplot that summarizes the aforementioned.

Figure 4.2 presents a box plot that summarizes the times it would take a hh

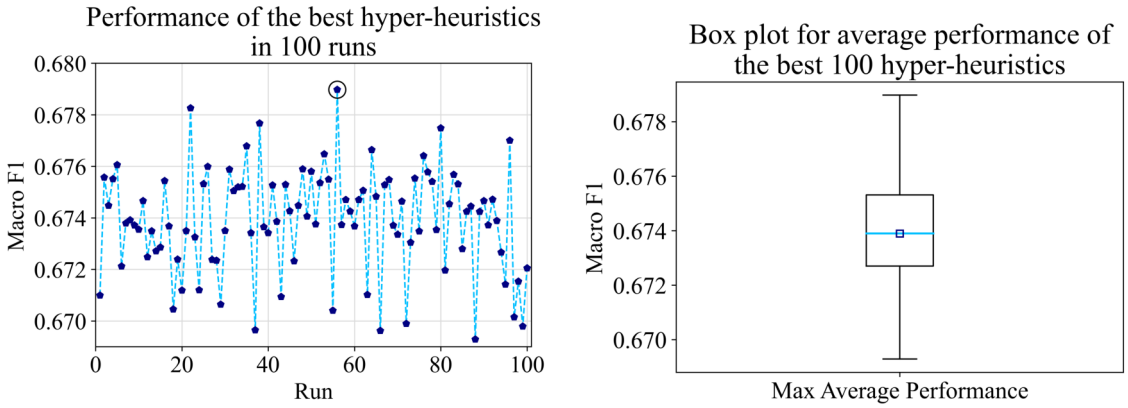

Figure 4.1: Performance of the best hhs from 100 independent runs: (a) Fitness of each best hh, (b) Boxplot for the best fitnesses.

to apply a single classification method to all datasets in the genetic train group, using the 60 classification methods in the pool. The multiple outliers of the boxplot (found at the top) mean that there are very expensive methods within the pool, which correspond to instances of the SVM method when using different kernels (polynomial, sigmoid, or radial basis function). In the figure, the values serve as a reference to determine the expected time of a run of the evolutionary model, giving the following summary (in minutes): a minimum of 0.15, a maximum of 3,929, and a median of 9.04.

The behavior of the hhs (in terms of fitness) during the evolutionary process corresponding to run 56 can be observed in Fig. 4.3. This execution has provided the best hh for the first part of the experiments. The evolutionary process lasted 100 generations, following the configuration shown in Table 4.1. This figure shows the fitness of the best and worst hh during each of the 100 generations, as well as the average fitness of the hhs of each of the populations. In this run, the convergence is reached around generation number 30, in such a way that the worst and the best hh have similar behavior to the passage of the following generations. This figure also gives us an idea about how the population behaves in other runs, in which we find that the evolutionary behavior is similar.

This hh (the one obtained in run 56) is presented as the final solution to the problem. The rules that make up this hh are shown in Fig. 4.4. The hh is made up of 7 rules (including the else-rule), in turn, the if-then rules are made up of 1, 2, or 3 conditions and involve 6 meta-features from the original set of 16 meta-features. According to this solution of the evolutionary model, classification methods such as MNB, BNB, KNN, and DT are not the most suitable for text classification tasks (at least for those addressed with datasets of both genetic groups), since they are not

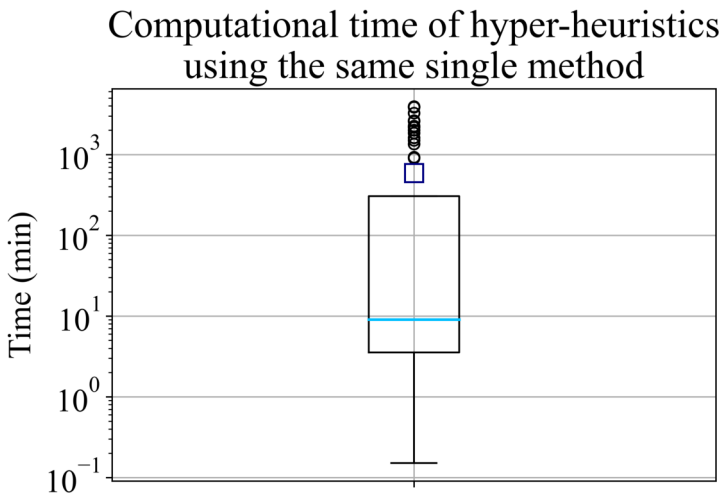

Figure 4.2: Computational time (in minutes) of hhs that use the same single method for classifying all the datasets in the genetic training group.

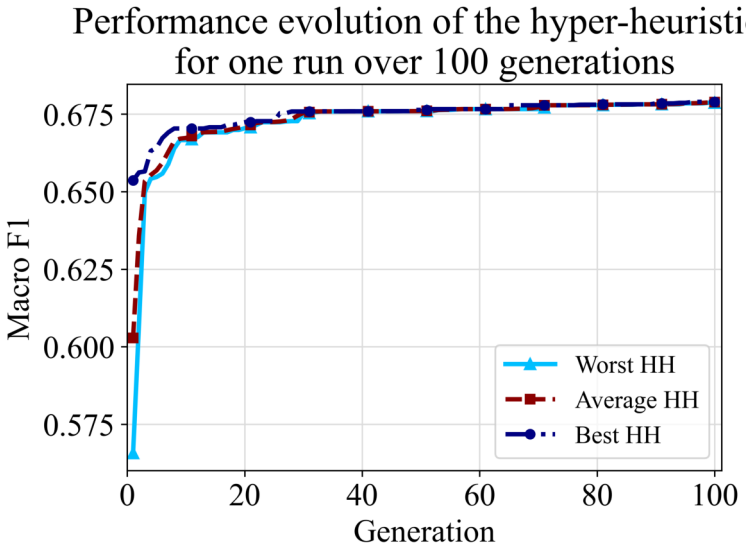

Figure 4.3: Performance of the hhs population over 100 generations.

associated with any of the rules of this hh. So, it seems that the methods found in the final solution rules (CNB, SVM, LR, and LSVM) seem to be superior to those mentioned above, that is, in terms of performance the first set of models classification is dominated by the latter. Analyzing more thoroughly the hh and its rules, within

the conditions that make up each of these, we can see that nDocs (the total number of documents within a dataset) is the most relevant meta-feature within the hh since it appears in a total of 4 rules. The meta-feature for the standard deviation of the number of words per document (wpdStd) only appears once, in Rule 1. Regarding the other 4 remaining meta-features, the Shannon entropy of documents per category (dptEnt), the median of words per document (wpdMed), the ratio between dptStd and dptAvg (dptStdAvg), and variance captured by the first 10 components of PCA (pca10), each one appears in two rules. In none of the rules is there a range check (that a meta-feature is being evaluated on more than one occasion), which indicates that it is not necessary or relevant to evaluate a dataset.

Rule 1: **if** *wpdStd* < 134.069 **and** *dptEnt* < 0.9378 **and** *wpdMed* < 12.027 **then**
CNB norm=False
Rule 2: **if** *dptStdAvg* < 0.8490 **and** *nDocs* > 5076.332 **then**
SVM C=10, ke='rbf'
Rule 3: **if** *dptStdAvg* < 0.1399 **then**
LR C=10, solver='liblinear'
Rule 4: **if** *nDocs* > 29463.055 **and** *pca10* < 0.5222 **and** *wpdMed* < 57.891 **then**
LR C=10, solver='lbfgs'
Rule 5: **if** *dptEnt* < 0.9378 **and** *nDocs* > 5076.332 **then**
LSVM C=1, l='s_hinge'
Rule 6: **if** *nDocs* > 29463.055 **and** *pca10* < 0.5848 **then**
LSVM C=1, l='s_hinge'
Rule 7: **else**
LSVM C=10, l='hinge'

Figure 4.4: Best hh selected as a final solution and obtained from 100 independent runs.

The second part of the general experiment consists of knowing the general performance of the evolutionary model. To carry this out it is necessary to compare, in this case, the selected final solution against the general optimal solution that is obtained using the genetic test group. This optimal solution is determined by finding for each dataset of the genetic test group its optimal classification method, that is, the one that provides the best classification performance according to the F1 macro evaluation metric. The classification methods considered are those belonging to the pool of classifiers (see Table 3.3). Table 4.2 presents the optimal classification methods for each dataset of the genetic test group, together with the F1 macro performance achieved. When analyzing these results, it can be observed that there is a dominance by the SVM classification methods with different kernels with a total of 30 appearances, of which; 10 belong when using a linear kernel (LSVM); 10 with sigmoid kernel; 9 with rbf kernel; and 1 with polynomial kernel. For the remaining datasets, the LR and Naïve Bayes (CNB and MNB) classification methods appear as

the optimal methods. Once these optimal classification methods have been found, we have observed that many of them are the actions associated with the rules of the hh chosen as the final solution, shown in Fig. 4.4, among which stand out SVM methods with rbf kernel; and LSVM with s_hinge as loss function. The last line of the table shows the average performance of the optimal classification methods for the genetic test group datasets, which is 0.6893, corresponding to the general optimal solution. This optimal value is used to compare the performance of the final selected hh obtained using the same genetic test group.

Dataset	**Classifier**	**Macro F1**	**Dataset**	**Classifier**	**Macro F1**
20ng	LSVM C=10, l='s_hinge'	0.8814	NYTAND	SVM C=10, ke='sigmoid'	0.3772
AGNews	SVM C=1, ke='poly', de=2	0.9130	NYTATM	LSVM C=10, l='s_hinge'	0.4608
CNNAC	SVM C=10, ke='sigmoid'	0.6733	oh	LSVM C=10, l='hinge'	0.6306
CNNAS	SVM C=10, ke='sigmoid'	0.5144	r8	LSVM C=1, l='s_hinge'	0.9709
CorTws	SVM C=10, ke='rbf'	0.5620	r52	SVM C=10, ke='sigmoid'	0.7033
csdmc	SVM C=10, ke='sigmoid'	0.9687	RFJob	LSVM C=10, l='hinge'	0.8539
CybTws	SVM C=1, ke='sigmoid'	0.8297	Rotten	SVM C=10, ke='rbf'	0.5071
DisTws	LSVM C=1, l='s_hinge'	0.7863	sen_pol	CNB norm=False	0.7418
F&RNS	SVM C=1, ke='sigmoid'	0.4885	StOvQR	SVM C=10, ke='rbf'	0.7972
gopds	SVM C=10, ke='rbf'	0.5906	SuiDect	SVM C=10, ke='rbf'	0.9376
HRCB	SVM C=10, ke='sigmoid'	1.0000	TMACS	SVM C=1, ke='rbf'	0.8704
HRCS	SVM C=10, ke='sigmoid'	0.0975	TMAMt	SVM C=10, ke='rbf'	0.8630
imdbs	MNB	0.7300	TMASt	LSVM C=1, l='hinge'	0.8399
IMDBR	SVM C=10, ke='rbf'	0.8998	TripAd	SVM C=1, ke='sigmoid'	0.5471
LvsC	SVM C=10, ke='rbf'	0.7166	wipo_l1	LSVM C=1, l='s_hinge'	0.4508
movies	CNB norm=True	0.8299	wipo_l2	LSVM C=1, l='s_hinge'	0.1394
NewsCat	LR C=10, solver='saga'	0.4745	yelp	LSVM C=1, l='s_hinge'	0.7895
Average					**0.6893**

Table 4.2: Optimal classification methods and their macro F1 performance for the datasets of the genetic test group.

One of the objectives of this work is to find a hh that is capable of achieving a value equal to or very close to the optimum. Fig. 4.5 shows the performance of the hh on the datasets of the genetic test group. In this case, the hh achieves a performance of 0.6805 by averaging the F1 macro values obtained in each of the datasets. This obtained value is very close to the optimal value of 0.6893, shown in Table 4.2. When analyzing the various behaviors with each of the datasets, it can be seen that for the hh, there were a large number of datasets in which what could be said as good performances were achieved by obtaining F1 macro values greater than 0.8. Within these, the HRCB dataset was the easiest to classify, obtaining a macro F1 of 1.0. On the other hand, the datasets with a poor classification rate were HRCS and

wipo_l2, obtaining F1 macro values below 0.2. According to the values of the meta-features, for these last two datasets, the meta-feature of the mean of documents per category (dptAvg) presents the lowest values of the entire group of datasets. On the other hand, for the HRCB dataset (the best classification performance achieved), the meta-feature of the variance captured by the first 30 components of PCA (pca30) presents the highest value within the same group. In this work, Pearson's correlation is applied in order to explore the possible relationships that could exist between the meta-features and the classification performance for each dataset. This correlation is calculated by means of the values of the meta-features used in the hh and the F1 macro values obtained by this hh for each dataset of the genetic test group. According to Pearson's correlation, the meta-features dptStdAvg and dptEnt, which deal with the dispersion of the data and the amount of information per category, present a negative correlation with the classification performance, with values of -0.49 and -0.59 respectively. This allows us to deduce that when the categories of a dataset are compact, it will tend to be easier to classify.

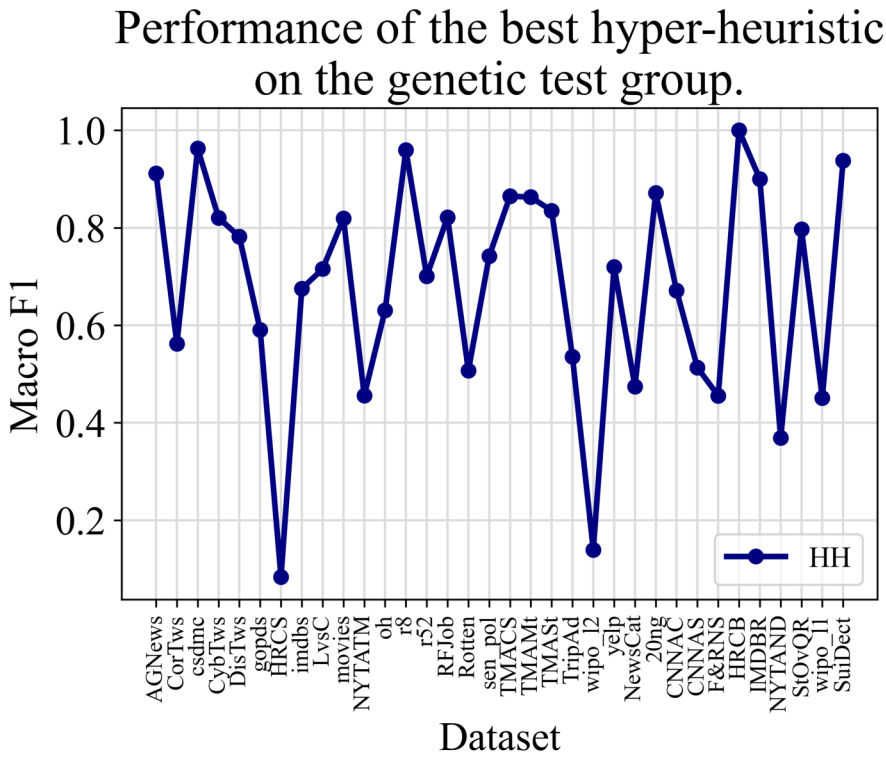

Figure 4.5: Behavior of the best hh on the genetic test group.

In order to understand the behavior of the final selected hh on the datasets of the genetic test group, Table 4.3 shows the rule of the hh that is activated for each of the datasets, the method of classification associated with such a rule and the F1 macro

value obtained by applying such a method. Table 4.4 shows how many times each of the hh rules were triggered. In both tables, it can be seen that rules 2, 7 (else-rule), and 1 are the most used by hh, with frequencies of 12, 11, and 5 respectively. The classification methods associated with these rules are different from each other, since for rule 2, the SVM method with C=10 and the rbf kernel is used; for rule 7, LSVM with C=10 and hinge as loss function; and for rule 1, CNB without normalization. As previously mentioned, in this final hh, there are classification methods that are optimal for some of the datasets of the genetic test group (see Table 4.2). In such a way that it has been found that of the 34 datasets of the group, the hh is capable of selecting the optimal classification method for 10 of these datasets. With the rest of the datasets, the hh is selecting very close to optimal classification methods, since the overall performance of the hh is very close to the overall optimum. In short, this final selected hh has the ability to select suitable classification methods for various datasets.

Dataset	**#Rule**	**Classification method**	**Macro F1**	**Dataset**	**#Rule**	**Classification method**	**Macro F1**
20ng	3	LR C=10, solver='liblinear'	0.8714	NYTAND	7	LSVM C=1, l='hinge'	0.3692
AGNews	2	SVM C=10, ke='rbf'	0.9114	NYTATM	7	LSVM C=1, l='hinge'	0.4560
CNNAC	7	LSVM C=1, l='hinge'	0.6712	oh	7	LSVM C=1, l='hinge'	0.6306
CNNAS	7	LSVM C=1, l='hinge'	0.5132	r8	7	LSVM C=1, l='hinge'	0.9597
CorTws	2	SVM C=10, ke='rbf'	0.5620	r52	7	LSVM C=1, l='hinge'	0.7007
csdmc	7	LSVM C=1, l='hinge'	0.9630	RFJob	5	LSVM C=1, l='s_hinge'	0.8216
CybTws	2	SVM C=10, ke='rbf'	0.8204	Rotten	2	SVM C=10, ke='rbf'	0.5071
DisTws	1	CNB norm=False	0.7820	sen_pol	1	CNB norm=False	0.7418
F&RNS	7	LSVM C=1, l='hinge'	0.4556	StOvQR	2	SVM C=10, ke='rbf'	0.7972
gopds	2	SVM C=10, ke='rbf'	0.5906	SuiDect	2	SVM C=10, ke='rbf'	0.9376
HRCB	7	LSVM C=1, l='hinge'	1.0000	TMACS	2	SVM C=10, ke='rbf'	0.8649
HRCS	7	LSVM C=1, l='hinge'	0.0834	TMAMt	2	SVM C=10, ke='rbf'	0.8630
imdbs	1	CNB norm=False	0.6753	TMASt	2	SVM C=10, ke='rbf'	0.8350
IMDBR	2	SVM C=10, ke='rbf'	0.8998	TripAd	2	SVM C=10, ke='rbf'	0.5353
LvsC	1	CNB norm=False	0.7156	wipo_l1	6	LSVM C=1, l='s_hinge'	0.4508
movies	3	LR C=10, solver='liblinear'	0.8193	wipo_l2	6	LSVM C=1, l='s_hinge'	0.1394
NewsCat	4	LR C=10, solver='lbfgs'	0.4745	yelp	1	CNB norm=False	0.7199

Table 4.3: Relationship between datasets and the rules of the best hh.

In some cases, there is the idea that a classification method with a certain configuration will always be better than others in all the problems that arise, which is not true, as we have explained in previous chapters. Consequently, we have also compared the performance of the selected hh with the performance of the 60 classification methods of the pool of classifiers. Fig. 4.6 shows the average performance of each classification method on the datasets of the training and test genetic groups. According to the graph in the figure, it can be seen that methods 35 to 45 (with the exception of 40 and 41), which correspond to LR and LSVM methods, are those with the best average performance in both genetic groups. The best classification

Rule	Frequency
Rule 1	5
Rule 2	12
Rule 3	2
Rule 4	1
Rule 5	1
Rule 6	2
Rule 7	11

Table 4.4: Frequency of use of rules of the best hh.

method within these is the number 42; a LSVM method with C=1 and s_hinge as loss function. With the genetic test group, this classification method achieves an average performance of 0.6747, being a lower performance than that obtained with the final hh (0.6805). Although at first glance the difference seems negligible, it can be significant in statistical terms. In order to know if this difference between the performance of the classification method 42 and the final hh is really significant, the Wilcoxon signed-rank test [80] was performed. This test assumes the null hypothesis that both classification methods (method 42 and the hh) perform equally well on N observations. In this case, to perform a test with a significance level $\alpha = 0.05$ and $N = 34$ (number of datasets/observations), the reference value extracted from the exact critical values table is 182. When finished the test results a value of $z = 100$, which, being less than 182, allows us to reject the null hypothesis and conclude that this small numerical difference between both performances is statistically significant, therefore, the behavior of the hh is better. In other words, the use of a specific classification method according to the type of dataset produces better classification performances than always using the same classification method.

Once it was confirmed statistically that the final hh is better than any individual classification method, we compared the performance of this hh with the performances that can be achieved by two state-of-the-art AutoML systems for text classification: AutoKeras [2] and AutoGluon [3]; of which we have explained how it works in Section 1.3. In short, both systems are based on DL methods, that is, selection, ensemble, or fine-tuning (depending on the operation of each system) only takes into account methods of this type. These systems are characterized by contemplating various word embeddings (such as word2Vec or fastText) and transformer-based methods (such as BERT or ALBERT) to define their search space. Our objective

[2]Available at: https://autokeras.com/
[3]Available at: https://auto.gluon.ai/stable/index.html

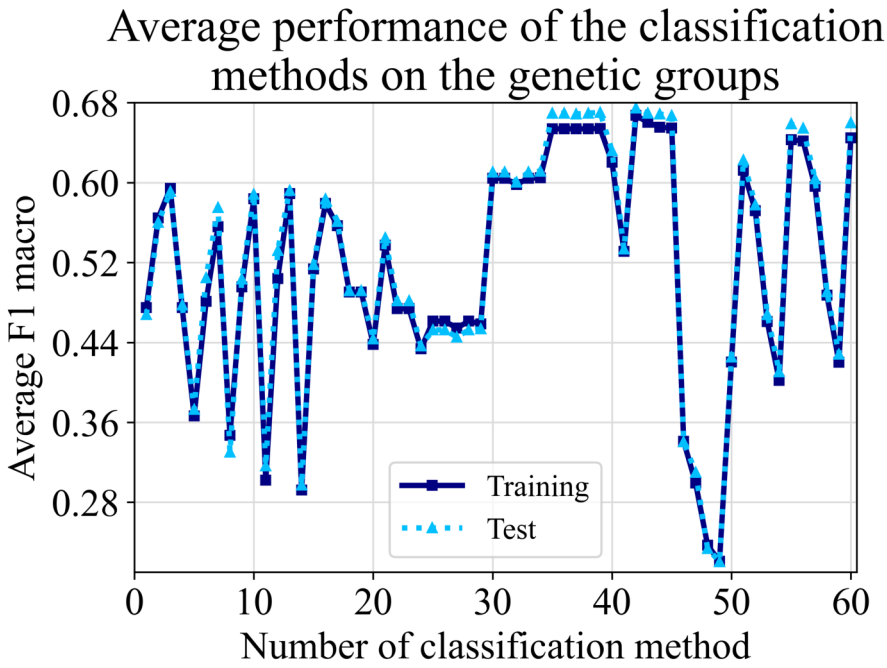

Figure 4.6: Average performance of each classification method in the datasets of each genetic group.

when using these systems is to see their ability to find the optimal classification method (according to their methodology and solution space) for each dataset of the genetic test group. For this experiment, some datasets in the group may be difficult to classify, so both systems were left with their default settings so as not to limit their search space. In this case, for using these AutoML systems we had to resort to using a server with 2 Intel Xeon Gold processors @2.6 GHz, 256 GB of RAM, a Tesla V100 GPU with 32 GB of RAM, and Windows Server 2016. Fig. 4.7 and Table 4.5 show the behavior of the selected hh, AutoGluon, and AutoKeras in each of the datasets of the genetic test group, in terms of macro F1. In these results, it can be seen that the AutoGluon system found a solution for each of the 34 datasets. On the other hand, the AutoKeras system was only able to provide solutions for 24 datasets, since for the other datasets this system crashed due to different problems, such as lack of memory in some cases. Likewise, it can also be observed that the difference in performance achieved by the three models on each of the datasets is small in most cases. In the CorTws, DisTws, and sen_pol datasets, the AutoKeras and AutoGluon systems provide better classification performances than those of the hh; on the other hand, the hh has higher performances in some datasets such as NYTATM, oh, r8, among others.

The average performance of AutoKeras over the 24 datasets (for which only

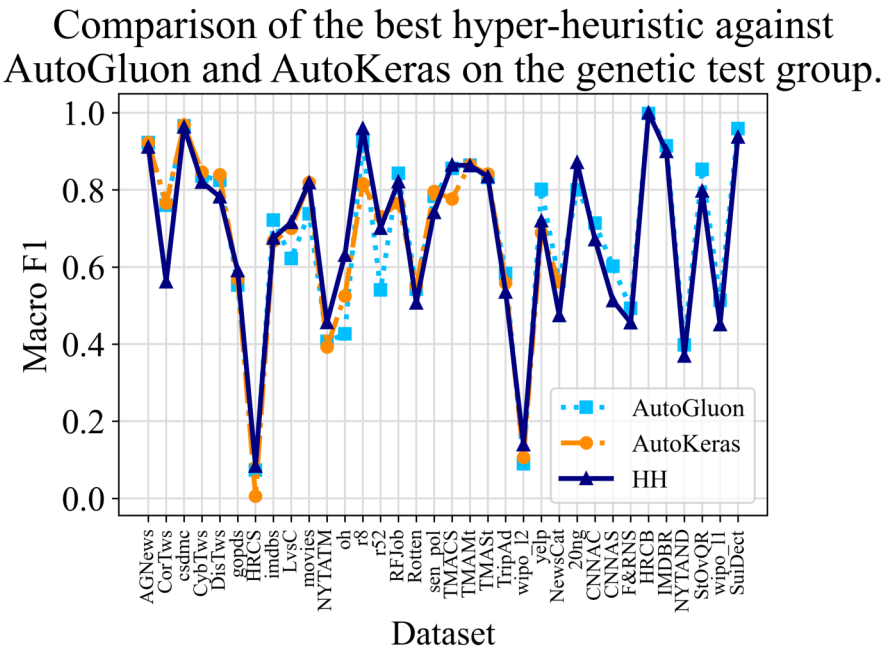

Figure 4.7: Comparison of the best hh against AutoGluon and AutoKeras.

results could be obtained) is 0.6700; instead, the hh achieves an average performance of 0.6739 on these datasets. On the other hand, when taking into account the 34 datasets, AutoGluon has an average performance of 0.6860, and the hh with a performance of 0.6805. The results shown in Fig. 4.7 and Table 4.5 together with the average performance obtained allow us to believe that the final hh behaves very similarly to the AutoKeras and AutoGluon systems. In order to know if this assumption is correct, we decided to apply the Wilcoxon signed-rank test again to provide more valid support for the differences that exist between them. Similar to the previous test, we start from the null hypothesis in which each AutoML system performs just as well as the hh. We use a significance level $\alpha = 0.05$ for the two tests (hh-AutoKeras and hh-AutoGluon); the reference values extracted from the tables is 81 for AutoKeras (with $N = 24$ datasets), and 182 for AutoGluon (with $N = 34$ datasets). The hh-AutoKeras test returns $z = 135$; while for the hh-AutoGluon test, the result is $z = 240$. These results are greater than their respective reference values, therefore, the null hypothesis cannot be rejected, which means that the final hh behaves just as well as either of the two AutoML systems.

In addition to the comparison of the final hh with both AutoML systems in terms of performance, the comparison in terms of time also brings interesting and relevant information. The results of this comparison are shown in Fig. 4.8. Because some datasets were much faster to classify than others, a logarithmic scale was used, which allows these differences to be observed more compactly. In general, the final hh

Dataset	Best hh	AutoGluon	AutoKeras	Dataset	Best hh	AutoGluon	AutoKeras
20ng	**0.8714**	0.8003	-	NYTAND	0.3692	**0.3984**	-
AGNews	0.9114	**0.9229**	0.9226	NYTATM	**0.4560**	0.4070	0.3930
CNNAC	0.6712	**0.7141**	-	oh	**0.6306**	0.4263	0.5251
CNNAS	0.5132	**0.6025**	-	r8	**0.9597**	0.9250	0.8160
CorTws	0.5620	0.7604	**0.7660**	r52	0.7007	0.5408	**0.7308**
csdmc	0.9630	0.9663	**0.9685**	RFJob	0.8216	**0.8434**	0.7653
CybTws	0.8204	0.8362	**0.8462**	Rotten	0.5071	0.5424	**0.5505**
DisTws	0.7820	0.8249	**0.8398**	sen_pol	0.7418	0.7834	**0.7958**
F&RNS	0.4556	**0.4927**	-	StOvQR	0.7972	**0.8536**	-
gopds	**0.5906**	0.5533	0.5687	SuiDect	0.9376	**0.9592**	-
HRCB	**1.0000**	0.9981	-	TMACS	**0.8649**	0.8556	0.7768
HRCS	**0.0834**	0.0731	0.0055	TMAMt	0.8630	0.8643	**0.8650**
imdbs	0.6753	**0.7220**	0.6673	TMASt	0.8350	0.8322	**0.8411**
IMDBR	0.8998	**0.9146**	-	TripAd	0.5353	**0.5833**	0.5586
LvsC	**0.7156**	0.6220	0.7007	wipo_l1	0.4508	**0.5136**	-
movies	**0.8193**	0.7383	0.8192	wipo_l2	**0.1394**	0.0897	0.1063
NewsCat	0.4745	**0.5635**	0.5620	yelp	0.7199	**0.8012**	0.6892

Table 4.5: Results (macro F1) of the best hyper-heuristic against AutoGluon and AutoKeras on the genetic test group.

behaves much faster than both AutoML systems, that is, it provides the solutions to the group of datasets in a shorter amount of time. Taking the entire genetic test group (only 24 datasets for AutoKeras), the hh has a median of 0.29 minutes, AutoGluon 24.77 minutes, and AutoKeras 167.87 minutes. The most expensive system is AutoKeras since to classify the 24 datasets it requires a total of 18,357 minutes, compared to 1,007 minutes for hh, and 1,357 minutes for AutoGluon, the latter two taking into account all 34 datasets in the group. It is worth mentioning that AutoGluon allows setting a limit to find a solution for a dataset. However, this is not highly recommended as a very small value could cause the system to provide bad results for some datasets. Finally, it is necessary to highlight that the times consumed by the classification methods used by the hh were estimated using only the CPU, unlike AutoML systems, where those times were estimated using the dedicated GPU.

The comparison of the evolutionary model against state-of-the-art systems with the respective analysis previously carried out, allows us to say that the evolutionary model has been able to develop a final hh that can easily compete against these systems that are based on DL methods. This is very interesting because the actions associated with the final hh rules are classical ML methods, and yet the behaviors between the AutoML systems and the hh are very similar. This being the case, it

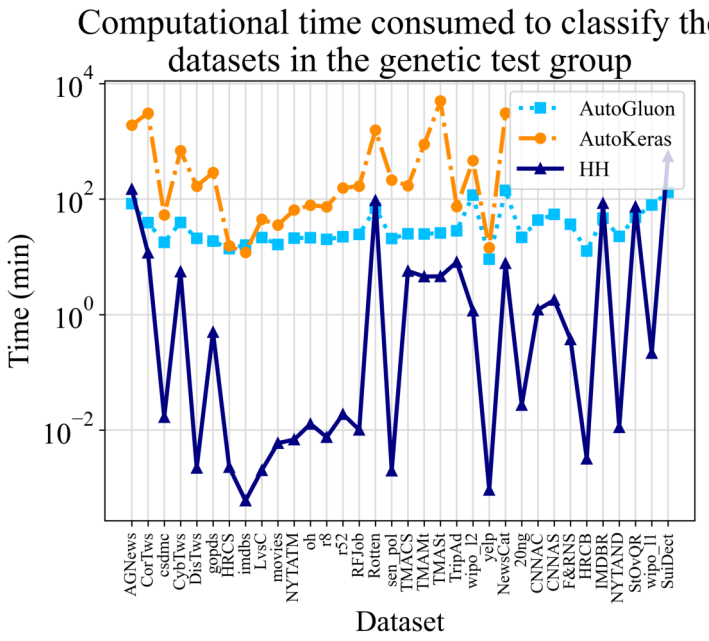

Figure 4.8: Comparison of computational times (in minutes) of the best hh against AutoGluon and AutoKeras.

can also be concluded that ML methods are still superior to DL methods for several problems or even text classification tasks, which has already been commented on by other researchers in the literature [44, 45]. The main causes of these behaviors are associated with the complexities of DL methods. For a DL method to achieve good classification performance, it requires large amounts of data during the training stage, in addition to large amounts of parameters that need to be tuned (such as epochs, layers, numbers of units, learning rates, etc.), which entails the need for large computational resources. In the work carried out in [44], the authors mention that it is more probable that a DL method can outperform an ML one when there are large amounts of data available for the training, but this does not ensure that the difference between the classification performances will be large, in some cases it is usually insignificant, unlike the computational costs, of which the differences could be very significant. The results shown in Fig. 4.7, Table 4.5, and the Wilcoxon signed-rank tests complement what was mentioned above, where it was possible to observe that the AutoGluon and AutoKeras systems (both based on DL methods), and the final hh (consisting of ML methods) have similar behaviors, even though ML methods do not require large amounts of computational resources to train and test. This also considering that the AutoML systems used a dedicated GPU, whereas the hh worked only with the CPU.

All this has led us to the following, the selection of an adequate classification method always depends on the problem, which can be observed with the hh. Our evolutionary model has provided new information on the relevance of meta-features (representing the data distribution) for the selection of suitable classification methods for a dataset in question.

Chapter 5

Conclusions

In this thesis work, we have presented an evolutionary model with the ability to learn hhs as sets of rules of the if-then form (including an else-rule), being able to generalize the method selection process for various problems of text classification. The method selection process is a problem that has been addressed in AutoML systems, where many of them do not have the ability to generalize the process, that is, they have the objective of finding the classification method with the best performance for a single problem. We have used a set of 16 meta-features to represent the data distribution of a dataset, based on statistics. Given a dataset, the rules of a hh evaluate its meta-features in order to select the most appropriate classification method with optimal performance or very close to it. In order to validate the solutions and the performance of our evolutionary model, we developed a series of experiments, divided into 3 phases. First, we carried out 100 runs independently of the evolutionary model in order to obtain 100 hhs from different starting points, using the same genetic training group of datasets in each run. Then, from the 100 hhs obtained, we select the one that achieved the best performance as the final hh. This hh was evaluated with a genetic test group to determine its generalization capacity with unseen datasets. Each genetic group consists of 34 datasets associated with various text classification tasks (e.g. sentiment detection, email filtering, among others). In the last phase of the experiments, we performed a comparison of the final hh performance with the results obtained by two state-of-the-art AutoML systems for DL-based text classification.

From the results obtained in each phase of the experiments, the following was concluded:

- The evolutionary model is capable of learning hhs that allow the selection of appropriate methods for various text classification problems, achieving very good and close to optimal classification performance for each dataset. During the training phase, hhs evolve to a point where they achieve performances very close to the general optimum of the datasets of the genetic training group.
- The method selection process has been generalized with the best hh obtained

since when it is tested with a group of unseen datasets (genetic test group) it obtains a performance very close to optimal.

- The LSVM, SVM, LR, and CNB classification methods with various configurations of hyperparameters dominate the rules in the final hh. Likewise, these classification methods also appear as some of the optimal methods for the datasets of both genetic groups (training and test).
- The Wilcoxon signed-rank test made it possible to discover that the difference in average performance, obtained with the final hh and the best classification method on the datasets of the genetic test group, is statistically significant. Giving, as a result, the behavior of the final hh is better.
- The final hh is capable enough to compete against two state-of-the-art AutoML systems since the average performances obtained with these three are very similar. The results of the Wilcoxon signed-rank tests guarantee this since there is no significant difference among their behaviors.
- The classical ML classification methods used in final hh rules have shown that despite being simpler, they can compete and even outperform in some cases the complex methods created by AutoML systems (using a DL approach).
- Of the set of 16 meta-features used to represent the datasets, those that represent the distribution of the data at document and category level are key to determining the appropriate classification methods for various datasets.
- The form of representation used for the rules of the hhs allows a better comprehension and understanding of how a hh determines which is the most appropriate classification method for a particular dataset.
- There is a correlation between the meta-features associated with the dispersion of the data and the amount of information at a level of documents per category (dptStdAvg and dptEnt respectively), with the classification performance obtained by the final hh. In such a way that these can serve as good indicators of the behavior that this hh will have with unseen datasets.

Finally, there are multiple directions for future research such as developing a multi-objective model. A model of this type has the purpose of not only considering the classification performance but also giving importance to the times (training and testing of the classification methods) and the computational resources required to achieve such performance. This is because DL methods can achieve better performance than ML methods on some problems but at higher computational time and cost. Developing a set of diverse solutions that contemplate different objectives is useful, since depending on the problem being addressed and the available resources, there are solutions that may be more appropriate than others. Other

research that could be developed in the future is according to an approach in which the selection of a classification method is at a lower level, that is, determining the most appropriate method for each document in a dataset, which leads to considering new meta-features that represent the data of a document. This would allow for forming customized solutions according to the provided dataset. In addition, new meta-features can be considered at different levels, some can be determined from word vectors or other forms of embeddings; part-of-speech features; as well as some based on the dataset content in general like n-grams, emojis/emoticons, or sentiment words. Finally, another research direction could be to develop a model that instead of selecting only one classification method, builds an ensemble of the most appropriate methods according to a hh for a given dataset.

Bibliography

[1] KR Chowdhary. *Fundamentals of artificial intelligence*. Springer, 2020.

[2] Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. Natural language processing: State of the art, current trends and challenges. *Multimedia tools and applications*, 82(3):3713–3744, 2023.

[3] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49, 1999.

[4] Jochen Hartmann, Juliana Huppertz, Christina Schamp, and Mark Heitmann. Comparing automated text classification methods. *International Journal of Research in Marketing*, 36(1):20–38, 2019.

[5] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning–based text classification: a comprehensive review. *ACM computing surveys (CSUR)*, 54(3):1–40, 2021.

[6] Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S Yu, and Lifang He. A survey on text classification: From traditional to deep learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(2):1–41, 2022.

[7] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[8] Andrea Gasparetto, Matteo Marcuzzo, Alessandro Zangari, and Andrea Albarelli. A survey on text classification algorithms: From text to predictions. *Information*, 13(2):83, 2022.

[9] Stijn Hoskens. Choosing the right text classifier: A meta-heuristic framework, 2015.

[10] Juan Carlos Gomez, Stijn Hoskens, and Marie-Francine Moens. Evolutionary learning of meta-rules for text classification. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 131–132, 2017.

[11] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, et al. Analysis of the automl challenge series. *Automated Machine Learning*, 177, 2019.

[12] Karansingh Chauhan, Shreena Jani, Dhrumin Thakkar, Riddham Dave, Jitendra Bhatia, Sudeep Tanwar, and Mohammad S Obaidat. Automated machine learning: The new wave of machine learning. In *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pages 205–212. IEEE, 2020.

[13] Hugo Jair Escalante. Automated machine learning—a brief review at the end of the early years. *Automated Design of Machine Learning and Search Algorithms*, pages 11–28, 2021.

[14] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

[15] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287*, 2019.

[16] Matthias Blohm, Marc Hanussek, and Maximilien Kintz. Leveraging automated machine learning for text classification: Evaluation of automl tools and comparison with human performance. *arXiv preprint arXiv:2012.03575*, 2020.

[17] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386, 2020.

[18] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[19] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[20] Forough Majidi, Moses Openja, Foutse Khomh, and Heng Li. An empirical study on the usage of automated machine learning tools. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 59–70. IEEE, 2022.

[21] Hugo Jair Escalante, Manuel Montes, and Luis Enrique Sucar. Particle swarm model selection. *Journal of Machine Learning Research*, 10(2), 2009.

[22] Siyu Huang, Xi Li, Zhi-Qi Cheng, Zhongfei Zhang, and Alexander Hauptmann. Gnas: A greedy neural architecture search method for multi-attribute learning. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 2049–2057, 2018.

[23] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*, pages 678–687. PMLR, 2018.

[24] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1304–1307. IEEE, 2016.

[25] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[26] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[27] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, volume 9, page 50. Citeseer Austin, TX, 2014.

[28] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015.

[29] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*, pages 1189–1232, 2018.

[30] Jorge G Madrid and Hugo Jair Escalante. Meta-learning of text classification tasks. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 24th Iberoamerican Congress, CIARP 2019, Havana, Cuba, October 28-31, 2019, Proceedings 24*, pages 107–119. Springer, 2019.

[31] Jorge G Madrid, Hugo Jair Escalante, and Eduardo Morales. Meta-learning of textual representations. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, pages 57–67. Springer, 2020.

[32] Jorge Madrid. Autotext: Automl for text classification, 2019.

[33] Washington Cunha, Sérgio Canuto, Felipe Viegas, Thiago Salles, Christian Gomes, Vitor Mangaravite, Elaine Resende, Thierson Rosa, Marcos André Gonçalves, and Leonardo Rocha. Extended pre-processing pipeline for text classification: On the role of meta-feature representations, sparsification and selective sampling. *Information Processing & Management*, 57(4):102263, 2020.

[34] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019.

[35] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[36] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[37] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[38] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

[39] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[40] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[41] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[42] Rushil Desai, Aditya Shah, Shourya Kothari, Aishwarya Surve, and Narendra Shekokar. Textbrew: Automated model selection and hyperparameter optimization for text classification. *International Journal of Advanced Computer Science and Applications*, 13(9), 2022.

[43] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[44] Washington Cunha, Vítor Mangaravite, Christian Gomes, Sérgio Canuto, Elaine Resende, Cecilia Nascimento, Felipe Viegas, Celso França, Wellington Santos Martins, Jussara M Almeida, et al. On the cost-effectiveness of neural and non-neural approaches and representations for text classification: A comprehensive comparative study. *Information Processing & Management*, 58(3):102481, 2021.

[45] Yaakov HaCohen-Kerner. Survey on profiling age and gender of text authors. *Expert Systems with Applications*, page 117140, 2022.

[46] Juan Carlos Gomez and Hugo Terashima-Marín. Evolutionary hyper-heuristics for tackling bi-objective 2d bin packing problems. *Genetic Programming and Evolvable Machines*, 19:151–181, 2018.

[47] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[48] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III: Third International Conference, PATAT 2000 Konstanz, Germany, August 16–18, 2000 Selected Papers 3*, pages 176–190. Springer, 2001.

[49] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. *Handbook of metaheuristics*, pages 449–468, 2010.

[50] John H Drake, Ahmed Kheiri, Ender Özcan, and Edmund K Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020.

[51] Jeffrey EF Friedl. *Mastering regular expressions*. " O'Reilly Media, Inc.", 2006.

[52] W John Wilbur and Karl Sirotkin. The automatic identification of stop words. *Journal of information science*, 18(1):45–55, 1992.

[53] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[54] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[55] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Madison, WI, 1998.

[56] Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 616–623, 2003.

[57] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes-which naive bayes? In *CEAS*, volume 17, pages 28–69. Mountain View, CA, 2006.

[58] Evelyn Fix and Joseph L Hodges Jr. Discriminatory analysis-nonparametric discrimination: Small sample performance. Technical report, California Univ Berkeley, 1952.

[59] Philip H Swain and Hans Hauska. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics*, 15(3):142–147, 1977.

[60] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[61] Martin JH Jurafsky D. *Speech and Language Processing*. Standford, 2023.

[62] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[63] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021.

[64] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[65] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[66] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.

[67] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[68] Jason Wang, Kaiqun Fu, and Chang-Tien Lu. Sosnet: A graph convolutional network approach to fine-grained cyberbullying detection. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 1699–1708. IEEE, 2020.

[69] Hadeer Ahmed, Issa Traore, and Sherif Saad. Detecting opinion spams and fake news using text classification. *Security and Privacy*, 1(1):e9, 2018.

[70] Dimitrios Kotzias, Misha Denil, Nando De Freitas, and Padhraic Smyth. From group to individual labels using deep features. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 597–606, 2015.

[71] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[72] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. *arXiv preprint cs/0409058*, 2004.

[73] Rishabh Misra. News category dataset. *arXiv preprint arXiv:2209.11429*, 2022.

[74] Issa Annamoradnejad, Jafar Habibi, and Mohammadamin Fazli. Multi-view approach to suggest moderation actions in community question answering sites. *Information Sciences*, 600:144–154, 2022.

[75] Md Hijbul Alam, Woo-Jong Ryu, and SangKeun Lee. Joint multi-grain topic sentiment: modeling semantic aspects for online reviews. *Information Sciences*, 339:206–223, 2016.

[76] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[77] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[78] Robert Gunning. The fog index after twenty years. *Journal of Business Communication*, 6(2):3–13, 1969.

[79] Michael Affenzeller, Stefan Wagner, Stephan Winkler, and Andreas Beham. *Genetic algorithms and genetic programming: modern concepts and practical applications*. Crc Press, 2009.

[80] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006.

Salamanca, Gto., a 7 de septiembre del 2023.

M. en I. HERIBERTO GUTIÉRREZ MARTIN
COORDINADOR DE ASUNTOS ESCOLARES
P R E S E N T E.-

Por medio de la presente, se otorga autorización para proceder a los trámites de impresión, empastado de tesis y titulación al alumno(a) Estrella Ramírez Jonathán de Jesús del ***Programa de Maestría en Ingeniería Eléctrica (Instrumentación y Sistemas Digitales)*** y cuyo número de ***NUA*** es: 145860 del cual soy director. El título de la tesis es: Evolutionary Learning of Selection Hyper-Heuristics for Choosing the Right Method in Text Classification Problems.

Hago constar que he revisado dicho trabajo y he tenido comunicación con los sinodales asignados para la revisión de la tesis, por lo que no hay impedimento alguno para fijar la fecha de examen de titulación.

A T E N T A M E N T E

Dr. Gómez Carranza Juan Carlos
DIRECTOR DE TESIS
SECRETARIO

Dr. Sánchez Yáñez Raúl Enrique
PRESIDENTE

Dr. Ruiz Pinales José
VOCAL