UNIVERSIDAD
DE GRANADA

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Máster Universitario en Ciencia de Datos e Ingeniería de Computadores

Master's Thesis

Development of Fair Machine Learning Algorithms based on Decision Trees

Presented by:
David Villar Martos

Supervising tutor:
Jorge Casillas Barranquero
Department of Computer Science and Artificial Intelligence

Academic year 2023-2024

Development of Fair Machine Learning Algorithms based on Decision Trees

David Villar Martos

David Villar Martos *Development of Fair Machine Learning Algorithms based on Decision Trees*. Master's Thesis. Academic year 2023-2024.

Supervising Tutor Jorge Casillas Barranquero
Department of Computer Science and Artificial Intelligence

Máster Universitario en Ciencia de Datos e Ingeniería de Computadores

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

University of Granada

Declaration of Originality

Mr. David Villar Martos

I explicitly declare that this work presented as the Master's Thesis (TFM), corresponding to the academic year 2023-2024, is original, understood in the sense that no sources have been used in the preparation of the work without proper citation.

In Granada, on June 7, 2024

Sgd. David Villar Martos

Para ti, mami.
Mi ejemplo de esfuerzo y superación. Mi guía

Contents

Summary

Fairness in machine learning is an area that has gained great prominence and relevance in recent years. This area deals with the study and quantification of different measures of fairness on various specific problems and decision processes, as well as the development and implementation of solutions to create fairer systems. Unfortunately, at the limits of joint optimization between accuracy and fairness, a trade-off is reached, so that requiring a fairer system will necessarily imply less accuracy and vice versa. Due to this fact, multiobjective optimization emerges as a solution that allows finding a wide range of solutions exploring this optimization frontier. Genetic algorithms represent the state of the art in these optimization processes.

This project will be developed within this area. After conducting a review of the context from a mathematical perspective analyzing different types of mathematical interpretations of justice and multiobjective optimization, the creation of 3 novel algorithms based on decision trees will be proposed. One of them modifies the information gain criterion to incorporate fairness, another proposes a genetic optimization procedure on prunings in a matrix tree, and the last one modifies the loss function of a LightGBM model. In the two algorithms that do not inherently use genetic optimization processes, a genetic hyperparameter optimization procedure will be used, allowing us to find those models on the joint optimization frontier, in this case, of an accuracy and a fairness objective. A study will be conducted using 10 relevant datasets in the area, and they will be compared with a baseline algorithm. The results obtained show how these algorithms are able to find a wide range of Pareto-optimal solutions, and how some algorithms are better in certain scenarios compared to others. The implementation has been made public so that anyone can use them for any problem, leaving the specialist the option to select the model that best suits their requirements for accuracy, fairness, or any other implemented objective.

Resumen

La justicia en el aprendizaje automático es un área que ha tenido un gran auge y relevancia en los últimos años. Este área trata sobre el estudio y cuantificación de diferentes medidas de justicia sobre diferentes problemas y procesos de decisión concretos, así como del desarrollo e implementación de soluciones para poder crear sistemas más justos. Desgraciadamente, en los límites de la optimización conjunta entre precisión y justicia se llega a un balance, de manera que si se quiere un sistema más justo obligatoriamente implicará que haya menos precisión y viceversa. Debido a este hecho, la optimización multiobjetivo surge como una solución que permite encontrar una amplia gama de soluciones que exploren dicha frontera de optimización. Los algoritmos genéticos suponen el estado del arte en estos procesos de optimización.

Este proyecto se desarrollará en el marco de este área. Tras realizar una revisión del contexto desde una perspectiva matemática analizando distintos tipos de interpretaciones matemáticas de justicia y optimización multiobjetivo, se planteará la creación de 3 algoritmos novedosos, basados en árboles de decisión. Uno de ellos modifica el criterio de ganancia de información para incorporar justicia, otro de ellos plantea un procedimiento de optimización genética sobre podas en un árbol matriz, y el último modifica la función de pérdida de un modelo LightGBM. En los dos algoritmos que no utilizan de por sí procesos de optimización genetíca, se utilizará un procedimiento genético de optimización de hiperparámetros multiobjetivo, que nos permita encontrar esos modelos en la frontera de la optimización conjunta, en este caso, de un objetivo de precisión y otro de justicia. Se realizará un estudio utilizando 10 conjuntos de datos relevantes dentro del área, y se compararán con un algoritmo base. Los resultados obtenidos muestran cómo estos algotitmos son capaces de encontrar una amplia gama de soluciones Pareto-optimales, y cómo unos son mejores en determinadas situaciones frente a otros. La implementación se ha hecho pública, de cara que puedan ser empleados para cualquier problema, dejando al especialista la opción de seleccionar el modelo que más le convenga en cuanto a sus requisitos de precisión, justicia, o cualquier otro objetivo implementado.

Part I.

Aims of the project

In this part, the main initial goals of the work will be discussed.

1. Main goals of this project

Fairness in Machine Learning (ML) is gaining increasing relevance in the field of Artificial Intelligence, and it is critical to develop techniques that ensure a certain level of fairness in a mathematical sense in our constructed models to avoid emergent biases based on sensitive attributes, which may be potentially dangerous in this context. Among the techniques that exist within this field, those which modify the learning process of the base algorithm to incorporate fairness as a metric to optimize jointly with other classical precision metrics are a good approach to deal with these problems.

This project will consist of the development of three new techniques that incorporate fairness in our classifiers. Our base classifiers will be decision trees, and the modifications to those trees or the learning of them will be based on different principles (namely: modification of the information gain criterion when learning decision trees, genetic pruning of a large matrix tree, and modification of the loss function of a LightGBM model). They will be later applied to some datasets well-known in Fairness for Machine Learning literature.

The main goals of this work are as follows:

- **Review of Fairness in Machine Learning**: Review of the area of Fairness in Machine Learning. Contextual introduction to the field, brief review of methods for incorporating fairness, analysis of different definitions and mathematical measures of fairness.
- **Review of Multiobjective Optimization**: Review of the multiobjective optimization area. Definitions, analysis quality measures for solution sets.
- **Review of Machine Learning**: Analysis of basic concepts of machine learning and decision tree models as binary classifiers.
- **Development of Algorithms**: Constructing classification techniques that incorporate mathematical fairness objectives using Decision Trees as base classifiers. This will involve the development of three new algorithms, based on different principles:
  - **Fair Decision Tree**: The first algorithm to develop will modify the impurity criterion to include fairness while learning a decision tree. This will involve the incorporation of a new hyperparameter, which controls the balance between the base impurity criterion and the fairness criterion, ensuring consistent splits with a measured balance between precision and fairness. Fairness metrics at node splitting need to be defined in order to calculate them accurately.
  - **Fair Genetic Pruning**: The second algorithm to develop will create the largest optimal possible tree to classify a certain dataset, which will perfectly fit the training data, and then prune it to optimize the objectives given by the user for validation data. Prunings need to be coded in such a way that a multiobjective evolutionary algorithm can be applied.

  - **Fair LightGBM**: The third algorithm to develop will adapt the loss function of a boosting algorithm, this time using LightGBM as it employs decision trees as base classifiers, to incorporate fairness. This will involve adding a hyperparameter in a similar manner to Fair Decision Trees. Additionally, it is necessary to adapt the fairness metrics and extend them continuously for use in this model.

- **Multiobjective Optimization Framework**: Development of a multiobjective optimization framework to enable the use of these algorithms for this type of problem. Fair Genetic Pruning is inherently adapted for these problems, but for the other two, a hyperparameter optimization procedure needs to be implemented. All algorithms will be based on the NSGA-II algorithm.

- **Experimental setup**: Definition of an experimental setup, selection of well-known datasets in the context of Fairness in Machine Learning, and other specifications for testing the developed algorithms.

- **Analysis of results**: Analysis of results and graphics to understand and interpret the algorithm's performance, both individually and comparatively. Extraction of conclusions.

Part II.

Foundations: state of the art and theoretical framework.

In this part, the main foundations for the project will be presented. All the necessary concepts will be covered and studied, including the state of the art and related work, to provide a general context for the topic.

2. Background and motivations, state of the art

In this chapter, a general background is provided in the context of Fairness in Machine Learning. The motivations for the increasing importance of this topic in the world of Artificial Intelligence, as well as previous approaches to study and address the problem, will be analyzed.

2.1. Motivation

Machine Learning (ML) is widely used today in a wide range of fields such as medicine, industry, commerce... Despite its undeniable applicability in helping solve problems in multiple social and knowledge areas, it has negative qualities that have not yet been generally resolved and are well-documented.

One of them is that the decisions made by these systems can not only create inequalities among different population groups characterized by a common value in at least one attribute, in a sense that can be considered discriminatory, but can also enhance these differences over time [O'neil, 2016, Eubanks, 2018]. This fact has been acknowledged in numerous fields: human rights, privacy, employment, health... [pro, 2009, Angwin et al., 2016, Bolukbasi et al., 2016, Zehlike et al., 2017]. As a result, in recent years, there has been a growing awareness in a large part of the community to address this issue, motivating the study of the causes of these biased and discriminatory situations [Selbst et al., 2019, Binns et al., 2018], and the incorporation of techniques to ensure justice in the decisions made by these methods [Zafar et al., 2017a, Zehlike and Castillo, 2020]. The focus of the study will be oriented towards the construction of one of these methods. It has been demonstrated that there is an inherent trade-off between the quality of predictions and criteria for achieving justice [Menon and Williamson, 2018], but, nevertheless, methods that consider definitions of justice may be able to find solutions with comparable or better quality than current ones, also ensuring a less discriminatory decision [Valdivia et al., 2020].

To mathematically define what is considered to be fair [Verma and Rubin, 2018] is a non-trivial and necessary topic to address, in order to be able to develop methodologies which use it. Currently, it is known that there are multiple valid definitions, some of which belong to mutually exclusive families. This directly limits our options, and we may need to change our definition based on what we precisely aim to achieve.

For the implementation of these justice measures in our models, there are three groups of techniques [Friedler et al., 2019] depending on when do they intervene: preprocessing, inprocessing, and postprocessing. Preprocessing techniques typically involve detecting the existence of bias or partiality in the data before applying them to the machine learning method, and manipulating the data to meet our fairness standards. Inprocessing techniques involve altering the machine learning procedure itself to ensure fair decision-making. The usually alter the learning method, which often occurs by imposing constraints to optimize

within the method, leading to fair solutions. Postprocessing techniques modify the results of an already trained classifier to meet specified definitions of justice.

The project will focus on developing 3 innovative ideas within algorithm modification methods. These algorithms will be included in a multiobjective optimization procedure, aiming to explore the limits of optimizing conflicting objectives, specifically precision and fairness objectives. All these algorithms will use decision trees as the base machine learning model and they will employ genetic algorithms for the optimization process. The way these genetic algorithms are utilized will vary between algorithms; notably, two out of the three algorithms will utilize a genetic hyperparameter optimization procedure, while another will incorporate a genetic optimization process itself. The use of these algorithms is an excellent choice for conducting a study on multiobjective optimization context, even more when considering an increase in the dimensionality of the objective space (Many Objectives Optimization). The results obtained will be analyzed using different datasets widely employed in the context of fairness in machine learning. Moveover, the different definitions of fairness which will be used, as well as measures of quality and performance of the methods, will be mathematically studied.

The perspective of this project establishes enormous potential applicability in all areas where to apply machine learning solutions. This is because discriminatory practices have been documented in almost all of these contexts. Furthermore, since optimization yields multiple solutions with different balances in the ranges of fairness, it offers the specialist the possibility to choose the classifier with the most suitable characteristics based on what they aim to achieve.

2.2. Background

Concerning the establishment of fairness criteria in machine learning algorithms, there has been a clear trend to introduce it, directly or indirectly, through constraints which imply fairness. In this way, joint optimizations have been performed based on both quality and fairness criteria[Zafar et al., 2017b].

With respect to classifiers, particularly decision trees, efforts have been made to study ways of altering their construction to facilitate the incorporation of fairness measures. In [Kamiran et al., 2010], the information gain function is slightly modified to perform operations on the tree that aim to minimize entropy concerning the attribute that may cause discrimination. Various alternatives were explored, such as entropy concerning the class label.

In [Agarwal et al., 2018], it was proposed to reinterpret the problem of fair classification as a series of cost-sensitive classification problems, with the aim of achieving Pareto-optimal solutions with respect to both prediction quality and the considered definition of fairness.

In [Balashankar et al., 2019], a Pareto-optimal point is identified, which improves accuracy and also optimizes various precision measures on subgroups. Additionally, the concept of equal opportunity is satisfied.

In [Zafar et al., 2019], a convex problem with fairness constraints is considered, where the method's accuracy is optimized based on these constraints. During the problem formulation fairness is introduced in terms of a measure defined with respect to justice at the decision boundary. This measure inherently encapsulates other fairness measures. Disparate treatment leads to a tradeoff between fairness and accuracy in the model, and this tradeoff is determined by a parameter specified by the user based on their preference. The problem formulation also allows adding certain attributes as constraints to the problem.

In [Hu and Chen, 2020], the classic minimization problem with constraints is transformed into a social welfare maximization problem. Support vector machine regularization techniques and other methods were employed, leading to the conclusion that, when applying strict fairness criteria, it may be possible to achieve worse overall welfare for the study groups.

The work proposed in [Valdivia et al., 2020] does address, for the first time, the use of multiobjective optimization using a modification of NSGA-II, and it serves as a base building block for this project. Unlike the majority of literature in this field, this work employs the fairness criterion not as a constraint but as an objective to be minimized. Two objectives are used, one for error through geometric mean and another for justice in terms of the difference in the false positive rate, a criterion falling within the branch of predictive parity fairness. Complexity is considered only as an auxiliary criterion to establish dominance among individuals. The conclusion of the work is that using this method can lead to better solutions that surpass not only in fairness but also in accuracy, some machine learning methods used in the problems of the datasets employed.

Therefore, during this time, the community has shown an interest in conducting simultaneous optimizations in the context of decision quality and fairness in machine learning methods. The project's work takes a novel approach by attempting to explore the entire Pareto front in a multiobjective optimization, which can discern a wide range of solutions, all with optimal characteristics in the Pareto sense. For the first time, a method will be designed capable of efficiently optimizing various criteria of justice, precision, and complexity simultaneously, allowing not only to obtain a large number of alternative solutions with different justice profiles but also to understand the reachable non-dominance boundaries in the dataset, thus characterizing the problem in terms of justice.

Multiobjective problems are very common in the current world, because we often encounter contradictory objectives that we want to optimize simultaneously. A typical example would be an optimization problem in which we want to maximize the quality in the production of a product and minimize the investment we make in its production. Usually, we work with two objectives. When the number of conflicting objectives increases, it is already referred to as many objectives optimization, which, due to its characteristics, requires special techniques for its resolution.

Currently, there is a wide range of multiobjective optimization paradigms [Li et al., 2015], based on various criteria such as relaxing the concept of Pareto dominance, restricting population diversity at specific points, basing the search process on different quality measures, niche-based algorithms, and more. Each algorithm has its own distinct properties regarding structure, diversity, and convergence of solutions, which can be studied in a subsequent analysis. Due to the diverse perspectives, implementing multiple algorithms, combining their

solutions, and studying them comparatively will allow us to achieve better reliability when extracting results.

With respect to fairness metrics, they are currently handled in three major groups: group fairness and individual fairness, and counterfactual fairness. It has been demonstrated that some definitions of justice belonging to certain subfamilies are incompatible [Chouldechova, 2017], making the use of multiobjective optimization techniques a justified approach to work in this context.

2.3. Fairness in machine learning

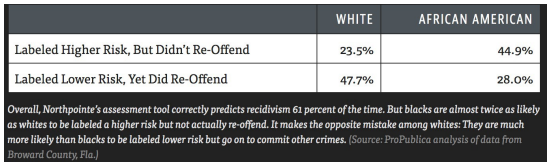
As time goes by, justice in machine learning is becoming an increasingly studied topic in the field of artificial intelligence. The issues related to fairness that machine learning could pose were known long before they gained the current significance. However, it wasn't until high-profile cases emerged that the community truly gave it the attention it deserves.

One of the cases that gained significant media impact was that of COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) in the USA. It was used to assess the risk of recidivism for individuals who had committed a crime, and served as a decision support system for determining variable sentencing periods for them.

In 2016, ProPublica published a data-driven analysis that had a significant impact. Its most striking finding was that COMPAS did not satisfy the false positive rate for both demographic subgroups, Caucasians and African Americans, being quite disparate. It turned out that among the inmates who did not reoffend, African American defendants were twice as likely to be classified as high risk. The study demonstrates how COMPAS has a much higher rate of false positives in the African American population compared to Caucasians (False Positives in Caucasians: 23.5%, in African Americans: 44.9%, False Negatives in Caucasians: 47.7%, and in African Americans: 28%). Based on this and similar findings, they labeled the tool as biased against the African American population.

However, Northpointe, now named Equivant, the developer of COMPAS, criticized ProPublica's work and pointed out that COMPAS met the criterion that positive predictive value remained consistent across both demographic groups. This concept means the proportion of those classified as high risk who were subsequently arrested for recidivism. COMPAS also met other fairness measures, such as calibration. Based on this, along with similar findings, it drew significant attention from the community, prompting a thorough examination of the origins, causes, and measures to eliminate or mitigate potential biases and discrimination in machine learning processes [Julia et al., 2016].

The origins and causes of this, as well as other biases in the context of machine learning, are none other than human beings. Humans have prejudices and biases, which lead us to transmit these values to machine learning models. This is quite common to occur in the creation or selection of data for the training of our models, which can happen either directly or indirectly. Indirect bias may occur when sampling from the population or when selecting features to measure from individuals. On the other hand, direct sources of bias occur, for



	WHITE	AFRICAN AMERICAN
Labeled Higher Risk, But Didn't Re-Offend	23.5%	44.9%
Labeled Lower Risk, Yet Did Re-Offend	47.7%	28.0%

Overall, Northpointe's assessment tool correctly predicts recidivism 61 percent of the time. But blacks are almost twice as likely as whites to be labeled a higher risk but not actually re-offend. It makes the opposite mistake among whites: They are much more likely than blacks to be labeled lower risk but go on to commit other crimes. [Source: ProPublica analysis of data from Broward County, Fla.]

Figure 2.1.: Image extracted from ProPublica website, where they showed their results [Julia et al., 2016].

example, if data is labeled based on subjective criteria. The involvement of humans in the dataset creation process, even if they are experts, makes it highly improbable that these have been taken objectively and unbiasedly. Some of the main causes of bias are listed below:

- **Biased sample**: It occurs when samples are taken with partiality. This fact is highly problematic because bias can become even more pronounced over time. An example of this can be observed in police patrols, which tend to patrol more at conflictive zones, leading to obtain more conflict reports over them.
- **Tainted examples**: Since Machine Learning systems use a sample to learn models, there is a problem if chosen examples are mislabeled or biased in any way. For instance, if it were selected by someone with a strong political or racial bias, the system probably will learn that bias and incorporate it into its predictions.
- **Limited attributes**: It may happen that the attributes chosen in the study affect minority groups differently and are not as representative or have a different type of correlation.
- **Disparity in sample size**: Imbalanced issues tend to consistently disadvantage minority groups, and to address them, we can incorporate data imbalance treatment techniques.
- **Proxies**: It may happen that we have certain attributes which serve as proxies or intermediaries for others that could be more sensitive attributes, such as using address instead of nationality.

These causes can be summarized into 3 groups: uncovering initially unnoticed differences in the starting domain (biased sample and tainted examples), problems with the sample taken while acknowledging differences in behavior (limited attributes and disparity in sample size), and understanding the causes of the disparities found (Proxies) However, once bias has occurred, it becomes a really challenging task to discern its origin.

With the aim of trying to avoid the adverse effects of bias, examples of definition include the rules, norms, and laws against discrimination in some countries. We can see that their definitions essentially fall into two major groups:

- **Disparate treatment**: If the decision-making process is entirely or highly based on the value of a sensitive attribute that is susceptible to being the source of discriminatory behaviors.
- **Disparate impact**: If the consequences based on the decisions made directly harm individuals with a specific value in a sensitive attribute that is susceptible to being the source of discrimination.

3. Mathematical foundations

In this chapter, the main contextual mathematical definitions and results will be presented and analyzed. This will help us gain a better and deeper understanding of the topic, as well as establish a notation to follow for further results.

3.0.1. Previous notations and notions

We will establish the convention to use class 1 as the positive class and class 0 as the negative class. Doing so, and using a predictor p to predict the class Y, we can differentiate the following sets:

- **Positive class**: Individuals for whom $Y = 1$.
- **Negative class**: Individuals for whom $Y = 0$.
- **True positives**: Individuals for whom $Y = 1$ and $p = 1$.
- **True negatives**: Individuals for whom $Y = 0$ and $p = 0$.
- **False positives**: Individuals for whom $Y = 0$ and $p = 1$.
- **False negatives**: Individuals for whom $Y = 1$ and $p = 0$.

3.1. Mathematical definitions of justice

It is then clear that there exists the need to incorporate methods that ensure fair decisions will be made in the context of each specific problem we try to solve. However, there is no defined consensus on what is considered fair within a given context, and multiple conceptions of justice have emerged from different, equally valid perspectives. The crucial question in our field is how can we define the concept of justice in a way which can be integrated in our machine learning systems.

There are numerous definitions of fairness in the literature. These can be classified into three major groups, which are not mutually exclusive, but have different approaches in terms of their conception. These three major groups are group fairness, individual fairness, and counterfactual fairness. In order to define each of these families more rigorously, we will introduce a notation.

Let us consider a probability space $(\Omega, \mathcal{A}, P)$, associated with an observational of a group of individuals with respect to a certain context. Over this probability space we will define a set of random variables: $X = X_1, \cdots, X_n, A, Y$. Each of these $X_i \colon (\Omega, \mathcal{A}, P) \to (\mathbb{R}, \mathcal{B}, P_{X_i}) \forall (i \in 1, \ldots, n)$ are random variables which represent specific observed characteristics of our individuals.

Random variable $A\colon (\Omega, \mathcal{A}, P) \to (\mathbb{R}, \mathcal{B}, P_A)$ will represent the value an individual takes on a sensitive attribute, i.e., one that may lead to discrimination. In the case of our study, A will be a 1-dimensional binary random variable, but there could be more than one sensitive attribute among those studied. Social groups will considered as collections of individuals who share the same value in the random variable A, thus describing two exhaustive and mutually exclusive social groups. It is important to note that the sensitivity of this attribute is context-dependent: there may be attributes that are considered sensitive in one context and not in others. For example, significant differences in the number of granted mortgage loans between individuals of different biological sexes may indicate discrimination, whereas when assessing an individual's suitability for a medical treatment, it may not be a sensitive attribute due to relevant physiological differences between individuals of different biological sexes.

Finally, $Y\colon (\Omega, \mathcal{A}, P) \to (\mathbb{R}, \mathcal{B}, P_Y)$ is a random variable representing a characteristic of individuals that we want to predict. For our study case, we will also consider that the random variable Y is binary, $Y \in 0, 1$, thus discrete. Therefore, our study will focus on supervised binary classification problems.

Our objective is to find a random variable $p\colon (\Omega, \mathcal{A}, P) \to (\mathbb{R}, \mathcal{B}, P_Y)$, which can be expressed as a function $p = p(X_1, \ldots, X_n, A)$, capable of approximating the variable Y as accurately as possible. This variable p will also be binary, taking the same values as Y. In our study case these will be 0 and 1.

Let us assume that all variables follow a joint distribution $(X, \dashv, Y) \sim D$. However, in most cases, we may not be able to sample data from that exact distribution; instead, the sampled examples will be biased. This is why, in a real-world scenario, we will sample from a biased distribution D', different from the theoretical true distribution D.

If we didn't consider anything about fairness and wanted to obtain a perfect classifier, we would achieve it if we could find a predictor p such that $p(X, A) = Y, \forall (X, A, Y) \sim D$. However, as we have seen, there is a high likelihood of some form of bias. Therefore, we need a method that allows us to ensure that we are minimizing the potential harmful effect that bias could have on the learning process.

We will now define useful concepts that will help us continue developing the theoretical framework for justice definitions:

Definition 3.1.1 (Independence of components of a random vector)**.** Let X be a random vector: $X = (X_1, \ldots, X_n), X_i\colon (\Omega, \mathcal{A}, P) \to (\mathbb{R}, \mathcal{B}, P_{X_i}), \forall i \in \{1, \ldots, n\}$. The components $X_1, \ldots, X_n$ of X are independent if they meet:

$$F_X(x) = F_{X_1}(x_1) F_{X_2}(x_2) \ldots F_{X_n}(x_n) \tag{3.1}$$

where F_X is the distribution function of the random vector X, and F_{X_i} is the distribution function of each 1-dimensional random variable $X_i, \forall i \in \{1, \ldots, n\}$.

Lemma 3.1.0.1 (First characterization of independence of components of a random vector)**.** *Let X be a discrete random vector, where $X = (X_1, \ldots, X_n), X_i\colon (\Omega, \mathcal{A}, P) \to (E_i, \mathcal{B}, P_{X_i}), \forall i \in$*

$\{1,\dots,n\}$. *The components* X_1, X_2 *are independent if and only if* $\forall(x_1,\dots,x_n) \in E_1 \times \cdots \times E_n$,

$$P_X(x_1,\dots,x_m) = p_{X_1}(x_1)\dots p_{X_n}(x_n) = P[X_1 = x_1]\dots P[X_n = x_n] \tag{3.2}$$

where p_{X_i} *is the probability mass function of the random variable* $X_i, \forall i \in \{1,\dots,n\}$.

Lemma 3.1.0.2 (Characterization of independence of the components of a 2-dimensional discrete random vector)**.** *Let* X *be a random 2-dimensional discrete random vector:* $X = (X_1, X_2)$, $X_i\colon (\Omega, \mathcal{A}, P) \to (E_i, \mathcal{B}, P_{X_i}), \forall i \in \{1,2\}$. *Random variables* X_1, X_2 *are independent* $\Leftrightarrow P[X_1 = x_1 | X_2 = x_2] = P[X_1 = x_1], \forall (x_1, x_2) \in E_1 \times E_2$

Proof. $\boxed{\Rightarrow}$ As X_1 and X_2 are independent, $P[X_1 = x_1, X_2 = x_2] = P[X_1 = x_1]P[X_2 = x_2]$. It is known that $X_1|X_2$ is a 2-dimensional random variable, with probability mass function $p_{X_1|X_2}(x_1,x_2) = \frac{P[X_1=x_1, X_2=x_2]}{P[X_2=x_2]} = \frac{P[X_1=x_1]P[X_2=x_2]}{P[X_2=x_2]} = P[X_1 = x_1]$.
$\boxed{\Leftarrow}$ It is known that $P[X_1 = x_1|X_2 = x_2] = p_{X_1|X_2}(x_1,x_2) = \frac{P[X_1=x_1, X_2=x_2]}{P[X_2=x_2]} = P[X_1 = x_1] \Rightarrow P[X_1 = x_1, X_2 = x_2] = P[X_1 = x_1]P[X_2 = x_2]$, therefore X_1 and X_2 are independent. □

In order to abbreviate, we will use the notation $P[X_1|X_2] = P[X_1]$ to refer to the fact that $P[X_1 = x_1|X_2 = x_2] = P[X_1 = x_1], \forall (x_1, x_2) \in E_1 \times E_2$.

Lemma 3.1.0.3. *Let* X *be a 2-dimensional discrete random vector,* $X = (X_1, X_2), X_i\colon (\Omega, \mathcal{A}, P) \to (E_i, \mathcal{B}, P_{X_i}), \forall i \in \{1,2\}$, *where additionally,* $E_2 = \{e_\alpha, e_\beta\}$. *Random variables* X_1, X_2 *are independent* $\Leftrightarrow P[X_1 = x_1|X_2 = e_\alpha] = P[X_1 = x_1|X_2 = e_\beta], \forall x_1 \in E_1$

Proof. $\boxed{\Rightarrow}$ Both X_1 and X_2 are independent, so using the previous lemma, $P[X_1 = 1|X_2 = e_\alpha] = P[X_1 = x_1|X_2 = e_\beta] = P[X_1 = x_1], \forall x_1 \in E_1$.
$\boxed{\Leftarrow}$ Now $P[X_1 = x_1|X_2 = e_\alpha] = P[X_1 = x_1|, \forall x_1 \in E_1$. The events $X_2 = e_\alpha$ and $X_2 = e_\beta$ constitute an Ω partition, and additionally $P[X_2 = e_\beta] = 1 - P[X_2 = e_\alpha$, so we can apply the law of total probability the following way:

$$\begin{aligned} P[X_1] &= P[X_1 = x_1|X_2 = e_\alpha]P[X_2 = e_\alpha] + P[X_1 = x_1|X_2 = e_\beta]P[X_2 = e_\beta] \\ &= P[X_1 = x_1|X_2 = e_\alpha](P[X_2 = e_\alpha] + P[X_2 = e_\beta]) \\ &= P[X_1 = x_1|X_2 = e_\alpha](P[X_2 = e_\alpha] + 1 - P[X_2 = e_\alpha]) \\ &= P[X_1 = x_1|X_2 = e_\alpha] = P[X_1 = x_1|X_2 = e_\beta], \forall x_1 \in E_1. \end{aligned} \tag{3.3}$$

And using lemma 3.1.0.2, X_1 and X_2 are independent. □

The following notation abbreviation will be used $P[X_1|X_2 = e_\alpha] = P[X_1|X_2 = e_\beta]$ to denote the fact that $P[X_1 = x_1|X_2 = e_\alpha] = P[X_1 = x_1|X_2 = e_\beta], \forall x_1 \in E_1$

3.1.1. Group fairness

Group fairness is based on the premise that for justice to prevail, there should be no significant probabilistic differences in the classifications made among different demographic groups. We will address decision problems involving two distinct demographic groups, one that can be considered privileged in social or historical terms, and the other, unprivileged, aiming to minimize differences between them. The way we consider these probabilities will

lead to the emergence of multiple measures, which will be discussed below.

One of the major advantages of fairness definitions within this field is that they are independent of the problem to be applied, as specific measures or considerations for the problem at hand do not need to be established in their definition. This is because they are entirely probabilistic, unlike the other two fairness models. A primary disadvantage could be that these measures generally face detection issues when multiple protected groups are considered simultaneously, and new protected groups are generated as intersections of several of these. [Binns, 2020]. Another main drawback is that some measures falling under this interpretation of justice cannot be simultaneously obtained, as we will see later on.

3.1.2. Unawareness

Definition 3.1.2 (Unawareness)**.** A predictor p satisfies the condition of unawareness if $p(X, A) = p(X)$, or in other words, the value of the sensitive attribute A is not used during the prediction process.

What we can observe here is that in theory, we completely eliminate disparate treatment, as we treat individuals indifferently regardless of the value they take on the sensitive attribute. However, this is not entirely the case, as some random variable considered in X may be highly correlated to the attribute A, and therefore, we may still take the sensitive attribute into account in our predictions, not solving the problem. These attributes are called proxy attributes. Even in the absence of these attributes correlated with A, removing this attribute results in a loss of information that could be crucial both in the learning process and in the final classification.

For these reasons, although it may be a very simple and initially effective approximation, it will not be genuinely useful in most cases. The preference is given to the use of other measures that more effectively ensure this fact, as will be seen in counterfactual fairness.

3.1.3. Demographic parity

Definition 3.1.3 (Demographic parity)**.** A predictor p satisfies the condition of demographic parity if p and A are independent. In our case, where the attribute A can only take the values 0 and 1, the condition can be expressed as $P[p = \alpha | A = 0] = P[p = \alpha | A = 1, \forall a \in \{0, 1\}$, which will be expressed as $P[p|A = 0] = P[p|A = 1] = P[p]$.

In the scenario where we are dealing with a classification problem and the sensitive attribute is not binary, it would be preferable to solely employ the definition of independence rather than opting for the exhaustive definition. However, in our case, it will be convenient and later necessary to demonstrate the fairness impossibility theorems. Since our predictor is binary, it is also clear, using this definition, that $P[p = \alpha | A = 0] = P[p = \alpha | A = 1] \Leftrightarrow P[p = 1 - \alpha | A = 0] = P[p = 1 - \alpha | A = 1]$, which justifies the simplification in the notation.

This measure aims to assign the same probability of classifying as positive/negative to an individual regardless of the value of their sensitive attribute. There is an extension to this definition known as conditional demographic parity, where we add to the conditioning the value of an attribute or a set of attributes considered relevant, known as legitimizing factors (L). Therefore, the adapted definition would be $P[p = \alpha | L = l, A = 0] = P[p = \alpha | L = l, A =$

$1], \forall a \in \{0, 1\}$ [Verma and Rubin, 2018].

This way of incorporating fairness is commonly used, but through the use of relaxed measures. This is because the condition is quite restrictive, and we could significantly lose prediction quality if we were to demand it. Some common ways in which this measure can be relaxed include the following:

- **Relative difference**: As $P[p|A = 0] = P[p|A = 1] \Leftrightarrow \frac{P[p|A=0]}{P[p|A=1]} = 1$, we can relax the measure incorporating a constant $\epsilon \in [0, 1]$ so that $\frac{P[p|A=0]}{P[p|A=1]} \geq 1 - \epsilon$.
- **Absolute difference**: As $P[p|A = 0] = P[p|A = 1] \Leftrightarrow P[p|A = 0] - P[p|A = 1] = 0$, we can relax the measure incorporating a constant $\epsilon \in [0, 1]$ so that $|P[p|A = 0] - P[p|A = 1]| \leq \epsilon$.

These relaxation rules have been part of significant measures, for example, in the Uniform Guidelines for Employee Selection Procedures, where the 80% rule for adverse impact was employed in the context of employee hiring. [Biddle, 2006] It has been studied and endorsed [Hu and Chen, 2018].

Among other advantages, we can note, for example, that it helps to increase the importance of the disadvantaged class over time. This is because, in the decision-making process, the probabilities of predicting an individual as positive or negative conditioned on the value of the sensitive attribute are forced to be equal or similar. These short or medium-term decisions can make a significant difference in the long run. Additionally, this measure is independent of the true value of the variable to be predicted Y, making it useful in contexts where this measure is difficult to quantify.

However, this latter fact can also be considered a disadvantage. For example, it ignores the possible correlation between A and Y. With high probability, the perfect predictor for this sample will not satisfy this condition since, in most cases $P[Y = 0|A = 0] \neq P[Y = 0|A = 1]$. Let us consider an example where we are in a job interview where there is an equal number of Caucasians and Asians, and by chance, there are more prepared Asians than Caucasians. Regardless of this fact, we would end up hiring an equal number of Caucasians and Asians, potentially assuming hiring an unprepared Caucasian, as we only care about the probability conditioned on the sensitive attribute.

3.1.4. Equalized odds

Definition 3.1.4 (Equalized odds)**.** A predictor p satisfies the condition of equalized odds if the probability p conditioned on the value of Y is independent of the value of A. In our context we can express this condition as: $P[p = r|Y = z, A = 0] = P[p = r|Y = z, A = 1] = P[p|Y], \forall r, z \in \{0, 1\}$, which will be expressed as $P[p|Y, A = 0] = P[p|Y, A = 1] = P[p|Y]$.

This measure is also known as separation. Similar to demographic parity, this definition can be relaxed, allowing for non-compliance up to a certain margin. The primary way to relax it is often to use the measure known as equality of odds, whose definition is as follows:

Definition 3.1.5 (Equalized odds)**.** A predictor p satisfies the condition of equalized odds if $P[p \neq Y|A = 0] = P[P \neq Y|A = 1]$.

The issue with this relaxed definition is that there can be an imbalance between false positives." ($P[p = 1|Y = 0]$) and false negatives ($P[p = 0|Y = 1]$).

Another way to relax it is to exclusively consider the true positive rate, which has traditionally been more relevant in some applications than true negatives. It is known as equality of opportunity and is defined as follows:

Definition 3.1.6 (Equality of opportunity)**.** A predictor p satisfies the condition of equality of opportunity if $P[p = 1|Y = 1A = 0] = P[p = 1 = 1, A = 1]$. It is also known as true positive parity.

Analogously, true negative parity, false positive parity, and alse negative parity can be defined. One way to relax this equalized odds condition is to use one or some of these parities.

As positive highlights, we can say that errors are reduced uniformly for each of the demographic subgroups. Additionally, contrary to what happened in the case of demographic parity, a perfect predictor always satisfies the condition of equalized odds, since $p = Y$.

One of the main problems of using this fairness measure is that over the long term, disparities may increase between the two demographic groups. For example, consider two groups of 100 people each, where the first group has 50 qualified individuals for a job position, while the second group has only 1. If we need to choose 26 people using equalized odds, we will select 25 indiviuals from the first group and 1 from the second. This results in individuals from the first group having more opportunities to secure the position than those from the second group, potentially creating a feedback loop. A better job leads to improved family conditions, enabling the children of the first group to access better education, and consequently, they become better qualified for the position compared to the children of the second group.

We will now provide a necessary and sufficient condition for the equalized odds condition to be satisfied [Tang and Zhang, 2020]:

Theorem 3.1.1 (Characterization of equalized odds)**.** *Let $\mathcal{X}, \mathcal{Y}$ y $\mathcal{A}$ the domains of the random variables X, Y and A respectively. Let us suppose that A and Y are independent, and inside $X = (X_1, \ldots, X_n)$ there are variables which are not independent of Y, expressing $S_A^{(y)} = \{a \in A : \exists x \in \mathcal{X} : p(a, x) = y\}$, and also $S_{X|a}^{(y)} = \{x \in \mathcal{X} : p(a, x) = y$, then the condition of equalized odds will be satisfied if and only if the next two conditions are met:*

I $S_A^{(y)} = \mathcal{A}, \forall t \in \mathcal{Y}$

II $\sum_{x \in S_{X|a}^{(y)}} P[X = x|Y = y, A = a] = \sum_{x \in S_{X|a'}^{(y)}} P[X = x|Y = y, A = a'], \forall y \ \in \mathcal{Y}, \forall a, a' \in \mathcal{A}$

Proof. For the equalized odds conditions to be satisfied, the following expression needs to happen:

$$P[p = \alpha|Y = y, A = a] = P[p = \alpha|Y = y], \forall a \in A, \forall \alpha, y \in \mathcal{Y} \tag{3.4}$$

We can manipulate the left-hand side of the equation to obtain the following equality:

$$P[p = \alpha|Y = y, A = a] = \sum_{x \in \mathcal{X}} P[p = \alpha|Y = y, A = a, X = x] P[X = x|Y = y, A = a] \tag{3.5}$$

We know that $p = p(X, A)$ is a deterministic function that depends on A and X, and using this we can write the following equality:

$$P[p(X,A) = \alpha | Y = y, A = a, X = x] = P[p(X,A) = \alpha | A = a, X = x] \in \{0,1\} \tag{3.6}$$

The value $P[p(X, A) = \alpha | A = a, X = x]$ will be 0 or 1, depending on wether $p(x, a) = \alpha$ or not. From this last equality, we can see that the conditional probability $P[X = x | A = a, Y = y]$ can contribute to the summation only when $p(a, x) = \alpha$. Therefore, we can rewrite the left-hand side of equation (3.4) as follows:

$$P[p = \alpha | Y = y, A = a] = \sum_{s \in S^{(\alpha)}_{X|a}} P[X = x | Y = y, A = a] \tag{3.7}$$

We will denote $Q^{(\alpha)}(a, y) = \sum^{(\alpha)}_{s \in S_{X|a}} P[X = x | Y = y, A = a]$. Now, let us manipulate the right-hand side of equation (3.4) to obtain:

$$\begin{aligned} P[p = \alpha | Y = y] &= \sum_{a \in \mathcal{A}} \sum_{x \in \mathcal{X}} P[p = \alpha | Y = y, A = a, X = x] P[X = x, A = a | Y = y] \\ &= \sum_{a \in S^{(\alpha)}_A} \sum_{x \in S^{(\alpha)}_{X|a}} P[X = x | Y = y, A = a] P[A = a | Y = y] \\ &= \sum_{a \in S^{(\alpha)}_A} Q^{(\alpha)}(a, y) P[A = a | Y = y] \end{aligned} \tag{3.8}$$

The condition of equalized odds occurs if and only if equation (3.4) is satisfied. Therefore, the value $Q^{(\alpha)}(a, y)$ cannot change with a. Thus, by making the substitutions that were established, equation (3.4) can be rewritten as:

$$Q^{(\alpha)}(a,y) = \sum_{a \in S^{(\alpha)}_A} Q^{(\alpha)}(a,y) P[A = a | Y = y] = Q^{(\alpha)}(a,y) \sum_{a \in S^{(\alpha)}_A} P[A = a | Y = y] \tag{3.9}$$

This equality gives us condition I), since if it doesn't hold, it would imply that $\sum_{a \in X^{(\alpha)}_A} P[A = a|Y)y] < 1$. Condition II) arises naturally because, as we mentioned earlier, the value of $Q^{(\alpha)}(a, y)$ does not change with a. By definition, we directly have the condition $Q^{(\alpha)}(a, y) = \sum_{s \in S^{(\alpha)}_{X|a}} P[X = x | Y = y, A = a] = \sum_{s \in S^{(\alpha)}_{X|a'}} P[X = x | Y = y, A = a']$

Finally, the entire procedure is reversible, thus establishing the double implication. □

This property is useful for implementing post-processing techniques with the aim of ensuring that the equalized odds condition holds for the applied predictor. [Tang and Zhang, 2020].

3.1.5. Predictive rate parity

Definition 3.1.7 (Predictive rate parity)**.** A predictor p satisfies the condition of predictive rate parity if the probability of the attribute Y taking a value conditioned on the value of p is independent of A. In our context we can express this condition as $P[Y = z | p = r, A = 0] = P[Y = z | p = r, A = 1], \forall z, r \in \{0, 1\}$ [Zafar et al., 2017a]. We can simplify it, writing $P[Y | p, A = 0] = P[Y | p, A = 1]$.

This condition is also known as Sufficiency. It is equivalent to verifying two conditions: $P[Y = 1|p = 1, A = 0] = P[Y = 1|p = 1, A = 1]$ and $P[Y = 0|p = 0, A = 0] = P[Y = 0|p = 0, A = 1]$. These are known in the literature as positive and negative predictive rate parity, respectively, or also positive and negative predictive values rate parity. One possible way to relax this measure would be to consider only one of these two measures instead of both.

In this case, it also happens that with a perfect predictor, where $Y = p$, predictive rate parity is trivially satisfied, just like with equalized odds. Therefore, it is a desirable feature if one considers using this fairness definition.

The problems associated with its use are similar to those that occurred with equalized odds. As will be seen later, in fact, it is impossible to find a predictor p that is able to satisfy any pair of group fairness measures among the three previously proposed: demographic parity, equalized odds, and predictive rate parity.

To define both this measure and the next one, it is necessary to consider that our predictor does not have a discrete output in the range of possible values that the attribute Y can take. We will consider that our predictor p assigns a score to each individual, denoted as $S(X, A) = s$. For simplicity and without loss of generality, we will assume that $s \in [0, 1]$. This value, for example, could be interpreted as the probability that a particular example belongs to the positive class. This value will later be subject to transformation into a label, with one of the most common approaches being the establishment of a threshold value above which instances will be classified as positive. Initially, we will focus only on the score assignment and not on the transformation into a label. Therefore, we can directly refer to the prediction process carried out by S.

3.1.6. Calibration

Definition 3.1.8 (Calibration)**.** A predictor p satisfies the condition of calibration if $P[Y = 1|S = s, A = 0] = P[Y = 1|S = s, A = 1]$, for every possible value of s.

The application of this measure is subject to being able to know a priori all possible values of S. In case the number of possible values is discrete and manageable, we could consider using it. However, in the case of a continuous range, its application as defined will not be appropriate, as the probability of S taking a specific value will be equal to 0. In the case of continuous S, to continue applying this definition, threshold values are used as follows:

$$P[Y = 1|S \geq s, A = 0] = P[Y = 1|S \geq s, A = 1] \text{ , for every value of } s. \tag{3.10}$$

In practice, there are several heuristics to achieve calibration. For example, Platt scaling is a popular method that takes a possibly uncalibrated score and treats it as a feature to train a logistic regression. The goal is to find coefficients a and b to fit the value of a sigmoidal function:

$$R = \frac{1}{1 + e^{(aS+b)}} \tag{3.11}$$

So that it fits the actual value of Y, attempting to minimize the loss function known as LogLoss or logarithmic loss function, which will be discussed later:

$$L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p) \tag{3.12}$$

This measure is widely used, for example, in risk assessment in criminology [Danner et al., 2016]. It represents, as is evident, a good extension of the concept of predictive rate parity in case the predictor returns a score instead of a label. It can also be considered as a generalization of predictive rate parity to non-binary classification contexts. However, being a direct extension of the previous case, it suffers from the same problems [Corbett-Davies and Goel, 2018].

We have already mentioned the relations between predictive rate parity and good calibration. Calibration can be seen as a generalization of predictive rate parity to the non-binary case. The score value s could, in principle, be continuous, but if it could only take the values 0 or 1, achieving calibration for a predictor p would be equivalent to ensuring that it satisfies the predictive rate parity condition.

Let us consider the case where, having a continuous score s, we classify by setting a threshold value s_{th}, so that individuals with $s(x,a) > s_{th} \Rightarrow p(x,a) = 1$ and individuals with $s(x,a) \leq s_{th} \Rightarrow p(x,a) = 0$. By following this procedure, we will not always achieve that our predictor satisfies the predictive rate parity condition, as we will see [Garg et al., 2020]. Therefore the probability $P[Y = 1|p = 1, A]$ can be expressed as $P[Y = 1|S > s_{th}, A]$. Using this, we can write:

$$\begin{aligned} P[Y, S > s_{th}|A] &= \int_{s_{th}}^{1} \underbrace{P[Y|S,A]}_{\text{Calibration}} P[S|A]ds \Rightarrow \\ \Rightarrow \underbrace{P[Y|S > s_{th}, A]}_{\text{Predictive rate parity}} &= \frac{\int_{s_{th}}^{1} P[Y|S,A]P[S|A]ds}{\int_{s_{th}}^{1} P[S|A]ds} \end{aligned} \tag{3.13}$$

This equation relates predictive rate parity to calibration, showing that even when the calibration term $P[Y|S, A]$ is the same for both groups, the probability distribution of the score, expressed in the lower equation by $P[S|A]$, can vary among different groups, causing parity in predictive ratios not to be satisfied. To make it more intuitive, consider a special case where there are only two values s_1, s_2 above the threshold value s_{th}, so that $P[S|A] \neq 0$. In other words, all individuals receiving risk scores above the threshold have the possibility of receiving only two scores: s_1 or s_2. Therefore, $P[S > s_{th}|A] = P[S = s_1|A] + P[S = s_2|A]$. In this special case, the last equation is reduced to:

$$P[Y = 1|P = 1, A] = \frac{P[Y = 1|S = s_1, A]P[S = s_1|A] + P[Y = 1|S = s_2, A]P[S = s_2|A]}{P[S = S_1|G] + P[S = s_2|G]} \tag{3.14}$$

Let us give an example in this sense where calibration is satisfied, but predictive rate parity is not. Suppose two demographic groups a and b. Suppose S can only return 3 possible values, being $0.25, 0.5, 0.75$, and we set a threshold value $s_{th} = 0.49$ Suppose we have 100 individuals in each group, and the classification conditions given in table 3.1 For each group, you can see the number of instances for which a certain score has been predicted, as well as the number of instances that were actually positive among those predicted with that score:

It can be seen that the calibration condition is satisfied because, for each score, the proportion of positive individuals among the predicted ones is equal for both groups. However, the predictive rate parity condition is not satisfied because in this case $P[Y = 1|p = 1|A = a] = \frac{2}{3}$, while $P[Y = 1|p = 1|A = b] = \frac{7}{12}$. Another measure with a more comprehensive definition

Table 3.1.: Table where a prediction procedure is shown, where Calibration is fulfilled and Parity in the predictive ratio is not fulfilled.

Score	**Group** a		**Group** b		**Prediction after setting**
	Predicted	Positives	Predicted	Positives	**the threshold** $s\_th = 0.49$
0.25	40	16	40	16	Negative
0.50	20	10	40	20	Negative
0.75	40	30	20	15	Positive
Total	100	56	100	51	

based on calibration, but also commonly used, is perfect calibration:

Definition 3.1.9 (Perfect calibration)**.** A predictor p satisfies the condition of perfect calibration if $P[Y = 1|S = s, A = 0] = P[Y = 1|S = s, A = 1] = s$, for every possible value of s.

This condition is also known as group calibration. If a predictor p satisfies the measure, it will not only be meeting calibration but also the score values assigned by the predictor truly correspond to the probabilities of belonging to the positive class. Let us see an interesting property of predictors that satisfy well calibration:

Lemma 3.1.1.1. *If a predictor p meets predictive rate parity, then it exists a function l: $[0,1] \to [0,1] : l(p)$ which satisfies the condition of perfect calibration.*

Proof. Let us consider a specific demographic group a and let $l(\alpha) = P[Y = 1|p = \alpha, A = a]$. Since p satisfies the predictive rate parity condition, this probability will be the same regardless of the value of a. Now, considering two different demographic groups a and b, we have:

$$\begin{aligned} \alpha &= P[Y = 1|l(p) = \alpha, A = a] \\ &= P[Y = 1|p \in l^{-1}(\alpha), A = a] \\ &= P[Y = 1|p \in l^{-1}(\alpha), A = b] \\ &= P[Y = 1|l(p) = \alpha, A = b] \end{aligned} \tag{3.15}$$

Where, for the third equality, we have once again used the predictive rate parity condition, thereby satisfying the perfect calibration condition. □

3.1.7. Balance for the positive/negative class

Definition 3.1.10 (Balance for the positive/negative class)**.** A predictor p satisfies the condition of balance for the positive class if $\mathbb{E}[s|Y = 1, A = 0] = \mathbb{E}[s|Y = 1, A = 0]$. Similarly, it can be defined for the negative class as $\mathbb{E}[s|Y = 1, A = 0] = \mathbb{E}[s|Y = 1, A = 0]$.

This condition can be seen as a generalization of the equalized odds metric to non-binary cases. Indeed, if s can only take the values 0 and 1, the value of s would directly become the prediction, and we would have exactly the equalized odds condition. Therefore, it is a natural extension to this family of metrics.

In [Kleinberg et al., 2016], it is demonstrated that the only cases in which the conditions of balance for the positive class, balance for the negative class, and calibration can be met are

when we have a perfect predictor p, or there is the same number of positive instances in each demographic group.

3.2. Fairness impossibility theorems

The three definitions of group fairness falling within the class of group fairness —demographic parity, equalized odds, and predictive rate parity— have simple definitions and can be comfortably expressed in probabilistic terms. However, they have a drawback, which is that no pair of them can be satisfied simultaneously for a given problem and classifier, except in extreme cases. Therefore, we will establish hypotheses, which also arise naturally, to avoid falling into such cases.

The first hypothesis is to assume dependence between A and Y. It is precisely understood that there is some kind of dependency between this characteristic and the label to be predicted. If there were no such dependency, the variable A would not be directly considered in an individual's feature set.

The second hypothesis is to assume dependence between p and Y. If this were not the case, no learning process would be taking place since our prediction would not depend on the variable to be predicted.

Once these hypotheses, which we will refer to as non-degeneracy hypotheses, are established, we will proceed to characterize the independence with which we can prove the theorems.

The basis for the proof of these theorems can be found in [del Barrio et al., 2020]. As we will verify, assuming the dependence of A and Y, as well as the dependence of p and Y, if the predictor p satisfies either demographic parity, equalized odds, or predictive rate parity, any of the other two conditions can be satisfied.

Theorem 3.2.1 (Impossibility of demographic parity and equalized odds)**.** *Under the non-degeneracy hypothesis, demographic parity and equalized odds conditions cannot hold simultaneously for any classifier.*

Proof. Let us prove it by contradiction. To start, let us consider a relevant equality, which will consist on another way to express $P[p = 1|A = a]$:

$$\begin{aligned} P[p=1|A=a] &= P[p=1,Y=1|A=a] + P[p=1,Y=0|A=a] \\ &= \frac{P[p=1,Y=1,A=a]}{P[A=a]} + \frac{P[p=1,Y=0,A=a]}{P[A=a]} \\ = \frac{P[p=1,Y=1,A=a]}{P[Y=1,A=a]} + \frac{P[Y=1,A=a]}{P[A=a]} &\frac{P[p=1,Y=0,A=a]}{P[Y=0,A=a]} \frac{P[Y=0,A=a]}{P[A=a]} \\ = P[p=1|Y=1,A=a]P[Y=1|A=a] &+ P[p=1|Y=0,A=a]P[Y=0|A=a] \end{aligned} \tag{3.16}$$

and this happen $\forall a \in \{0, 1\}$.

Let us suppose now that the condition of equalized odds holds, and therefore $P[p = 1|Y = 1, A = 0] = P[p = 1|Y = 1, A = 1] = P[p = 1|Y = 1]$, as well as $P[p = 1|Y = 0, A = 0] =$

$p[p = 1|Y = 0, A = 1] = P[p = 1|Y = 0]$.

Let us consider that the condition of demographic parity must also hold, so we have the following expression:

$$0 = P[p = 1|A = 1] - P[p = 1|A = 0] \tag{3.17}$$

Using the last equality from equation (3.16) this can be expressed as:

$$\begin{aligned} &P[p = 1|Y = 1, A = 1]P[Y = 1|A =]1 + P[p = 1|Y = 0, A = 1]P[Y = 0|A = 1] \\ &-(P[p = 1|Y = 1, A = a]P[Y = 1|A = 0] + P[p = 1|Y = 0, A = a]P[Y = 0|A = 0]) \end{aligned} \tag{3.18}$$

As equalized odds holds, the previous expression is equivalent to:

$$\begin{aligned} &P[p = 1|Y = 0](P[Y = 0|A = 1] - P[Y = 0|A = 0])+ \\ &P[p = 1|Y = 1](P[Y = 1|A = 1] - P[Y = 1|A = 0]) \end{aligned} \tag{3.19}$$

Finally, we observe that the probabilities in both subtractions are opposite. Therefore, we can rewrite everything as:

$$(P[p = 1|Y = 0] - P[p = 1|Y = 1])(P[Y = 0|A = 1] - P[Y = 0|A = 0]) \tag{3.20}$$

Therefore, we have obtained that $0 = (P[p = 1|Y = 0] - P[p = 1|Y = 1])(P[Y = 0|A = 1] - P[Y = 0|A = 0])$, and therefore one of the two factors must be 0. However, if one were 0, it would violate the non-degeneracy hypothesis. □

Theorem 3.2.2 (Impossibility of demographic parity and predictive rate parity)**.** *Under the non-degeneracy hypothesis, demographic parity and predictive rate parity conditions cannot hold simultaneously for any classifier.*

Proof. Let us consider that both parities hold true simultaneously. If this is the case, on the one hand, $P[p|A] = P[p]$, and on the other hand $P[Y|p, A] = P[Y|p]$, leading to a contradiction. Due to the non-degeneracy hypothesis, we know that both Y and A are dependent, thus $P[Y|A] \neq P[Y]$. Nonetheless, using a similar process done at the previous proof using the last equality from equation (3.16):

$$\begin{aligned} P[Y|A] &= P[Y, p = 0|A] + P[Y, p = 1|A] \\ &= \frac{P[Y, p = 0, A]}{P[A]} + \frac{P[Y, p = 1, A]}{P[A]} \\ &= \frac{P[Y, p = 0, A]}{P[p = 0, A]} \frac{P[p = 0, A]}{P[A]} + \frac{P[Y, p = 1, A]}{P[p = 1, A]} \frac{P[p = 1, A]}{P[A]} \\ &= P[Y|p = 0, A]P[p = 0|A] + P[Y|p = 1, A]P[p = 1|A] \end{aligned} \tag{3.21}$$

As both parities occur, we obtain:

$$\begin{aligned} &P[Y|p = 0, A]P[p = 0|A] + P[Y|p = 1, A]P[p = 1|A] = \\ &P[Y|p = 0]P[p = 0] + P[Y|p = 1]P[p = 1] \end{aligned} \tag{3.22}$$

Using the law of total probability, this is the same as $P[Y]$, and consequently, $P[Y|A] = P[Y]$, so by definition Y is independent of A, which is a contradiction.

□

Theorem 3.2.3 (Impossibility of equalized odds and predictive rate parity)**.** *Under the non-degeneracy hypothesis, demographic parity and predictive rate parity conditions cannot hold simultaneously for any classifier except for a perfect classifier.*

Proof. Let us consider that both conditions hold true simultaneously. In particular, we will be focusing on the following equalities:

$$\begin{aligned} P[p=1|Y=1,A=0] &= P[p=1|Y=1,A=1] \\ P[p=1|Y=0,A=0] &= P[p=1|Y=0,A=1] \\ P[Y=1|p=1,A=0] &= P[Y=1|p=1,A=1] \end{aligned} \tag{3.23}$$

We will now use the first equation used in the proof of theorem 3.2.1 to write some expressions, which are true $\forall a \in \{0,1\}$:

$$\begin{aligned} P[p=1|A=a] = P[p=1|Y=1,A=a]P[Y=1|A=a] + \\ P[p=1|Y=0,A=a]P[Y=0|A=a] \end{aligned} \tag{3.24}$$

Once stated this expression, let us consider the following one:

$$\begin{aligned} P[p=1|Y=1,A=a]P[Y=1|A=a] = \frac{P[p=1,Y=1,A=a]}{P[Y=1,A=a]}\frac{P[Y=1,A=0]}{P[A=a]} = \\ \frac{P[p=1,Y=1,A=a]}{P[p=1,A=a]}\frac{P[p=1,A=0]}{P[A=a]} = P[Y=1|p=1,A=a]P[p=1|A=a] \end{aligned} \tag{3.25}$$

We can substitute in the last expression $P[p=1|A=a]$ by what we have just obtained, getting the following expression:

$$\begin{aligned} P[p=1|Y=1,A=a]P[Y=1|A=a] = \\ P[Y=1|p=1,A=a](P[p=1|Y=1,A=a]P[Y=1|A=a] + P[p=1|Y=0,A=a]P[Y=0|A=a]) = \\ P[Y=1|p=1,A=a](P[p=1|Y=1,A=a]P[Y=1|A=a] + P[p=1|Y=0,A=a](1-P[Y=1|A=a])) \end{aligned} \tag{3.26}$$

We can now solve for $P[Y=1|A=a]$ to obtain:

$$P[Y=1|A=a] = \frac{P[Y=1|p=1,A=a]P[p=1|Y=0,A=a]}{P[Y=1|p=1,A=a]P[p=1|Y=0,A=a]+(1-P[Y=1|p=1,A=a])P[p=1|Y=1,A=a]} \tag{3.27}$$

Using the three equalities exposed at the beginning of the proof we can demonstrate that $P[Y=1|A=0] = \frac{P[Y=1|p=1]P[p=1|Y=0]}{P[Y=1|p=1]P[p=1|Y=0]+(1-P[Y=1|p=1])P[p=1|Y=1]}$ in all cases where the denominator is not 0. In cases where it is, as both summands are positive, each has to be 0, resulting in a $\frac{0}{0}$ indetermination. Both conditions could happen at the same time, but the classification done will be really poor, as $P[Y=1|p=1,A=a] = 0 \forall a$. There is another appealing case, which is the case of a perfect classifier, where in fact both conditions can actually meet. In any other case, A and Y will be independent, contradicting the non-degeneracy hypothesis. □

3.3. Individual fairness

Individual fairness [Dwork et al., 2011] is based on the principle that similar individuals should be treated similarly. Therefore, the classes assigned to our individuals should be similar for similar individuals. This definition is inherently ambiguous, and, as can be inferred, the main problem that comes with this type of fairness measures lies in the need to define metrics to compare both individuals in the population and their labels, which inherently depends on the specific problem to be solved. There is a wide variety of these metrics with diverse characteristics, and it is not clear for a specific problem what type of metric to use. Different interpretations of the problem could lead to using different measures, resulting in different outcomes. Positive aspects regarding group fairness include the fact that it does not have subfamilies of measures that are inherently contradictory, and predictors that comply with them are less likely to "intentionally" make classification errors to balance conditional probabilities for each demographic subgroup.

To achieve a measure that complies with this definition, the first step is to define a way to measure the similarity between individuals, as well as a way to measure the similarity in predictions made about them. For this purpose, let us consider the representation of an individual by its feature vector, obtained as a sample realization of the random vector $(X_1, \ldots, X_n, A) \in \mathbb{R}^{n+1}$. To measure similarity between individuals, we will consider the notion of distance.

Definition 3.3.1 (Distance)**.** Let M be a set. A function $d\colon M \times M \to \mathbb{R}$ is a distance $\Leftrightarrow$ it satisfies the following properties:

- **Non-negativity**: $d(x_1, x_2) \geq 0, \forall x_1, x_2 \in M$
- **Non-degegeneracy**: $d(x_1, x_2) = 0 \Leftrightarrow x_1 = x_2, \forall x_1, x_2 \in M$
- **Symmetry**: $d(x_1, x_2) = d(x_1, x_2), \forall x_1, x_2 \in M$
- **Triangle inequality**: $d(x_1, x_2) \leq (x_1, x_3) + d(x_3, x_2) \forall x_1, x_2, x_3 \in M$

Definition 3.3.2 (Metric space)**.** The pair (M, d) where M is a set and $d\colon M \times M \to \mathbb{R}$ is a metric, is known as a metric space.

In the same way, we want to measure the distance between the predictions of our classifier. In this case, we will consider p to be a random classifier that transforms each individual in the population into a new probability function. This will induce a new probability space over the set of possible classification outcomes. Let us make a series of definitions to formalize this fact:

Definition 3.3.3 (Δ set associated with another set)**.** Let $E \subset \mathbb{R}$. The set Δ_E is defined as $\Delta_E = \{(E, \mathcal{P}(E), P) : P$ is a probability measure. In other words, it is the set of all possible probability spaces over the measurable space $(E, \mathcal{P}(E))$. We can also interpret this set as the set of all probability metrics defined over the measurable space $(E, \mathcal{P}(E)$.

We can use this definitions to define a predictor function. In this case, given the probability space $(\Omega, \mathcal{A}, P)$, and being the random variable to predict $Y\colon (\Omega, \mathcal{A}, P) \to (E \subseteq \mathbb{R}, \mathcal{B}, P_Y)$, our predictor function will be a function:

$$\begin{aligned} p\colon \Omega &\to \Delta_E \\ \omega &\to p(\omega) = (E, \mathcal{P}(E), P_\omega) \end{aligned} \tag{3.28}$$

So that each individual will be associated with a probability function P_Ω over the measurable space $(E, \mathcal{P}(E))$. Therefore, we will be interested in measuring the distance between the probability functions associated with each individual. The concept associated with this fact is called statistical distance [Martos Venturini, 2015].

Definition 3.3.4. (Statistical distance) A statistical distance is a measure of similarity between two statistical objects.

These objects can vary significantly in nature; they can be probability measures, samples, random variables... In our case, we will be interested in distances to compare probability measures. Statistical distances differ from traditional distances in that they do not necessarily satisfy the properties of distances; in general, they only satisfy a subset of them, although these conditions could also be relaxed. Examples of these include semi-distances, pseudodistances and quasi-distances. A subgroup with considerable relevance is that of divergences.

Definition 3.3.5 (Individual fairness)**.** Let $(\Omega, \mathcal{A}, P)$ be a probability space, $Y\colon (\Omega, \mathcal{A}, P) \to (E \subseteq \mathbb{R}, \mathcal{B}, P_Y)$ a random variable to predict, and $p\colon \Omega \to \Delta_E$ a probabilistic classifier. Consider a distance function $d_1\colon \Omega \times \Omega \to \mathbb{R}$ and a statistical distance $d_2\colon \Delta_E \times \Delta_E \to \mathbb{R}$. The predictor p satisfies the individual justice condition if

$$d_2(p(\omega), p(\omega')) \leq d_1(\omega, \omega'), \forall \omega, \omega' \in \Omega \tag{3.29}$$

This condition is also known as $(d_2, d_1)-$Lipschitz [Dwork et al., 2011].

Obviously, whether the individual justice condition is fulfilled or not depends on the chosen functions d_1 and d_2. In general, to measure the distance between individuals, we will measure the distances between their measured random feature vectors, that is, and according to the previous notation:

$$d_1(\omega, \omega') = d_1((X_1, \ldots, X_n, A)(\omega), (X_1, \ldots, X_n, A)(\omega')) \tag{3.30}$$

The metric d_2 must be a statistical distance. In [Dwork et al., 2011], two different divergences are proposed, independent of the problem definition, with which the justice-constrained optimization problem can be formulated as a polynomial-sized linear program with respect to the cardinalities of Ω and E:

Definition 3.3.6 (Total variation distance)**.** Let P_1, P_2 be two probability measures, both defined with respect to the same measurable space $(E, \mathcal{P}(E))$. The total variation distance between them is a statistical distance defined as::

$$d_{tv}(P_1, P_2) = \max_{F \subset E} |P_1(F) - P_2(F)| \tag{3.31}$$

Now, let us explore an equivalent way to express this measure, making the calculation much simpler for our context.

Lemma 3.3.0.1 (Characterization of total variation distance)**.** $d_{tv}(P_1, P_2) = \frac{1}{2} \sum_{e \in E} |P_1(e) - P_2(e)|$

Proof. Let $G = \{e \in E : P_1(e) \geq P_2(e)\}$. Let any event $F \subset E$. Then, the following inequality chain is satisfied:

$$P_1(F) - P_2(F) \leq P_1(F \cap G) - P_2(F \cap G) \leq P_1(G) - P_2(G) \tag{3.32}$$

The first inequality is true as, being $\overline{G}$ the complementary event of the event G, we know that every element $e \in F \cap \overline{G}$ are the only elements in F that follow $P_1(e) - P_2(e) < 0$, and this elements are the only removed in $F \cap G$. On the other hand, G has at least, as many elements as in $F \cap G$, and additionally, all of them meet $P_1(e) \geq P_2(e)$, with which we can obtain the second inequality. Analogously we can deduce that:

$$P_2(F) - P_1(F) \leq P_2(\overline{G}) - P_1(\overline{G}) \tag{3.33}$$

Furthermore, both bounds that have been set are the same, as $1 = P_1(G) + P_1(\overline{G}) = P_2(G) + P_2(\overline{G}) \Rightarrow P_1(G) - P_2(G) = P_2(\overline{G}) - P_1(\overline{G})$. We are able to reach the bound when we set $F = G$. Using all of the above we can rewrite:

$$d_{tv}(P_1, P_2) = \frac{1}{2}|P_1(G) - P_2(G) + P_2(\overline{G}) - P_1(\overline{G})| = \frac{1}{2}\sum_{e \in E}|P_1(e) - P_2(e)| \tag{3.34}$$

□

Corollary 3.3.0.1. *Total variation distance is a mathematical distance.*

Proof. All properties are trivial, except for triangle inequality. Let us consider 3 probability metrics P_1, P_2, P_3. Then, $d_{tv}(P_1, P_2) = \frac{1}{2}\sum_{e \in E}|P_1(e) - P_2(e)| = \frac{1}{2}\sum_{e \in E}|P_1(e) - P_3(e) + P_3(e) - P_2(e)| \leq \frac{1}{2}\sum_{e \in E}|P_1(e) - P_3(e)| + \frac{1}{2}\sum_{e \in E}|P_3(e) - P_2(e)| = d_{tv}(P_1, P_3) + d_{tv}(P_3, P_2)$ □

Definition 3.3.7 (l_∞ relative measure)**.** Let P_1, P_2 be two probability measures, defined over the same measurable space $(E, \mathcal{P}(E))$, where we will additionally require that $P_1(e), P_2(e) > 0, \forall e \in E$. Then l_∞ relative measure between both of them, in case E is continuous, is defined as:

$$d_\infty(P_1, P_2) = \sup_{e \in E} \log\left(\max\left\{\frac{P_1(e)}{P_2(e)}, \frac{P_2(e)}{P_1(e)}\right\}\right) \tag{3.35}$$

Corollary 3.3.0.2. l_∞ *relative measure is a distance.*

Proof. All properties are trivial but the triangle inequality. Let P_1, P_2, P_3 be probability measures. Using the additional hypothesis given at the definition of the measure, the set defined as $\left\{\log\left(\max\left\{\frac{P_1(e)}{P_2(e)}, \frac{P_2(e)}{P_1(e)}\right\}\right)\right\}, e \in E$ is upper bounded, and for that reason, $d_\infty(P_1, P_2)$ exists. The same condition will be needed for each relative set using P_1 and P_3, and P_2 and P_3. As all this sets are upper bounded, we can take a sequence $\{e_n\}_{n \in \mathbb{N}} \subseteq E$ such that $\lim_{n \to \infty} \log\left(\max\left\{\frac{P_1(e)}{P_2(e)}, \frac{P_2(e)}{P_1(e)}\right\}\right) = d_\infty(P_1, P_2)$. We can write this convergent sequence the following way:

$$\{e_n\}_{n \in \mathbb{N}} : e_n = \begin{cases} \log\left(\frac{P_1(e_n)}{P_2(e_n)}\right) = \log\left(\frac{P_1(e_n)}{P_3(e_n)}\right) + \log\left(\frac{P_3(e_n)}{P_2(e_n)}\right) & \text{if} \frac{P_1(e_n)}{P_2(e_n)} > \frac{P_2(e_n)}{P_1(e_n)} \\ \log\left(\frac{P_2(e_n)}{P_1(e_n)}\right) = \log\left(\frac{P_2(e_n)}{P_3(e_n)}\right) + \log\left(\frac{P_3(e_n)}{P_1(e_n)}\right) & \text{if} \frac{P_1(e_n)}{P_2(e_n)} \leq \frac{P_2(e_n)}{P_1(e_n)} \end{cases} \tag{3.36}$$

Using this sequence we can directly verify that:

$$\begin{aligned} \max\left\{\log\left(\frac{P_1(e_n)}{P_2(e_n)}\right), \log\left(\frac{P_2(e_n)}{P_1(e_n)}\right)\right\} \leq \\ \max\left\{\log\left(\frac{P_1(e_n)}{P_3(e_n)}\right), \log\left(\frac{P_3(e_n)}{P_1(e_n)}\right)\right\} + \max\left\{\log\left(\frac{P_3(e_n)}{P_2(e_n)}\right), \log\left(\frac{P_2(e_n)}{P_3(e_n)}\right)\right\} \end{aligned} \tag{3.37}$$

being this true, $\forall n \in \mathbb{N}$. For this reason, the sequence $\{e_n\}_{n\in\mathbb{N}}$ will help us to establish the following inequality:

$$\begin{aligned} d_\infty(P_1, P_2) &= \lim_{n\to\infty} \log\left(\max\left\{\frac{P_1(e)}{P_2(e)}, \frac{P_2(e)}{P_1(e)}\right\}\right) \\ &\leq \lim_{n\to\infty} \log\left(\max\left\{\frac{P_1(e)}{P_3(e)}, \frac{P_3(e)}{P_1(e)}\right\}\right) + \lim_{n\to\infty} \log\left(\max\left\{\frac{P_3(e)}{P_2(e)}, \frac{P_2(e)}{P_3(e)}\right\}\right) \\ &\leq d_\infty(P_1, P_3) + d_\infty(P_3, P_2) \end{aligned} \tag{3.38}$$

□

The first of the two employed comparison measures is bounded, while the latter is not. For the first one, two probability measures can be considered similar if the measure is close to 0, and different when it approaches 1. In the case of the second one, two distributions may be deemed similar when the measure is $\ll 1$, while they can be considered different if their associated measure is $\gg 1$.

Besides these two measures presented here for their good properties in implementing constraint optimization problems using them and their independence from context, many other context-independent measures can be found in [Deza and Deza, 2009]. The distance measure between individuals within the population could be proposed by an expert, as mentioned earlier and as stated in the original publication. However, recently, methods have been proposed to define such measures based on the datasets themselves [Ilvento, 2020], rather than being solely reliant on human expert proposals, although their definition may still involve information that only an expert can provide.

In [Ilvento, 2020], methods are developed based on the definition of submetrics to ensure the fulfillment of individual justice. Through the construction of these submetrics based on representative elements, it is ensured that the distance cost will not be overestimated, and the information required from our expert is also regulated.

Let us briefly delve into some of the theory in [Mukherjee et al., 2020]. Here, two relatively simple ways of learning metrics from data are proposed, but some level of expert information will always be necessary. These metrics will be based on the Mahalanobis distance:

$$d_1(\omega, \omega') = \langle \phi(x_1) - \phi(x_2), \Sigma(\phi(x_1) - \phi(x_2)) \rangle \tag{3.39}$$

where $\phi\colon \Omega \to \mathbb{R}^d$ is an injective function, and $\Sigma \in S_+^d$, being S_+^d the set of positive-semidefinite matrices of rank d.

3.3.1. FACE Method: Factor Analysis of Comparable Embeddings

The first proposed method is known as Factor Analysis of Comparable Embeddings, and it involves learning the value of Σ based on comparable samples. These samples can be extracted by experts or manually generated samples that differ based on certain meaningful forms.

In this case, the value of the function ϕ will consist of:

$$\phi_i = A_* x_i + B_* a_i + \epsilon_i \tag{3.40}$$

where $\phi_i \in \mathbb{R}^d$ is the representation of our individual, identified by the pair (x_i, a_i), and ϵ_i is an error term. A pair of samples will be comparable if their relevant attributes are similar. For example, in [Bolukbasi et al., 2016], where the goal is to eliminate bias in the context of word embeddings, words that only differ in gender context, such as he, she, or king, queen, are considered as similar.

This factorial model decomposes the variance of the representation ϕ_i into the variance due to the sensitive attribute and the variance due to the relevant non-sensitive attributes. The goal is to establish a technique that disregards the variance associated with the sensitive attribute and only considers the variance due to the rest of the attributes. This way, the metric will treat any pair of instances that only differ in the sensitive attribute equally. To demonstrate a possible choice of such a matrix, we will introduce a series of concepts:

Definition 3.3.8 (Orthogonal complement)**.** Let $(V, +, \cdot, K)$ be a vector space, and let us suppose that V is associated with the bilinear form $B : V \times V \to K$. The orthogonal complement of a subset $W \subseteq V$ is the set of all vector which are orthogonal to each vector in W. It is represented as follows:

$$W^{\perp} = \{x \in V : B(x, y) = 0, \forall y \in W\} \tag{3.41}$$

Lemma 3.3.0.2 (Orthogonal complement as subspace)**.** *The orthogonal complement $W^{\perp}$ of $W \subseteq V$, with the bilinear product B is a vector subspace of V.*

Proof. We will test if every vector subspace definition conditions are met:

- **Closed for sum**: $\forall x, y \in W^{\perp}, x + y \in W^{\perp}$. Let us consider $x, y \in W^{\perp} \Leftrightarrow B(x, z) = 0 \wedge B(y, z) = 0, \forall z \in W \Rightarrow B(x + y, z) = B(x, z) + B(y, z) = 0, \forall z \in W$, using the basic properties of bilinear forms.
- **Closed for scalar product**: $\forall x \in W^{\perp}, k \in K, kx \in W^{\perp}$.. As for every $x \in W^{\perp} \Leftrightarrow B(x, z) = 0, \forall z \in W$, using analogously the bilinear product basic properties, we can obtain that $\forall k \in K, B(kx, z) = kB(x, z) = 0, \forall z \in W$.

□

Lemma 3.3.0.3 (Vector space division using orthogonal complement)**.** *If W is a finite-dimensional vector subspace of V, then $V = W \oplus W^{\perp}$.*

Proof. Any vector $w \in W$ can be orthogonal to itself, unless it being 0, by definition of the bilinear form B, and for that reason $W \cap W^{\perp} = 0$. Let us now check that $W + W^{\perp} = V$. Let $U = W + W^{\perp}$. We can build an orthonormal basis of U, and extend it to a basis for V. If $U \neq V$, there exists at least one element e more in V's extended basis, and as it belong to the basis, it is orthogonal to every other basis component, and for that reason it is orthogonal to U. $W \subset U$, and as e is orthogonal to $W \Rightarrow e \in W^{\perp}$, and following the exact same reasoning, $e \in W \Rightarrow e = 0$, but if $e = 0$ it can not belong to any basis, reaching a contradiction. □

Definition 3.3.9 (Orthogonal projection)**.** Let W be a finite-dimensional subspace of V. Orthogonal projection of V over W is the lineal operator $P_W \colon V \to V$, described as follows: for each $v \in V$, we will write $v = w + w'$, where $w \in W$ and $w' \in W^{\perp}$. Then, $P_W(v) = w$.

A possible choice for matrix Σ is the projection matrix over the orthogonal complement of the subspace generated by the columns of matrix A_*. That subspace will be expressed as $V^c_{A_*}$. Doing so, if we consider two comparable elements, we will obtain:

$$\begin{aligned} d_1(w, w') &= \langle \phi_1 - \phi_2, (I - P_{V^c_{A_*}})(\phi_1 - \phi_2) \rangle \\ &\approx \langle B_*(v_1 - v_2), (I - P_{V^c_{A_*}}) B_*(v_1 - v_2) \rangle \end{aligned} \tag{3.42}$$

This choice ignores differences between ϕ_1 and ϕ_2 due to variations in sensitive attributes. Although the matrix $V^c_{A_*}$ is inherently unknown, it is possible to estimate it based on the learned representation and comparable sample groups. It is important to emphasize that the objective is $V^c_{A_*}$, not A_*, which leads to a simplification in the typical requirements of factor analysis. This allows for the application of an estimation process through a proper factor analysis procedure.

3.3.2. EXPLORE Method: Embedded Xenial Pairs Logistic Regression

The EXPLORE method learns a fair pairwise comparison metric. More specifically, the data comes in the form of triplets that encapsulate expert knowledge: $(x_{i_1}, x_{i_2}, y_i)_{i=1}^n$, where the value $y_i \in \{0, 1\}$ indicates whether an expert considers the data comparable $y_i = 1$ or not. It is hypothesized that (x_{i_1}, x_{i_2}, y_i) follows a binary response model of being:

$$y_i | x_{i_1}, x_{i_2} \sim \text{Ber}(2\sigma(-d_i)) \tag{3.43}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic function, which will be later exposed, and d_i is defined as:

$$\langle \underbrace{(\phi_{i_1} - \phi_{i_2})(\phi_{i_1}) - \phi_{i_2})^T}_{D_i}, \Sigma_0 \rangle \tag{3.44}$$

where ϕ_{i_1} and ϕ_{i_2} are the learned representations of x_{i_1} and x_{i_2} respectively, and matrix $\Sigma_0 \in S^d_+$ is needed to be estimated. In order to do that, the following function is proposed:

$$l_n(\Sigma) = \frac{1}{n} \sum_{i=1}^{n} y_i \log \left(\frac{2\sigma(-\langle D_i, \Sigma \rangle)}{1 - \sigma(-\langle D_i, \Sigma \rangle)} \right) + \log(1 - \sigma(-\langle D_i, \Sigma \rangle)) \tag{3.45}$$

As the function $l_n(\Sigma)$ is concave with respect to Σ, it is suggested to perform an iterative optimization process, such as stochastic gradient descent, to find the matrix that maximizes the function l_n.

3.4. Counterfactual fairness

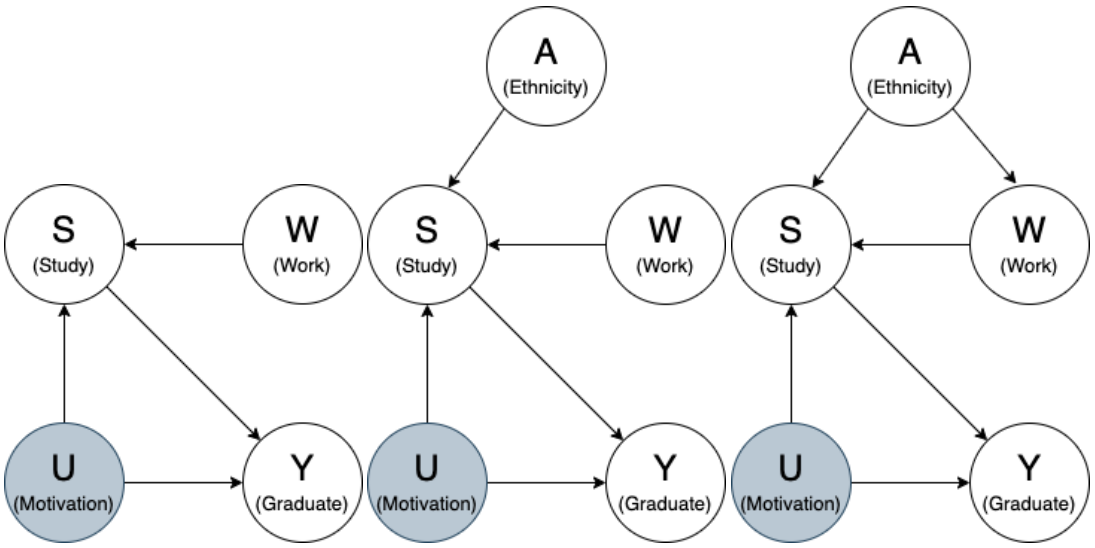
For the specification of counterfactual fairness, as outlined in [Kusner et al., 2018], we will need to predefine our problem using a causal model. Based on these causal models, we will be able to construct probabilities and consider how each element influences the system.

We will start defining what is a causal model [Galles and Pearl, 1998]:

Definition 3.4.1 (Causal model)**.** A causal model is a triplet, $M = \langle U, V, F \rangle$, where its components are:

- **Exogenous variables**: U is a set of variables called exogenous or unobservable variables, which will be determined by different factors external to the model.
- **Endogenous variables**: V is a set of variables called endogenous or observable variables, which will be determined by variables in the model. We denote them as $V = \{V_1, \ldots, V_i\}$.
- **Structural equations**: F is a set of functions $\{f_1, \ldots, f_n\}$ known as structural equations that must completely determine the value of variables V in terms of variables $U \cup (V \backslash V_i), \forall i \in \{1, \ldots, n\}$. It is often represented in the form $v_i = f_i(pa_i, u), \forall i \in \{1, \ldots, n\}$, where pa_i refers to the set of variables in $V \backslash V_i$ on which the value of V_i effectively depends, known as the parent variables of V_i.

Definition 3.4.2 (Causal graph)**.** Every causal model M has an associated causal graph $G(M)$, which is a directed graph where the variables U, V are the nodes, and the functions F establish directed edges between them. For each f_i, directed edges are formed from all variables on which that function effectively depends to V_i. It can also be verified that this graph is always acyclic since, if it were not, it would contradict the first property of set F. Examples of this can be seen in Figure 3.1.



(a) Causal graph associated with a causal model where there is no protected attribute considered.

(b) Causal graph associated with a causal model where there is one protected attribute considered.

(c) Causal graph associated with a causal model where there is one protected attribute considered, and expanded structural equations.

Figure 3.1.: Three examples of causal graphs.

Definition 3.4.3 (Causal submodel)**.** Let M be a causal model, let $X \subset V$, and let x be a set of different values that X can take. We will define the causal submodel M_x as the causal model given by the following tern: $M_{X \leftarrow x} = \langle U, V, F_{X \leftarrow x} \rangle : F_{X \leftarrow x} = \{f_i : V_i \notin X\} \cup \{X = x\}$

In other words, the functions f_i associated with variables $V_i \in X$ are removed, and they are replaced with constant functions that assign the corresponding value to each variable fixed in x. Additionally, it is necessary to establish the condition that these new structural equations must completely determine the value of the remaining free variables in V in terms of the variables U, as it cannot be guaranteed for any assignment x of any subset of variables X. A simple figure is shown in the figure 3.2.

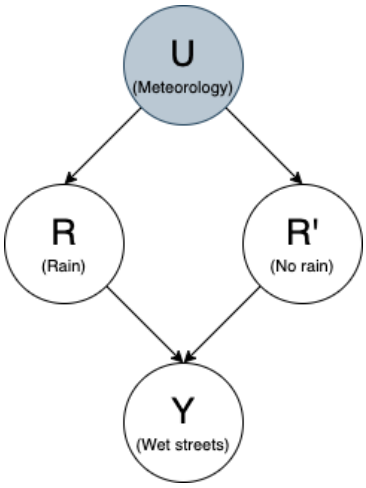


Figure 3.2.: For this causal model, if we consider $X = (L, L')$, the assignment $(0, 0)$, would not be valid, meaning false and false. Consequently, the value of the variable Y cannot be uniquely determined.

This assignment x to X is also known by the name of intervention, and it is usually denoted by $X \leftarrow x$. Complex interventions can be constructed applying logical expressions to simple interventions, and their notation extends to the one previously discussed. The interventions mentioned so far can be interpreted as a conjunction of interventions with respect to each of the individual variables forming X, assigning them their specific value from x. Only in these cases, and if the set of variables X is understood, the notation can be simplified, and we can write directly M_x and F_x. We will only deal with this last type of interventions.

Definition 3.4.4 (Effect of an intervention)**.** Let $M = \langle U, V, F \rangle$ be a causal model, X a set of variables in V, and x a specific realization of X. The effect of an intervention $X \leftarrow x$ on M is given by the causal model M_x.

Definition 3.4.5 (Potential response over an intervention)**.** Let $M = \langle U, V, F \rangle$ be a causal model, let $Y \in V$ be an observable variable, and $X \subset V$. The potential response of variable X to a specific intervention $X \leftarrow x$ is denoted as $Y_{X \leftarrow x}(u)$ or, in a simplified form, $Y_x(u)$ if the context allows. It represents the value that variable Y would take under the mentioned intervention and given a realization u of the unobservable variables U. Since M_x is a causal model, $Y_x(u)$ will take a unique value.

Definition 3.4.6 (Counterfactual)**.** Let $M = \langle U, V, F \rangle$ be a causal model, and let $Y \in V$ be an observable variable, and $X \subset V$. The value that the variable Y would have taken under the intervention $X \leftarrow x$, also known as the counterfactual value under that context, is represented by the potential response value $Y_x(u)$.

Definition 3.4.7 (Probabilistic causal model)**.** A probabilistic causal model is a pair $\langle M, P[U = u]\rangle$, where M is a causal model and $P[U = u]$ is a probability measure over the domain of U."

Now, the values of the observable variables are still determined by the values of the structural equations in F, but we consider the unobservable variables as random variables to which we assign probabilities. In this way, we can define probabilities over the observable variables as follows:

$$P[Y = y] = \sum_{\{u:Y(u)=y\}} P(u) \tag{3.46}$$

where $Y(u)$ indicates the value of variable Y based on the realization u of the unobservable variables. Similarly, we can define counterfactual probabilities in an analogous manner:

$$P[Y_x = y] = \sum_{\{u:Y_x(u)=y\}} P(u) \tag{3.47}$$

We can additionally define joint probabilities. For example, we can define probabilities $P[Y_x = y, X = x']$, and in case $x \neq x'$, these probabilities are naturally defined as follows:

$$P[Y_x = y, X = x'] = \sum_{\{u:Y_x(u)=y \wedge X(u)=x'\}} P(u) \tag{3.48}$$

$$P[Y_x = y, Y_{x'} = y'] = \sum_{\{u:Y_x(u)=y \wedge Y_{x'}(u)=y'\}} P(u) \tag{3.49}$$

For the calculation of conditional probabilities, we might encounter issues if there are variables assigned in an incompatible way with the conditioning variable. However, in [Pearl, 1999], a method for calculating these conditional probabilities is demonstrated as follows:

$$\begin{aligned} P[Y_{x'} = y' | X = x, Y = y] &= \frac{P[Y_{x'} = y', X = x, Y = y]}{P[X = x, Y = y]} \\ &= \sum_u P[Y_{x'}(u) = y'] P[u = U | x = X, y = Y] \end{aligned} \tag{3.50}$$

In this way, for each possible realization that the unobservable variables u can take, we should calculate their conditional probability given that $x = X, y = Y$, and then multiply it by the probability that variable Y takes the value y' according to the model $M_{x'}$.

In order to define the concept of counterfactual fairness, we must consider our variables X, A as observable variables in the model. Additionally, our predictor will be a function of these variables, so we can include it as an observable variable, and therefore, $X \cup A \cup p = V$.

Definition 3.4.8 (Counterfactual fairness)**.** Let $M = \langle U, V, F\rangle$ be a causal model, under the conditions previously described. The predictor p is said to satisfy the counterfactual fairness condition if:

$$P[p_{A \leftarrow a}(U) | X = x, A = a] = P[p_{A \leftarrow a'}(U) | X = x, A = a] \tag{3.51}$$

for each possible value y of Y and each possible value a' of A.

This condition can be interpreted as the value of A should not be the cause of the value of p for any given instance. In other words, changing the value of A while keeping the rest of

the values for variables that do not directly depend on the value of A should not change the probability distribution of p.

The main advantage of using this fairness measure is that it proposes a way to understand the causes of possible bias through the causal graph. Thus, by changing the value of the attribute A, we can see how the rest of the variables that depend on its value are influenced. For this reason, it eliminates all possible problems of justice due to unawareness. On the other hand, it also does not require the definition of additional metrics, as individual fairness does.

However, the main problem of using this model lies in the definition of the causal model. Indeed, the model gives us an idea of how the rest of the variables are influenced by fixing a certain value of A, but this influence, based on the model, is determined by the model we consider, which may not be equal to the underlying real model. Constructing a correct causal model requires an in-depth knowledge of the context of the problem and a clear understanding of the dependencies between variables to define the structural relations, which is generally challenging. Therefore, although the idea is theoretically robust, in practice, it will have a more limited application than other fairness criteria. However, if the causal model is known with certainty, it could be the best fairness measure among all the proposed ones.

4. Multiobjective optimization and many objectives optimization

In this chapter, the mathematical theory supporting multiobjective optimization and Many Objectives optimization will be studied. Concepts such as Pareto dominance and quality metrics for approximation sets will be discussed.

4.1. Multiobjective optimization problems

In this section we will show some basic definitions about multiobjective optimization problems.

Definition 4.1.1 (Multiobjective optimization problem (MOP)). . Let $\Omega \subset \mathbb{R}^n, \Theta \subset \mathbb{R}^m, m > 1$. Let $f\colon \Omega \to \Theta$, with $f(\omega) = (f_1(\omega), \ldots, f_m(\omega)), \forall \omega \in \Omega$, with each $f_i\colon \Omega \to \mathbb{R}, \forall i \in \{1, \ldots, m\}$. A multiobjective optimization problem consists of:

$$\min_{\omega \in \Omega} \text{ or } \max_{\omega \in \Omega}(f_1(\omega), \ldots, f_m(\omega)) \tag{4.1}$$

Depending on wether we want to minimize or maximize, we will refer to a multiobjective minimization or maximization problem respectively.

This definition can lead to ambiguity, as there is not a specified order relation in $\Theta \subset \mathbb{R}^m, m > 1$, from which we can compare the image of the function f, and we can not directly induce the usual order relation in $\mathbb{R}$ that enables us to use maximum and minimum. The comparison criterion from which we can define maximum and minimum will be later discussed.

Definition 4.1.2 (Decision or search space). The set $\Omega \subset \mathbb{R}^n$ used in definition 4.1.1 is known as decision or search space.

Definition 4.1.3 (Objective space). The set $\Theta \subset \mathbb{R}^m, m > 1$ used in definition 4.1.1 is known as objective space.

Definition 4.1.4 (Dimension of a multiobjective optimization problem). The dimension of a multiobjective optimization problem is the value $m > 1$ in definition 4.1.1.

Definition 4.1.5 (Objective functions). Each $f_i : \Omega \to \mathbb{R}, i \in \{1, \ldots, m\}$ in definition 4.1.1 is known as an objective function. Following this notation, the objective functions of a multiobjective optimization problem are $\{f_1, \ldots, f_m\}$. They will be also known as objectives.

4.2. Dominance of solutions in multiobjective optimization problems

We need a comparison method for images of the objective function f, as stated before, in order to determine the optimal value that we search, which can be a maximum or minimum.

We will now expose that criterion, and we explain concepts around optimal solutions in a multiobjective optimization problem.

Definition 4.2.1 (Feasible solution of a multiobjective optimization problem)**.** A feasible solution of a multiobjective optimization problem, or simply a solution, is an element $\omega \in \Omega$.

The criterion of solution dominance will take into account that the image by the objective function of each solution is in $\mathbb{R}^m, m > 1$, and there may be some solutions that are better in some objectives but worse in others. This occurs in multiobjective problems of interest, where the objectives are conflicting. That is, although we may improve all objective functions uniformly for a while, there will come a point where if we want to continue improving a particular objective or set of objectives, there must necessarily be a trade-off in the rest of the objectives, which will worsen in our optimization. Examples could include maximizing user comfort and energy savings in a heating system, or simultaneous minimization in a production chain of unit time in the chain, unit cost, and product variability with respect to the quality standard.

To continue defining concepts, we will need to focus on either minimization or maximization problems. We will consider minimization problems, as the practical study we will undertake is centered around minimization problems, but definitions for maximization problems are analogous.

Definition 4.2.2 (Pareto dominance)**.** Let $\psi, \omega \in \Omega$ be solutions to a multiobjective minimization problem. ψ Pareto dominates ω, or simply ψ dominates ω, noted as $\psi \prec \omega$, if and only if the following properties are met:

1. $f_i(\psi) \leq f_i(\omega), \forall i \in \{1, \ldots, m\}$.
2. $\exists j \in \{1, \ldots, m\} : f_j(\psi) < f_j(\omega)$.

Definition 4.2.3 (Order relation)**.** A binary relation R over a set X is an order relation if and only if the following properties are met:

- **Reflexivity**: $aRa, \forall a \in X$.
- **Antisymmetry**: $aRb, bRa \Rightarrow a = b, \forall a, b \in X$.
- **Transitivity**: $aRb, bRc \Rightarrow aRc, \forall a, b, c \in X$.

Lemma 4.2.0.1 (No ordering in Pareto dominance)**.** *Pareto dominance relation is not an order relation.*

Proof. It is an irreflexive relation. By contradiction, if $\exists \omega \in Omega : \omega \prec \omega \Rightarrow \exists j \in 1, \ldots m :$ $f_j(\omega) < f_j(\omega)$, which is a contradiction. □

As it is not an order relationship, we cannot define a partially ordered set from which to obtain lower bounds or the minimum. In fact, even if we were to eliminate the second condition in the definition of Pareto dominance, we could not guarantee that this relationship would be one of order. This is because, even if we achieved reflexivity, we would need injectivity in each function f_i to ensure antisymmetry. What we will do is directly provide a definition with a notion equivalent to that of lower bound, which will allow us to precisely define what we are seeking in our optimization process.

Definition 4.2.4 (Pareto-optimal solution)**.** Let $\omega \in \Omega$ be a solution of a multiobjective minimization problem. ω is a Pareto-optimal solution $\Leftrightarrow \forall \omega' \in \Omega, \omega' \nprec \omega$

If a solution is Pareto-optimal, it means that there is no other solution that is better in at least one objective without worsening for another objective with respect to the Pareto-optimal solution. Therefore, these solutions are not improvable in all objectives and represent a limit in our optimization. This is the reason why our goal is to find the greatest set of this solutions. As we have mentioned, there are solutions that may not be comparable to each other, so it will be natural to have multiple such solutions in our search space. We will consider these as the best solutions, and they will be our search objective, allowing the end users to choose a preferred solution based on their criteria.

Definition 4.2.5 (Pareto-optimal set)**.** The Pareto-optimal set ($\mathcal{P}$) is the set of Pareto-optimal solutions, $\mathcal{P} = \{\omega \in \Omega : \forall \omega' \in \Omega, \omega' \nprec \omega\}$

Definition 4.2.6 (Pareto front)**.** The Pareto front ($\mathcal{PF}$) is the set of images of the elements from the Pareto-optimal set under f: $\mathcal{PF} = \{f(\omega) : \omega \in \mathcal{P}\}$

By studying the Pareto front, we can observe and understand how the objectives relate to each other, as well as how the objectives of these solutions behave, providing us with a criterion for choosing among them. However, finding the entire Pareto front is computationally expensive, as to identify each element of the Pareto front, we would need to compare it with the rest of the elements in Ω, which is impractical. Therefore, we will introduce the following concepts:

Definition 4.2.7 (Approximation set)**.** An approximation set is a set $A \subseteq \Omega : \forall a_1, a_2 \in A$, with $a_1 \neq a_2 \Rightarrow a_1 \nprec a_2 \wedge a_2 \nprec a_1$.

Definition 4.2.8 (Pareto dominance in approximation sets)**.** Let $A, B \subseteq \Omega$ be approximation sets. A dominates B, which is written $A \prec B \Leftrightarrow \forall a \in A \exists b \in B : a \prec b$.

Lemma 4.2.0.2 (Pareto dominant subset)**.** *For any non-empty subset $B \subseteq \Omega, \exists A \subseteq B$, where A is a non-empty approximation set, and $A \prec B \backslash A$, in case $A \neq B$.*

Proof. $A = \{a \in B : \nexists b \in B : b \prec a\}$. A is an approximation set, as for any two distinct elements $a_1, a_2 \in A, a_1 \neq a_2 \Rightarrow a_1 \nprec a_2 \wedge a_2 \nprec a_1$, by the definition of A, and for that reason A is an approximation set. A is non-empty, as in any other case, $\forall b \in B \exists b' \in B : b' \prec b$, which is impossible. By contradiction, let us suppose A is empty. Let $b_0 \in B$ be a fixed element of B. Then, $\exists b_1 \in B : b_1 \prec b_0$. That $b_1 \neq b_0$, as $\prec$ is irreflexive. For that b_1 there is another $b_2 \in B : b_2 \prec b_1 \prec b_0$. $b_2 \neq b_1$, using the same reasoning as before, and $b_2 \neq b_0$ because $b_2 \prec b_0$ using transitivity, and we can apply again that $\prec$ is irreflexive. For that reason, we can create a chain $b_{|B|} \prec \cdots \prec b_0$, where all are distinct elements, and therefore every B element is present. But in that case, by definition of A, $b_{|B|} \in A$, as if $b_{|B|} \notin A$ that would mean that $\exists b \in B : b \prec b_{|B|}$. But because of how was $b_{|B|}$ chosen, $b_{|B|} \prec b$, and that would contradict that $\prec$ is irreflexive.

To proof that $A \prec B \backslash A$, in case that $A \neq B$, let us see that $\forall b \in B \backslash A, \exists a \in A : a \prec b$. To do that, the following procedure is considered: Let us suppose that there is an element $b \in B \backslash A$. By definition of A, we know that $\exists b_0 \in B : b_0 \prec b$. If $b_0 \in A$, we have finished. In any other case, $b_0 \in B \backslash A$, and for that reason $\exists b_1 \in B : b_1 \prec b_0$. We can repeat this procedure as much as needed. As B is a finite set, this procedure has to end using a finite amount of steps. □

In case B is an infinite set, this previous lemma is not true. We can find a simple counterexample, using the multiobjective minimization problem $f\colon \mathbb{R}^2 \to \mathbb{R}^2$, where f is the identity function. In this case, we could choose an infinite set B containing every element of the sequence $\{x_n\}_{n\in\mathbb{N}} : x_n = (\frac{1}{2^n}, \frac{1}{2^n})$. Given a fixed $m \in \mathbb{N}, x_r \prec x_m, \forall r > m$. But the set $\{r \in \mathbb{N} : r > m\}$ is an infinite set, as $\mathbb{N}$ is a non-majorized infinite set. So B is a set in which each element has an infinite set of elements that dominates it.

Definition 4.2.9 (Front of level 0)**.** Given a finite non-empty set $B \subseteq \Omega$, the front of level 0 of B, which is written as B_0, is the set $f(a) : a \in A$, where A is the set found in lemma 4.2.0.2 .

With this definition it is clear that if B is an approximation set, $B_0 = B$. Once found B_0 from set B, and in case $B \backslash B_0$ is not empty, we can repeat the process done in the previous definition iteratively, up to reach an empty set. And as B is finite, we will end in a finite amount of steps. Having observed this, the following definition is natural:

Definition 4.2.10 (Front of level n)**.** Given a non-empty finite subset $B \subseteq \Omega$, the front of level n of B, which is written as B_n, is the set B_0 with respect to the set $B \cup_{i=0}^{n-1} B_i$, in case it is a non-empty set.

Definition 4.2.11 (Optimization goal in a MOP)**.** The optimization goal in a MOP will be to find an approximation set $A \subset \Omega$ meeting the following conditions:

- **Convergence**: Every $a \in A$ has to be as close as possible to $\mathcal{P}$.
- **Spread**: Every $a \in A$ has to be as diverse as possible within the objective space.

This definition is very ambiguous. To begin with, if we want to solve the problem, we don't know the set $\mathcal{P}$, so there is no actual way to measure how close we are to $\mathcal{P}$. We understand closeness as dominance, so as close the meaning of as possible to $\mathcal{P}$ would mean having the least amount of other solutions which dominates them. But our interest not only relies on this dominance criterion, but also on how well distributed are with respect to the objective space. The distribution has to be as spread and uniform as possible, and describe the best as possible the actual shape of $\mathcal{P}$. This idea of having well distributed sets is enhanced as, even we could extend the concept of dominance to approximation sets, there are lots of incomparable pairs of sets in terms of this Pareto dominance. So it is important to define a set of other quality measures in order to better evaluate the approximation sets found, which will be the aim of the following section.

4.3. Quality metrics over a solution set

Definition 4.3.1 (Quality metric)**.** A k-ary quality metric is a function $Q\colon \mathcal{P}(\Omega)^k \to \mathbb{R}$.

Currently, there are a large number of quality measures, some of which are outlined in [Li et al., 2014, Audet et al., 2020, Van Veldhuizen, 1999]. Quality measures can be classified based on the aspect they attempt to measure on the given sets in their arguments. We can make the following classification, which is not mutually exclusive:

- **Cardinality**: Cardinality of the set with respect to a certain feature.
- **Convergence**: Convergence to the Pareto-optimal set.
- **Uniformity**: Uniformity in the distribution of the set's elements.

- **Diversity**: Diversity within the wider portion of objective space as possible.

For the calculation of some of these measures, knowledge of the Pareto-optimal set $\mathcal{P}$ is assumed. Since, in most cases, and particularly in our study, we do not know it, we will treat such k-ary quality measures as (k+1)-ary measures. That last parameter will be a set that we know and that approximates the set $\mathcal{P}$. When this happens, we will denote the last parameter as P_{ref} to reference this fact.

Let us now look at the definitions of some of the most common quality measures, which we will be used in our analysis:

Definition 4.3.2 (Hypervolume or $\mathcal{S}$ metric)**.** Without loss of generality, let us consider a multiobjective minimization problem with $f : \Omega \to (0,1)^m$. Hypervolume is an unary quality metric, $\mathcal{H} : \mathcal{P}(\Omega) \to (0,1)$, such that for every solution set $A \subseteq \Omega$,

$$\mathcal{H}(A) = \int_{(0,1)^m} \mathbb{1}_{D_A}(z)dz \tag{4.2}$$

being $D_A = \{z = (z_1, \ldots, z_m) \in (0,1)^m : \exists a \in A : a \prec z\}$ and $\mathbb{1}_{D_A}$ the characteristic function of the set D_A.

Hypervolume measures the quality of the given set in its argument with respect to the convergence, diversity, and uniformity of its solutions. The main problem with this measure is that its value depends heavily on the shape of the Pareto front, which can favor certain solutions over others. In terms of all these aspects, we can show examples where, despite having a more diverse / uniformly distributed / approximate set to the Pareto-optimal set, we have a lower hypervolume, although this generally does not occur. Regarding convergence, however, we have a monotonicity property which almost any other metric with such good properties has, among those widely used nowadays:

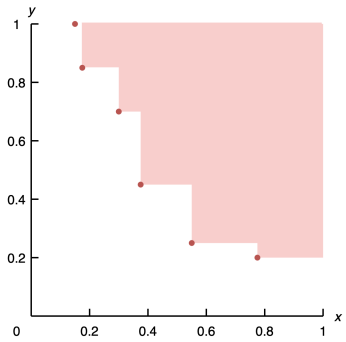


Figure 4.1.: 2D example of the hypervolume for a given solution set, represented using dots.

Lemma 4.3.0.1 (Hypervolume monotony)**.** *Let $A, B \subseteq \Omega$ be two finite approximation sets. If $A \prec B$, and $\mathcal{H}(A) = \mathcal{H}(B) = 0$ does not happen, then $\mathcal{H}(A) > \mathcal{H}(B)$.*

Proof. As $A \prec B \Rightarrow \forall b \in B \exists a \in A : a \prec b$. Therefore, $\forall z \in (0,1)^m : \mathbb{1}_{D_B}(z) = 1 \Rightarrow \exists b \in B : b \prec z$, and as $\exists a \in A : a \prec b \Rightarrow \mathbb{1}_{D_A}(z) = 1$, then $\mathcal{H}(A) \geq \mathcal{H}(B)$. We will now proof that $\mathcal{H}(\mathcal{A}) \gneq \mathcal{H}(B)$. To achieve this, we will find a measurable non-null set S which will be

contained in D_A and not in D_B. Let us take any $b_{\in}B$, and its respective $a \in A : a \prec b$, which exists as $A \prec B$. Now, $\forall i \in \{1,\dots,m\}$, if $f_i(b) \neq \max\{f_i(b') \in \mathbb{R} : b' \in B\}$ (that maximum exists $\forall i \in \{1,\dots,m\}$ as B is a finite set), then there exists at least one element, which will be expressed as $b^i \in B$, which meets that $f_i(b^i) = \min\{f_i(b') \in \mathbb{R} : b' \in B \wedge f_i(b') > f_i(b)\}$. As $a \prec b \Rightarrow f_i(a) \leq f_i(b), \forall i \in \{1,\dots,m\}$. We will now introduce a set of elements $\{d_1,\dots,d_m\}$ where:

$$d_i = \begin{cases} f_i(b) - f_i(a) & \text{if } f_i(b) - f_i(a) > 0 \\ f_i(b^i) - f_i(a) & \text{if } f_i(b) - f_i(a) = 0 \wedge \exists b^i \\ 1 - f_i(a) & \text{if } f_i(b) - f_i(a) = 0 \wedge \nexists b^i \end{cases} \tag{4.3}$$

Each of this d_i is always strictly greater than 0 unless that $f_i(b) - f_i(a) = 0 \wedge f_i(a) = f_i(b) = 1$, and consequently their hypervolume contribution for both a and b will be 0. In that case, we have to select another point. If all points meet that condition then $\mathbb{H}(A) = \mathbb{H}(B) = 0$, being this the only exception.

If that is not the case, we selected a point b for which this does not happen. Then, $\forall z = (z_1 \dots z_m) \in \mathcal{S} = (f_1(a), f_q(a) + d_1) \times \cdots \times (f_m(a), f_m(a) + d_m), \mathbb{1}_{D_A}(z) = 1 \wedge \mathbb{1}_{D_B}(z) = 0$. Indeed, $\mathbb{1}_{D_A}(z) = 1$ as $z_i > f_i(a) \forall i \in \{1,\dots,m\}$ and $a \in A$. We will prove that $\mathbb{1}_{D_B}(z) = 0$ by contradiction, so let us suppose that $\mathbb{1}_{D_B}(z) = 1 \Rightarrow \exists \tilde{b} \in B : f_i(\tilde{b}) \leq z_i \forall i \in \{1,\dots m\} \wedge \exists j \in \{1,\dots,m\} : f_j(\tilde{b}) < z_j\}$.

In case any d_i is defined using the third case ($\nexists i \in \{1,\dots,m\} : f_i(b) - f_i(a) = 0 \wedge \nexists b^i$), we have to observe that $a \prec b \Rightarrow \exists j \in \{1,\dots,m\} : f_j(a) < f_j(b)$. Therefore, $z_j \in (f_j(a), f_j(b)) = (f_j(a), f_j(a) + d_j) \Rightarrow z_j < f_j(b)$, and in case that $\tilde{b}$ exists, for it to be $f_j(\tilde{b}) \leq z_j$, at least $f_j(\tilde{b}) < f_j(b)$ needs to happen. For the rest of objectives, depending on how was its respective d_i defined:

- If $d_i = f_i(b) - f_i(a)$, then we can follow a similar reasoning and $f_i(\tilde{b}) < f_i(b)$.
- If $d_i = f_i(b^i)$, following an analogous reasoning, then we obtain that $f_i(\tilde{b}) < f_i(b^i)$, but in this case, using the definition of b^i, and for $\tilde{b} \in B \Rightarrow f_i(\tilde{b}) \leq f_i(b)$.
- If $d_i = 1 - f_i(a)$ then $f_i(b) - f_i(a) = 0 \wedge \nexists b^i \Rightarrow f_i(b) = f_i(a) \wedge f_i(b) = \max\{f_i(b') : b' \in B\}$. Because of this reasoning and for $\tilde{b} \in B$, it needs to happen that $f_i(\tilde{b}) < f_i(b)$.

Consequently, $\tilde{b}$ needs to meet that $\forall i \in \{1,\dots,m\}, f_i(\tilde{b}) \leq f_i(b) \wedge \exists j \in \{1,\dots,m\} : f_j(\tilde{b}) < f_j(b) \Rightarrow \tilde{b} \prec b$, which contradicts that B is an approximation set.

To summarize, it has been proven that $\mathbb{1}_{D_B}(z) = 0$. Therefore it is immediate to deduce that $\mathcal{S} \in D_A \backslash D_B$. Additionally, as $(S) = (f_1(a), f_1(a) + d_1) \times \cdots \times (f_m(a), f_m(a) + d_m)$, and each $d_i > 0$, the following is met:

$$\begin{aligned} \mathcal{H}(A) - \mathcal{H}(B) &= \int_{(0,1)^m} \mathbb{1}_{D_A}(z)dz - \int_{(0,1)^m} \mathbb{1}_{D_B}(z)dz = \int_{(0,1)^m} \mathbb{1}_{D_A}(z) - \mathbb{1}_{D_B}(z)dz \\ &= \int_{(0,1)^m} \mathbb{1}_{D_A \backslash D_B}(z)dz \geq \int_{(0,1)^m} \mathbb{1}_{\mathcal{S}}(z)dz = \prod_{i=1}^{n} d_i > 0 \Rightarrow \mathcal{H}(A) > \mathcal{H}(B) \end{aligned} \tag{4.4}$$

□

We can see some images that can help us understand the procedure carried out in the proof in Figure 4.2. Additionally, we can observe the case where $\mathcal{H}(A) = \mathcal{H}(B) = 0$ in Figure 4.3.

The reciprocal of this result is not true, and can be proven taking $A = (B \cup a)_0, a \in \Omega$, with A continuing being an approximation set.

Definition 4.3.3 (Error ratio)**.** Error ratio is an unary quality metric. $ER\colon \mathcal{P}(\Omega)^2 \to [0,1]$ such that:

$$ER(A, P_{ref}) = 1 - \frac{|\{a \in A :\in P_{ref}\}|}{|A|} \tag{4.5}$$

which is the portion of elements from A which not belong to P_{ref}. The lower this value, the more solutions in A will also be in P_{ref}, which is preferable.

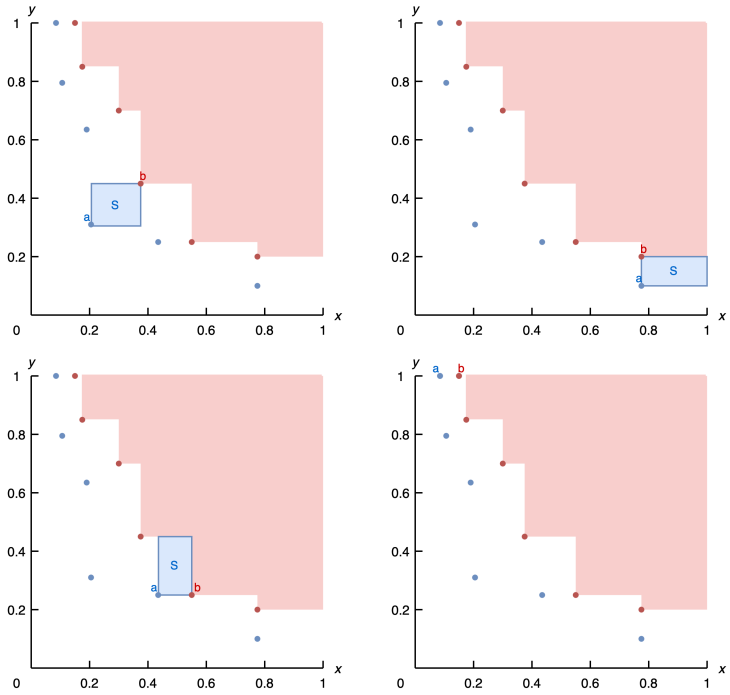


Figure 4.2.: Building of set S used in the proof of lemma 4.3.0.1, depending on the chosen solution b, and its associated solution a (identified by their objectives). Red solutions belong to B set, while blue solutions belong to A set. There are 4 possible scenarios, where only the last one forces us to choose another solution b.

It could happen that there were elements in our set which were solutions belonging P_{ref}, but due to rounding errors, they are not considered equal. For that reason we will introduce a margin of error ϵ when comparing both sets. Therefore, the actual metric is calculated as: $ER(A, P_{ref} = 1 - \frac{|\{a \in A : \exists b \in P_{ref} : ||a-b|| < \epsilon\}|}{|A|}$.

Definition 4.3.4 (Proportion)**.** Proportion is an binary quality metric, $PROP\colon \mathcal{P}(\Omega)^2 \to [0,1]$ such that:

$$PROP(A, P_{ref}) = 1 - \frac{|\{a \in A : a \in P_{ref}\}|}{|P_{ref}|} \tag{4.6}$$

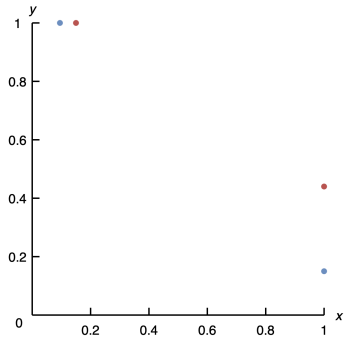


Figure 4.3.: Example of B (red solutions) and A (blue solutions) sets where $A \prec B$ and $\mathcal{H}(A) = \mathcal{H}(B) = 0$.

which is the proportion of elements from P_{ref} that belong to A. The greater it is, the more elements from P_{ref} are also in A, which is preferable.

We can relax this metric exactly the same way as we did with the error ratio, adding a constant ϵ which represents an error margin.

Definition 4.3.5 (Generational distance)**.** Generational distance is a binary quality metric, $GD\colon \mathcal{P}(\Omega)^2 \to \mathbb{R}_0^+$, such that:

$$GD(A, P_{ref}) = \frac{(\sum_{i=0}^{|A|} d_i^t)^{1/t}}{|A|} \tag{4.7}$$

where t is a fixed integer and $d_i = \min\{d(f(x_i), f(y)) : y \in P_{ref}\}$. d can be any euclidean distance, but for our purpose, we will consider the distance associated with the euclidean norm in $\mathbb{R}^m$.

The bigger t is, the lower generational distance will be. Absolute difference between 2 solutions will be lower, whereas relative difference will increase. Usual values of t are 1 and 2. We will consider the value $t = 2$, resulting in the following metric:

$$GD(A, P_{ref}) = \frac{(\sum_{i=0}^{|A|} \min_{y \in A} ||f(x_i) - f(y)||_2^2)^{1/2}}{|A|} \tag{4.8}$$

Generational distance is a clear convergence metric to the Pareto-optimal set (our goal set), it is simple, direct and widely used in the field of multiobjective optimization.

Definition 4.3.6 (Inverted generational distance)**.** Inverted generational distance is a binary quality metric, $IGD\colon \mathcal{P}(\Omega)^2 \to \mathbb{R}_0^+$, such that:

$$IGD(A, P_{ref}) = \frac{(\sum_{i=0}^{|P_{ref}|} d_i^t)^{1/t}}{|P_{ref}|} \tag{4.9}$$

where t is a fixed integer and $d_i = \min\{d(f(x_i), f(y)) : y \in A\}$. d can be any euclidean distance, but for our purpose, we will consider the distance associated with the euclidean

norm in $\mathbb{R}^m$. Just the same as before, we will take $t = 2$, and for that reason the final metric will be:

$$IGD(A, P_{ref}) = \frac{(\sum_{i=0}^{|P_{ref}|} ||f(x_i) - f(y)||_2^2)^{1/2}}{|P_{ref}|} \tag{4.10}$$

Inverted generational distance is also a measure of convergence, but it takes a different approach compared to generational distance. While the latter evaluated only convergence, this measure is sensitive to both convergence and the representativeness of solutions from A with respect to P_{ref}. If there is a region in P_{ref} that has not been explored by the solutions in A, inverted generational distance would increase, whereas generational distance might not necessarily do so.

Definition 4.3.7 (Maximum spread)**.** Maximum spread is an unary quality metric, $MS : \mathcal{P}(\Omega) \to \mathbb{R}_0^+$, such that:

$$MS(A) = \sqrt{\sum_{i=0}^{m} \max_{a,a' \in A} (a_i - a'_i)^2} \tag{4.11}$$

Maximum spread is a distribution metric. The higher it is, the more distant values have been found, so a wider portion of the objective space has been explored.

In order to define the next quality metric, named overall Pareto spread, we will introduce some preliminary definitions:

Definition 4.3.8 (Ideal point)**.** Given a multiobjective optimization problem and let $f = (f_1, \dots f_m)$ its objective function, the ideal point (F^I) is the point with the following coordinates: $F_i^I = \min_{x \in \Omega} f_i(x)$. This is the solution to the single objective minimization problem, considering only f_i as the objective function and ignoring the rest.

If we want to find F^I, then m single objective minimization problems have to be solved. However, this is not always feasible, as these problems can be challenging due to the complexity of either Ω, f_i, or both. Metaheuristics emerge as the primary method not to solve this problems directly but to obtain good approximations. Nevertheless, such approximations may not meet the required precision. The execution time to solve these problems is typically unacceptable.

Because of that, the practical approach to using F^I involves utilizing approximations instead of the real value of F^I. The next property will help us to define a good approximation criterion. Let $C^I = \{x \in \Omega : \exists i \in \{1, \dots m\} : f_i(x) \leq f_i(x'), \forall x' \in \Omega\} = \{x \in \Omega : \exists i \in \{1, \dots m\} : f_i(x) = F_i^I\}$. With this set, we can do the following observation:

Lemma 4.3.0.2 (Single minimization solutions and Pareto-optimal set)**.** *The set* $C_0^I \subseteq \mathcal{P}$

Proof. Let $p \in C_0^I$. As $p \in C^I \supseteq C_0^I \Rightarrow \exists i \in \{1, \dots m\} : p_i \leq \omega_i, \forall \omega in \Omega$. Additionally, as $p \in C_0^I$ it is not dominated by any other element from C^I. Furthermore, is it not dominated by any other element from $\Omega \backslash C^I$, as all elements from this set have strictly greater values in coordinate i compared to p_i. This implies that p is not dominated by any element in $(\Omega \backslash C^I) \cup C^I = \Omega \to p \in \mathcal{P}$. □

Thanks to this lemma, if we had the real $\mathcal{P}$, then the calculation of F^I would be trivial. Using this information, we can approximate it by obtaining an approximation set as the solution of the problem, and the approximation would be $F^{I*} = \min_{x \in A} f_i(x)$. If the approximation set is good enough, then F^{I*} will be as well.

Definition 4.3.9 (Nadir point)**.** Given a multiobjective optimization problem, and let $f = (f_1, \dots f_m)$ be its objective function, the nadir point (F^I) is the point having the following coordinates: $F_i^N = \max_{x \in \Omega} f_i(x)$. This is the solution to the single objective maximization problem, considering only f_i as the objective function, ignoring the rest.

The nadir point tells us what is the worst possible value that a specific objective can reach, without considering the rest of the objectives. Similarly to the ideal point, finding it involves solving m single objective optimization problems, which is not generally feasible. This time, we do not have a clear way to approximate this point using the final approximation set, as the solutions it provides are optimized in terms of minimization. Various methods for approximating the nadir point were tested, but in the end the decision was made to use a method that introduces a relatively low trade-off with respect to the total program execution time. This method calculates the nadir point as the point where each coordinate is equal to the maximum value for that coordinate among all individuals used to calculate it. We will denote the approximation of the nadir point as F^{N*}.

Definition 4.3.10 (Overall Pareto spread)**.** Overall Pareto spread is a binary quality metric, $OS \colon \mathcal{P}(\Omega)^2 \to [0, 1]$ such that:

$$OS(A) = \prod_{i=0}^{m} \frac{|\max_{x \in A} f_i(x) - \min_{x \in A} f_i(x)|}{|F_i^N - F_i^I|} \tag{4.12}$$

In order to use this expression, we need to know both ideal and nadir points, which are not easy to calculate as discussed before, so the metric which will be used in practice will use the prior mentioned approximations, having the following final expression:

$$OS(A) = \prod_{i=0}^{m} \frac{|\max_{x \in A} f_i(x) - \min_{x \in A} f_i(x)|}{|F_i^{N*} - F_i^{I*}|} \tag{4.13}$$

Definition 4.3.11 (Spacing)**.** Spacing is an unary quality metric, $SP \colon \mathcal{P}(\Omega) \to [0, 1]$ such that:

$$SP(A) = \sqrt{\frac{1}{|A| - 1} \sum_{i=1}^{|A|} (\overline{d} - d_i)^2} \tag{4.14}$$

where $d_i = \min_{x_j | in A, x_j \neq x_i} \{||f(x_i) - f(x_j)||_1\}$, and being $\overline{d} = \frac{\sum_{i=0}^{|A|} d_i}{|A|}$.

This measure is a uniformity measure with respect to the distribution. The larger it is, the greater the average separation distance between each individual and its nearest neighbor, which is preferable up to a point not too large, because the solutions will be more scattered, exploring a larger proportion of the objective space.

Definition 4.3.12 (Coverage)**.** Coverage is a binary quality metric, $COV \colon \mathcal{P}(\Omega)^2 \to [0, 1]$ such that:

$$COV(A, B) = \frac{|\{b \in B : \exists a \in A : a \prec b\}|}{|B|} \tag{4.15}$$

The greater the coverage by a set A of another set B, the greater the number of solutions in A that dominate those in B, making the choice of set A preferable over B. However, to make a choice in a sufficiently informed manner, we should also calculate $COV(B, A)$.

An example of 2-dimensional coverage can be seen in figure 4.4

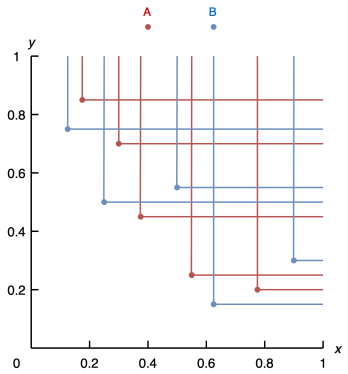


Figure 4.4.: In this image we can see solutions sets, identified by their objective values: A and B. It can be seen that $C(A,B) = \frac{2}{5}$ while $C(B,A) = \frac{3}{5}$. In set B there are points which dominates other points from the same set, and even so it covers more A solutions than what set A covers to set B.

4.3.1. Many-objectives optimization problems

We will also study a specific class of multiobjective optimization problems, known as Many-objectives optimization problems (MaOPs).

Definition 4.3.13 (Many-objectives optimization problem)**.** A Many-objective optimization problem is a multiobjective optimization problem where $m > 3$.

Due to their high dimensionality, MaOPs problems are particularly expensive to solve, and specialized techniques are required for this task. They are different from those we might use for problems with 2 or 3 objective functions. This is because various phenomena arise, such as the convergence resistance phenomenon. Therefore, we must employ algorithms specifically dedicated to addressing these challenges.

The main challenges that one has to face when trying to solve a Many-objectives optimization problem are the following [Li et al., 2015]:

- **Dominance Resistance (DR)**: When the number of dimensions increases, due to the dominance criterion we employ, many solutions that do not dominate each other start appearing. Remember that if a solution is worse in all objectives compared to another except in one, there will be no dominance. This is why many conventional algorithms, such as NSGA-II, struggle to discern between better and worse solutions when the number of conflicting objectives increases [Purshouse and Fleming, 2007, Fonseca and Fleming, 1998, Corne and Knowles, 2007].
- **Population size**: Related to the above, due to the large number of solutions that cease to dominate each other, there will presumably be a growing range of values in the Pareto-optimal set. If we want to fully describe it, the population size will have to be increased exponentially with respect to the number of objectives. This leads to a considerable decrease in the computational efficiency of the methods that solve them.

- **Visualization**: Data visualization cannot be done directly, as it could for a lower-dimensional space. Therefore, special techniques are required, such as projection to different dimensions or dimensionality reduction, as well as examining other types of functions in the objective space of the found solutions. However, when applying these visualization techniques, we are inherently losing information, and to reconstruct the original information, we should use a large number of studied and specific representations, which is not usually the best option. Hence, a compromise needs to be established.

The algorithms currently representing the state-of-the-art in optimization problem-solving techniques are Many Objectives Evolutionary Algorithms (MaOEAs). These genetic optimization algorithms, initially based on the NSGA-II algorithm, employ various techniques to better handle the abundance of objectives to optimize. There are many MaOEAs based on a multitude of concepts and utilizing various techniques. In fact, they can be classified into different families [Li et al., 2015].

4.4. NSGA-II Algorithm

The NSGA-II algorithm, or Non-Dominated Sorting Genetic Algorithm II, will be our proposal for a practical, efficient multiobjective optimization algorithm with a genetic approach. This algorithm emerges as an improvement over the original NSGA and is considered a basic and highly versatile model for multiobjective optimization problems. Its main differences from NSGA are that it uses a crowding operator instead of niches, it compares the parent population with the offspring population, and it is computationally more efficient. A theoretical overview of how the algorithm works can be seen in Figure 4.5.

The main features of this algorithm are the following:

- **Random seed**: It uses a seed for random calculations.
- **Initial population**: Creates an initial population of solutions, initially random. This initial population in our case will contain individuals with unrestricted extension constraints, so that the largest possible individuals are generated at the beginning, allowing for a more responsible comparison of individuals in this context.
- **Offspring generation**: Generates offspring population using a classic genetic approach (selection, crossover, and mutation).
- **Non-dominance**: Once all offspring are combined with the original population, we sort them according to the non-dominance criterion into as many level fronts as necessary.
- **Selection**: Having sorted them this way, we will select all individuals from the best fronts available until we reach a quantity of individuals such that if we added the next complete front, we would exceed the population size. For this last front, we order the individuals using a crowding distance function, which indicates the elements that are more dispersed with respect to the other elements in that front, and we take the ones that have performed the best until we reach the population limit. This way, we aim to pick individuals from less explored areas in this last front and promote the exploration of the method.

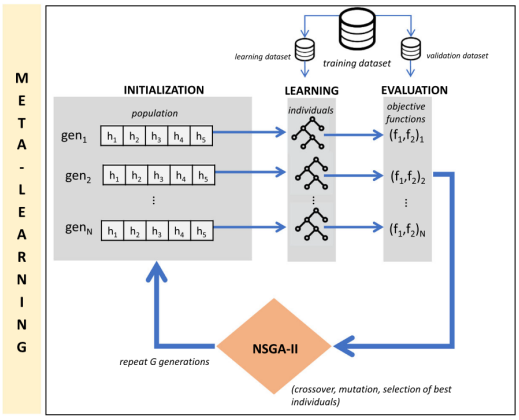


Figure 4.5.: Diagram illustrating the flow in the execution of our NSGA-II meta-learning algorithm. The initial population is randomly generated during initialization. Subsequently, a model is trained for each gene in the population with its corresponding hyperparameter combination, and their objectives are evaluated with the validation set. After that, the algorithm generates offspring, which are also trained and evaluated. Finally, the algorithm selects the best individuals among them using a level-wise front division and a crowding function.

The implemented algorithm is based on the one available in a public Github repository, and it can be found at the following link: https://github.com/baopng/NSGA-II. The pseudocode of the implementation is shown in Algorithm 1:

Crossover is performed according to the probability p_c. If it is applied, two parents (p, p') will be selected to generate offspring of two children (o, o'). For each hyperparameter h_i, the corresponding one for each of the two children is calculated as follows:

$$h_{i,o} = \frac{h_{i,p} + h_{i,p'}}{2} + \beta \frac{|h_{i,p} - h_{i,p'}|}{2}, h_{i,o'} = \frac{h_{i,p} + h_{i,p'}}{2} - \beta \frac{|h_{i,p} - h_{i,p'}|}{2} \tag{4.16}$$

where $\beta \sim \mathcal{U}(0,1)$. Therefore, each hyperparameter of each child is calculated as an average of the values of the parents, plus a random factor, which models how different the values of the parents were. In this way, the smaller the difference, the less variability in the new hyperparameters.

With respect to mutation, the probability of it occurring for each generated individual is p_m. When it occurs, only one hyperparameter of the individual will be mutated, which may result in a detriment to diversity as the dimension of the search space increases, but it is an acceptable trade-off to improve the efficiency of the algorithm. Once the gene to be mutated is specified, two random values are generated, with $u \sim \mathcal{U}(0,1)$, and δ being the value given by:

$$\delta = \begin{cases} -1 + 2u'^{\frac{1}{1+\gamma}} & \text{if } u' \leq \frac{1}{2}. \\ 1 - 2(1-u')^{\frac{1}{1+\gamma}} & \text{if } u' > \frac{1}{2}. \end{cases} \text{ where } u' \sim \mathcal{U}(0,1) \tag{4.17}$$

Algorithm 1: Meta-learning algorithm based on NSGA-II.

Input: Objective function $(f_1, \ldots, f_n)$, hyperparameter range $([\min(h_i), \max(h_i)], \forall i \in \{1, \ldots, m\}$, protected attribute name (A).
Data: Complete dataset (D)
Parameters: Number of generations (n_g), population size (n_i), mutation probability (p_m), crossover probability (p_c), crossover parameter (γ), random seed (s)
Output: Set containing predictors whose validation functions is the best approximation set found.

Set the random seed s.
Divide dataset D into training, validation and test sets (D_l, D_v, D_t).
Create initial population (P_0), containing n_i individuals.
Train each individual using the training dataset D_l.
Calculate objectives using validation dataset D_v.
Rank individuals with respect to Pareto dominance, dividing them into ranks.
Calculate crowding distance within each set.
for *j in range* $[1, n_g)$ **do**
    Create next population P_j using elitist selection, crossover (with probability p_c), and mutations (with probability p_m) over P_{j-1}.
    Train each individual using the training dataset D_l.
    Calculate objectives using validation dataset D_v.
    Rank individuals with respect to Pareto dominance, dividing them into ranks.
    Calculate crowding distance within each set.
    Remove individuals from P_j until only n_i remain.
return *Non dominated individuals from* $P_{n_g} - 1$

Once δ is set, mutation is done as follows:

$$h_{i,o} = \begin{cases} h_{i,o} + \delta(h_{i,o} - \min(h_i)) & \text{if } u' \leq \frac{1}{2}. \\ h_{i,o} + \delta(\max(h_i) - h_{i,o}) & \text{if } u' > \frac{1}{2}. \end{cases} \text{ where } u' \sim \mathcal{U}(0,1) \tag{4.18}$$

The time complexity of the basic NSGA-II algorithm is $\mathcal{O}(nn_i^2 n_g)$ (where n is the number of objectives, n_i is the population size, and n_g is the number of generations). This is because, to select individuals in each front, we need to compare each individual with all others, objective by objective. Therefore, the time complexity to compare an individual with all others in one generation would be of the order $\mathcal{O}(nn_i)$ and since this needs to be repeated for all individuals, the total time complexity would be of the order $\mathcal{O}(nn_i^2)$. Taking into account all generations, we arrive at the order $\mathcal{O}(nn_i^2 n_g)$ [Deb et al., 2002].However, for the basic case of NSGA-II, the evaluation of the objective function for an individual is done in $\mathcal{O}(1)$, while in our case, it depends on the time complexity of our training algorithm.

Therefore, now our algorithm will have a time complexity of the order $\mathcal{O}(n_g(nn_i^2 + n_i a(n, n_i)))$, where $a(n, n_i)$ is the time complexity of the learning algorithm. If the learning algorithm has a complexity greater than nn_i, then the time complexity will be dominated by the second term, which is highly likely to happen. In our case, for CART algorithm, this dominance happens.

This algorithm is basic in classical multiobjective problems, but in many objective problems (MaOPs), it does not yield such good solutions [Knowles and Corne, 2007], and it is preferable to use other algorithms designed to address these aspects.

5. Machine Learning and Decision Trees class of functions

In this chapter, a brief introduction to machine learning concepts will be provided. Properties of decision trees that will be used later will be described, as well as the NSGA-II algorithm.

5.1. Machine learning, context and concepts

Since our multiobjective optimization methods will have classification functions as individuals, we must predefine the class of functions that we will attempt to approximate the real function f that explains the phenomenon of the problem. This ensures that the classification functions handled by our method will belong to the predefined classes.

We will introduce standard notation in the context of machine learning and explore some of the key concepts that will enable us to discuss the chosen classes of functions, their properties, and the procedure for finding good classifiers [Abu-Mostafa et al., 2012]. In our case, we have a variable to predict, denoted as Y, based on a set of attributes X, including our protected attribute. The prediction function $f\colon X \to Y$ is a function that, for each element represented by the set of attributes in X, assigns the associated specific value of the variable to be predicted Y. As mentioned earlier, dealing with binary classification problems means that Y can only take on the values 0 or 1. The function f is unknown, so our goal is to find the best possible approximation to it based on the data we have. We will seek this approximation h among the functions of a manageable class denoted as $\mathcal{H}$. This class of functions may not necessarily guarantee that it contains the true function f, but it is necessary to use it to make the search process manageable.

Therefore, our goal is to find a function $\hat{h} \in \mathcal{H}$ such that we can approximate the function f as closely as possible. To achieve this, we need to define an error function that indicates the quality of our approximation h based on the errors it makes. Since we are in a classification context, the error we will use to measure the quality of our model is known as classification error. However, for learning the function h and evaluating its error, we only have a sample from the total population, denoted as $\mathcal{D}$. The error of the function h can be calculated within the sample $\mathcal{D}$ (E_{in}), although what we aim to minimize is the error for the entire population (E_{out}). We need to find methods that ensure we can bound the error E_{out} based on the error E_{in}. In this way, we must find:

$$\hat{h} \in \mathcal{H} : E_{out}(\hat{h}) \leq E_{out}(h), \forall h \in \mathcal{H} \tag{5.1}$$

Where the error we will use is the classification error. Its expression for E_{in}, calculable for a specific sample $\mathcal{D}$, and E_{out} will be as follows:

$$\begin{aligned} E_{in}(h) &= \frac{1}{N}\sum_{i=0}^{N}[y_i \neq h(x_i)] \\ E_{out}(h) &= P_x[h(x) \neq x] \end{aligned} \tag{5.2}$$

To achieve this, we will introduce a series of properties and theory that will allow us to approximate this better function based E_{in}.

Definition 5.1.1 (Hoeffding inequality for finite $\mathcal{H}$)**.** Given a binary classification problem, where we use classification error and a finite $\mathcal{H}$, with a sample $\mathcal{D}$, then $\forall \epsilon \in (0, 1]$:

$$P_{h \in \mathcal{H}}[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2|\mathcal{H}|e^{-2\epsilon^2 N} \tag{5.3}$$

where ϵ is a constant fixed by the user, and the value of N is the amount of training instance we have.

This bound is not excessively tight, but it provides us with a way to control the value of E_{out} in terms of E_{in} with a certain probability. This inequality can be expressed equivalently as:

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{1}{2N} \log \left(\frac{2|\mathcal{H}|}{\delta} \right)}, \text{ with probability } 1 - \delta. \tag{5.4}$$

In the case where the function class $\mathcal{H}$ to be used is not finite, we cannot apply this inequality. To remedy this, we can take two different approaches:

- **Generalized Vapnik-Chervonenkis theory**: Use the generalization theory of Vapnik-Chervonenkis. A bound similar to that of the Hoeffding inequality can be established using the Vapnik-Chervonenkis dimension. However, there are some classes for which this dimension is infinite, and other methods must be employed, such as algorithms based on Structural Risk Minimization.
- **Discrete class of functions**: Attempt to discretize the class of functions, leveraging discrete numerical representations. This is useful in a practical context, as the representation of a real number by a computer is discrete through a sequence of bits. However, in general, these methods often result in worse bounds compared to the previous approach.

Let us briefly introduce the theory of Vapnik-Chervonenkis.

Definition 5.1.2 (Growth function)**.** Let $(x_1, \ldots, x_n)$ be a sample of size n, and let $\mathcal{H}$ be a class of functions. Then, the growth function associated with that class of functions $\mathcal{H}$ is a function that counts how many different binary classifiers can generate a function from $\mathcal{H}$:

$$m_{\mathcal{H}}(n) = \max_{x_1, \ldots x_n} |\mathcal{H}(x_1, \ldots, x_n)| \tag{5.5}$$

It is clear that $m_{\mathcal{H}} \leq 2^n, \forall \mathcal{H}$. Moreover, the fact that $\exists n \in \mathbb{N} : m_{\mathcal{H}}(n) < 2^n$ will suppose a big role in error bounding.

Definition 5.1.3 (Break point)**.** A break point for a class of functions $\mathcal{H}$ is a value $k \in \mathbb{N}$: $m_{\mathcal{H}}(k) < 2^k$.

As an example of function classes $\mathcal{H}$ that have breakpoints, we can mention, for instance, the class of linear functions ($k = 4$), the class of positive rays on a line ($k = 2$), or intervals on a line ($k = 3$). A class that, for example, does not have breakpoints could be the class of convex polygons in $\mathbb{R}^2$.

Lemma 5.1.0.1 (Break point extension)**.** *If a class of functions $\mathcal{H}$ has a break point k, then $\forall k' > k, m_{\mathcal{H}}(k') < 2^{k'}$.*

Proof. If k is a breakpoint, it means that there does not exist any set of size k that we can binary classify in all possible ways using functions from the class $\mathcal{H}$. By contradiction, if there were a point $k' > k$ such that $m_{\mathcal{H}}(k') < 2^{k'}$, it would imply that there exists a sample of size k' to which we can assign a binary classification in all possible ways using functions from our class $\mathcal{H}$. Taking a subset of this sample of size k, we can observe that we would be able to do the same for this subset as well, contradicting our hypothesis. □

Lemma 5.1.0.2 (Main result of Vapnik-Chervonenkis)**.** *If k is a break point of class of functions $\mathcal{H}$, then $m_{\mathcal{H}}(N) \leq \sum_{i=0}^{k-1} \binom{N}{i}, \forall N \in \mathbb{N}$, thus $m_{\mathcal{H}} \in \mathcal{O}(N^{k-1})$.*

The proof can be consulted in [Vapnik, 1971]. Therefore, in the case where a function class has a breakpoint, it necessarily implies that its growth function has polynomial growth. This fact allows us to establish a bound in the following manner.

Definition 5.1.4 (Vapnik-Chervonenkis dimension)**.** Vapnik-Chervonenkis dimension, denoted as $d_{vc}(\mathcal{H})$, or simply d_{vc}, is the greatest value of N such that $m_{\mathcal{H}} = 2^N$. In case it does not exist, then $d_{vc}(\mathcal{H}) = \infty$.

Lemma 5.1.0.3 (Vapnik-Chervonenkis bound)**.** *Having a class of functions $\langle$ with $d_{vc} < \infty$, then we can establish the following bound for E_out given E_in:*

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{8}{N} \log\left(\frac{4((2N)^{d_{vc}} + 1}{\delta}\right)}, \text{ at least with probability } 1 - \delta. \tag{5.6}$$

In the case where the value of the Vapnik-Chervonenkis dimension is infinite, we cannot establish a bound using this theory. We can resort to the discretization trick, but the bound obtained may be quite imprecise. Therefore, it is often better to make bounds using other procedures.

5.1.1. E_{out} approximations using test sets

The methods discussed earlier provide bounds on the value of E_{out}, but these bounds are often not of high quality. A more effective way to calculate the value of E_{out} involves using a test set to evaluate the quality of our model. This dataset represents a new sample drawn from the population, independently and identically distributed with respect to the one previously drawn for training. Typically, and as will be the case in our experimentation, we do not have two distinct datasets for training and testing. Therefore, we will split the dataset into two, using one part for training and the other for testing. The error obtained on the test set will be denoted as E_{test}.

Using a test set provides a significant advantage, as now we can apply the Hoeffding inequality where the class $\mathcal{H}$ consists of a single function, which is the one obtained after training. With this, the following expression is satisfied:

$$P[|E_{test}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N_t} \tag{5.7}$$

where in this case, N_t is the number of instances in the test set. This represents a much more accurate bound than the one used when considering only the training set. That is the reason why using a test set to evaluate our models will be practically indispensable. This approximation will be employed to evaluate the models for the experimentation conducted, which will be discussed in section 9.5.

Keep in mind that allocating more data to training makes our test approximation less precise, and dedicating more data to testing means having fewer data for training, potentially resulting in poorer classifiers. It is important to note that the test dataset will be completely separate and only used to evaluate the models as the final step in our procedures. Processes such as data normalization, if performed, should be done separately for both sets to avoid the phenomenon known as Data Snooping, where information from the test set is directly or indirectly used for training, making the procedure theoretically invalid.

5.1.2. Bias-variance tradeoff

The error made by our model can be decomposed into the sum of three distinct components, namely:

- **Bias**: One component is called bias, which indicates how far the average prediction made over the function class is from the real function to be approximated.
- **Variance**: Another component is called the variance component, which indicates the internal variability of the function class with respect to the class mean, and it depends on the size of the taken sample.
- **Random noise**: A final component indicates the variance of the noise, a component that cannot be controlled.

We can express this tradeoff as: $\mathbb{E}_{\mathcal{D}}[E_{out}(h^{(\mathbb{D})})] = \text{bias} + \text{variance} + \sigma^2$.

This concept of bias-variance tradeoff was initially developed for regression, but it has been shown to hold true for classification contexts as well, as demonstrated in [Domingos, 2000].

5.1.3. Overfitting and regularization

A common phenomenon that can occur during the training of our models is that known as overfitting. Overfitting happens when models with lower values of E_{in} result in a higher error E_{out}. This phenomenon may be due to stochastic errors, where there might be noisy examples, and to adapt to them, the classifier needs significant modifications that hinder its generalization to examples outside the sample. It can also be attributed to deterministic errors, as the trained model may not accurately represent the real function f.

To address this issue, regularization is proposed as a technique that limits the function class through soft constraints, making it challenging to train excessively complex models that are prone to overfitting. Common regularization techniques include weight decay for linear models or parameter growth limitations for decision trees, among others.

5.1.4. Validation sets

The validation set involves a process similar to that carried out using test sets. In this case, the training set is divided into two subsets: the actual training set and the validation set. Models are trained with the pure training set, and once trained, their goodness is assessed based on their performance when applied to the data in the validation set. This allows us to evaluate the behavior of the models against independently and identically distributed data to those used in training, aiming for a better estimation of out-of-sample error. This procedure is done with the goal of selecting a better hypothesis or set of hypotheses during the training of the models.

It is possible that our learning models have a series of parameters (known as hyperparameters) that we want to estimate to build the best possible model. In this case, we will need to make estimates about the values for these hyperparameters that provide the best results. In our context, since we will use optimization algorithms where the decision space consists of a set of possible values for hyperparameters, and to evaluate the objectives, we will need to check different measures based on the already trained classifier, the need for a validation set becomes evident.

Currently, sophisticated validation techniques such as Leave-One-Out or K-fold cross-validation are employed. However, for our experimentation, we will use a simpler approach, which involves setting aside a subset of the data explicitly dedicated to validation, similar to what was done with the test set. This chosen validation technique will be explained in section 9.5.

5.2. Decision Trees as a class of functions

In this case, and given the context in which we find ourselves, each individual managed by our genetic algorithms represents a classification function. We need to adjust this function using a machine learning model and subsequently evaluate the quality of its predictions. To demonstrate the potential that can be achieved through the proposed procedure and to ensure that the total execution time of our algorithm is not excessively high due to the significant time invested in training each individual, we will use function classes $\mathcal{H}$ that are relatively simple. These classes should have computationally inexpensive training methods yet possess good properties. We will use decision trees which basic properties will now be studied.

5.2.1. Decision Trees

Decision Tree classifiers have been broadly used in the context of machine learning. It is a basic model to be studied in any basic ML course and can be used within the context of a great amount of problems, as it is a simple, yet really deep model, which can greatly adapt to the disposed training data. This is mainly due to their ability to find the combination of features and values which best split the data into subgroups of similar characteristics. Furthermore, its learning algorithm is relatively simple and intuitive, which increases its interest.

Decision trees are a class of functions that partition the space into different regions, whose boundaries are parallel to the coordinate axes of each variable if they do not extend to infinity.

Interpreting all our variables as real variables without loss of generality, we can represent the class of classification functions described by decision trees as follows:

$$\mathcal{H} = \left\{ \sum_{i=0}^{N} \alpha_i \mathbb{1}_{\mathcal{X}_i}(x) : N \in \mathbb{N}, \alpha_i \in \mathbb{R}, \mathcal{X}_i \text{ meet the following properties} \right\} \tag{5.8}$$

- **Half-closed intervals**: Each $\mathcal{X}_i$ is a cartesian product of half-closed intervals, not necessarily bounded.
- **Partition**: $\{\mathcal{X}_i : i \in \{1, \ldots, N\}\}$ conform a partition of X, which means:
  - $\cup_{i \in \{1,\ldots,N\}} \mathcal{X}_i = \mathbb{R}^N$
  - $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset, \forall i, j \in \{1, \ldots, N\}$

To perform this partition, trees are structured as a set of hierarchical rules, effectively forming a tree structure, where each non-terminal node represents a rule, and each edge represents a possible outcome for that rule. The rules used are inequality comparison rules with respect to a single variable, ensuring that the structure of decision boundaries is parallel to the axes. Thus, for each new example to classify, decision rules are applied consecutively based on the results of previous rule evaluations. The leaf nodes represent the class in which to classify the given example.

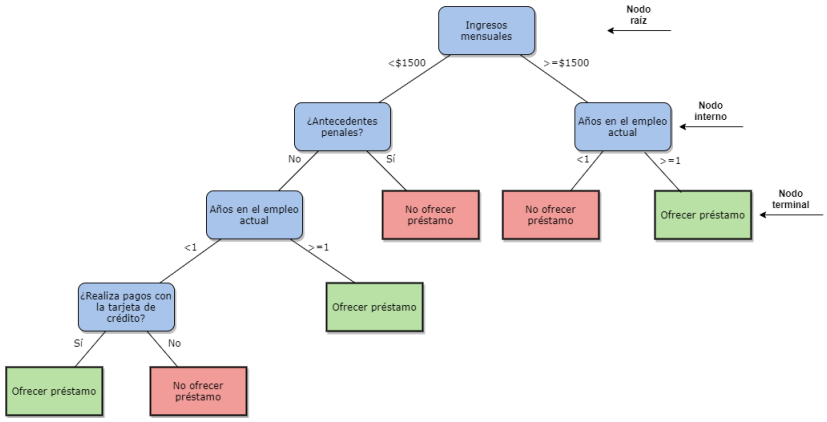


Figure 5.1.: Example of a binary decision tree. The terminal nodes represent the classification made, the non-terminal nodes represent the decision rules to apply, and each edge is directed from nodes of lower to higher depth and represents a possible response to the decision rule. This tree is binary because each node has at most (in this case, exactly) two child nodes.

5.2.2. CART learning algorithm

The learning algorithm to fit the best function from the mentioned class $\mathcal{H}_{DT}$ to our sample is called CART (Classification and Regression Trees). The pseudocode of the algorithm can be seen in Algorithm 2, and later, clarifications will be provided on specific aspects of its operation.

Algorithm 2: Basic CART algorithm for decision trees.

Input: Impurity criterion g.Stop criterion.
Data: Training dataset (D_t).
Output: Trained decision tree

1 Initialize the tree with a single node that contains all the training data D_t.
2 **if** *All instances that fall onto the current node have the same class, or some stopping criterion is met.* **then**
3 Convert the current node into a leaf node that assigns the class to all instances, and we finish.
4 **else**
5 Calculate the attribute and feature that best separates the data falling onto the current node based on the node's impurity measure.
6 Assign to this node the associated decision rule.
7 A cost-complexity pruning process is carried out for the possible child nodes: $g(n) + \alpha|N|$ where α is a free parameter subject to optimization, and $|N|$ is the number of nodes created up to that point. We create two child nodes if they have not been pruned, and recursively launch this algorithm, using only the data filtered by the decision rule for each child.
8 **return** *Decision tree regressor with parameters adjusted*

To discuss the efficiency of the algorithm, we need to introduce the $\mathcal{O}$ notation. Let us provide a brief definition of notation $\mathcal{O}$ and mention some of its most interesting properties.

Definition 5.2.1 ($\mathcal{O}$ notation)**.** The class of functions $\mathcal{O}(g)$, being g a function, as the following set:

$$\mathcal{O}(g) = \{f : \exists x_0, c > 0 : 0 \leq |f(x)| \leq c|g(x)|, \forall x \geq x_0 > 0\} \tag{5.9}$$

Despite $\mathcal{O}(g)$ being a set, it is denoted as $f = \mathcal{O}(g)$, instead of $f \in \mathcal{O}(g)$.

A function $f = \mathcal{O}(g)$ if it growth is bounded by function g (except for a constant). Let us now write some simple properties which directly derive from the definition of (O) notation:

- **Transitivity**: If $f_1 = \mathcal{O}(g_1)$ and $g_1 = \mathcal{O}(g_2)$, then $f_1 = \mathcal{O}(g_2)$.
- **Product of functions**: If $f_1 = \mathcal{O}(g_1)$ and $f_2 = \mathcal{O}(g_2)$, then $f_1 f_2 = \mathcal{O}(g_1 g_2)$.
- **Product of class by a function**: $f_2\mathcal{O}(g_1) = \mathcal{O}(f_2 g_1)$.
- **Sum of functions**: If $f_1 = \mathcal{O}(g_1)$ and $f_2 = \mathcal{O}(g_2)$, then $f_1 + f_2 = \mathcal{O}(|g_1| + |g_2|)$.
- **Scalar product of a class**: If $k \neq 0$, then $\mathcal{O}(g_1) = \mathcal{O}(kg_1)$.
- **Scalar product of a function**: If $f_1 = \mathcal{O}(g_1)$, then $kf_1 = \mathcal{O}(g_1)$.
- **Class of a sum of functions of the same order**: If $f_1 = \mathcal{O}(g_1 + g_2)$ and $g_1 = \mathcal{O}(g_2)$, then $f_1 = \mathcal{O}(g_2)$.

5.2.3. Impurity metrics

The choice of the best attribute and node is made through a greedy heuristic procedure (best-first). All possible space partitions X into regions that effectively separate the data in

different ways using the value of only one variable at a time will be evaluated. The chosen partition is the one that yields the best result with respect to the node's impurity function. Wit m being the number of attributes and n the number of instances, the time complexity of the algorithm is $\mathcal{O}(mn\log(n))$, as the evaluation of a possible partition can be done in logarithmic time with respect to the number of instances [Sani et al., 2018].

There are several options to measure the impurity of a partition made by a node. The two most common ones that we will consider are:

- **Gini index**: For leaf nodes, its value is $1 - \sum_{i=0}^{n} p_i^2$, where n is the number of possible clases, and p_i is the proportion of elements from class i which fall into that node. For the rest of nodes its value is calculated using the value $1 - \sum_{i=0}^{n} p_i^2$ for each one of its children nodes (being them leaves or not), weighted using the amount of samples which fall into each children node. Possible values for this metric are in the range $[0, 0.5]$.
- **Entropy**: Entropy calculation is done following the same pattern as for Gini index, but using the expression $-\sum_{i=0}^{n} -p_i \log_2(p_i)$ instead. Possible values for this metric are in the range $[0, 1]$.

For both metrics, the closer they are to 0, the worse will be their split. We can see a comparison between both Gini and entropy functions in figure 5.2

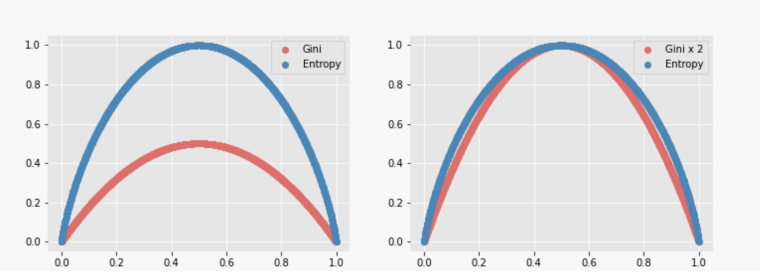


Figure 5.2.: Relation between Gini and entropy impurity functions. Both functions are very close, and could lead to little diferences during the learning process.

5.2.4. Advantages and Disadvantages

As advantages we can highlight the following:

- **Independent to scale**: The method is completely invariant to the scale of the data.
- **Data types**: Decision trees can handle both categorical and real-valued data.
- **Explainability**: Decision trees have high explainability, and it is possible to understand the process of classifying an individual, as well as the most important variables for separating the data.
- **Representation**: They have a direct and interpretable representation through their associated set of decision rules.

- **Prediction**: Once learned, prediction using a decision tree is quick as it involves applying the associated set of rules.
- **Missing values**: Decision trees can easily handle missing values, as they do splits based on the available data.

On the contrary, as disadvantages we can highlight the following:

- **Overfitting**: They tend to overfit the sample if their growth is not controlled using hyperparameters.
- **Vapnik-Chervonenkis dimension**: $d_{vc} = \infty$
- **Variance**: The decision tree class has high variance. Given small variations in the training data, the resulting classification tree can be very different. To address this issue, the Random Forest model emerged, although it sacrifices the explainability of the model itself.

5.2.5. Additional considerations: Regularization

Decision trees constitute a function class with $d_{vc} = \infty$. This is evident because a decision tree could grow as much as necessary, generating as many rules as needed to perfectly classify any sample. This would result in each individual falling into a different node, providing a tree with a total of n nodes, where n is the number of instances in the sample. The way to introduce regularization in the model involves limiting the growth of the tree by restricting the number of leaf nodes it can have or requiring a certain number of individuals to fall into a node before it can split.

Part III.

Methodology

In this part, the theory and complete descriptions behind the developed algorithms will be explained to deeply understand them.

6. First algorithm: Fair Decision Tree

In this chapter, the underlying theory behind our first algorithm, Fair Decision Tree (FDT), will now be discussed. As previously explained, this algorithm will consist of modifying the impurity criterion in the learning process of a decision tree to incorporate a fairness criterion.

6.1. Description of the algorithm

A traditional decision tree learning algorithm relies on an impurity calculation criterion, with the most commonly used ones being the Gini and entropy criteria. Both are closely related, as observed in Figure 5.2. However, these criteria do not consider fairness in the learning process. It would be beneficial to incorporate a fairness criterion to influence the learning process, aiming to infer fairer trees. Therefore, the main goal of this first algorithm is to incorporate and consider a fairness criterion during the learning process.

The incorporation of a fairness criterion during the decision tree learning process will be achieved by modifying the node impurity calculation criterion. This modification will involve a convex combination of a traditional impurity criterion with a fairness measure of our choice. Therefore, our impurity criterion will take the form:

$$(1-\lambda) * \text{traditional criterion} + \lambda * (-\text{fairness criterion}) \tag{6.1}$$

where λ is a parameter that controls the relative importance assigned to our fairness criterion. The fairness criterion needs to be negative, as FALTA This parameter specification is crucial, and even a small change in it could lead to learning very different trees structurally. The optimal value for this parameter can heavily be influenced by the structure of each given problem, so a parameter optimization procedure can be crucial to find it. It is clear that if $\lambda = 0$, then no fairness metric is taken into account during the learning process, while if $\lambda = 1$, no Gini or entropy criterion will be considered during training (which usually is far from optimal).

In order to perform this calculation, we need to determine the value of the chosen fairness criterion at each node of the tree. During the training phase, new nodes of the decision tree will be generated, and a given set of instances from the training set will fall into them. Traditionally, for the Gini and entropy impurity criteria, only the number of instances from each class falling into that node is taken into account. However, when considering a fairness criterion, we cannot solely divide those instances by class; instead, we have to additionally consider the value of the protected attribute of each instance, as well as other relevant metrics.

Calculating group fairness metrics involves using the confusion matrix, dividing the instances by each demographic group. For this reason, these confusion matrices have to be considered to calculate any group fairness metric. However, to calculate a confusion matrix, predictions need to be made. The question arises: how do we calculate those predictions at any interme-

diate node, thereby determining the number of correctly and incorrectly classified instances from each group.

Using the traditional decision tree classification criterion could be considered, but this hinders the calculation of fairness metrics. For this reason, other methods for calculating these confusion matrices will be proposed. A classification of these methods will now be presented. They are divided using two dimensions. First, we will consider whether the prediction is constant (always predicting the same class) or probabilistic (the prediction is not constant, and there is a probability of predicting one class or another). Second, we will determine if the prediction is made dependent on the protected attribute or independent. This classification can be seen in Table 6.1.

Table 6.1.: Classification of criteria for calculating predictions, and thus confusion matrices in any given node of a decision tree.

Are predictions probabilistic?	Considers fairness in node split criterion?	
	No	Yes
No	Constant prediction	Fair constant prediction
Yes	Probabilistic prediction	Fair probabilistic prediction

All the classes appearing in Table 6.1 will be now explained:

- **Constant prediction**: This criterion is employed by traditional decision trees after being trained to assign a class to an instance, predicting the same class for all instances falling into a certain leaf. This poses a significant problem for our procedure because, in such cases, many fairness measures will yield degenerate values, and some may not even be calculable, thus assigning degenerate values as well. This limitation prevents us from considering gradual changes in fairness and implies an additional computational cost. Theoretically, the prediction in each leaf in this case would be $P[p = 0|A = 0] = P[p = 0|A = 1] = c \in 0, 1$, $P[p = 1|A = 0] = P[p = 1|A = 1] = 1 - c$. The estimation of the value c is typically made using maximum likelihood estimation. Let P be the number of training instances falling into that node belonging to the positive class, and N be the number from the negative class. Then, $c = 0$ if $P > N$ and $c = 1$ if $P < N$. If $P = N$, any criterion could be used.

- **Fair constant prediction**: In this case, it is more appropriate for the decision-making process to account for the value of the protected attribute since fairness measures are theoretically calculated using probabilities conditioned on this attribute's value. However, integrating this consideration does not resolve the issues of degeneration or the prior calculation impossibility. In each leaf, we would have $P[p = 0|A = 0] = c_0$, $P[p = 0|A = 1] = c_1$, with $c_0, c_1 \in 0, 1$, $P[p = 1|A = 0] = 1 - c_0$, and $P[p = 1|A = 1] = 1 - c_1$. Extending the notation from the previous class, considering P_0 as the number of training instances belonging to positive class and unprivileged group (P_1, N_0 and N_1 analogously), then $c_i = 0$ if $P_i > N_i$, and $c_i = 1$ if $P_i < N_i$, $i \in \{0, 1\}$.

- **Probabilistic prediction**: In this scenario, we encounter a situation analogous to CDT, with the distinction being that probabilities may assume values other than 0 or 1.

In each leaf, we would have $P[p = 0|A = 0] = P[p = 0|A = 1] = c \in [0, 1]$, and $P[p = 1|A = 0] = P[p = 1|A = 1] = 1 - c$. In this case, for the estimation of parameter c using maximum likelihood estimation we can be a little more precise. The value of c will then be $c = \frac{N}{P+N}$.

- **Fair probabilistic prediction**: This prediction approach is arguably the most theoretically accurate among those discussed because the group fairness criteria to be utilized always involve conditioning probabilities on the protected attribute. Hence, it is logical to calculate prediction probabilities by conditioning them on the values of the protected attribute, even though the final probability in the leaves is computed without considering these values. Consequently, we can compute these conditioned probabilities using the examples falling into that node. For instance, the probability of predicting an instance as positive, given that the value of its protected attribute is 0, would be $\frac{P_0}{P_0+N_0}$, and similarly, for a value of 1 for the protected attribute, it would be $\frac{P_1}{P_1+N_1}$. However, this method introduces a drawback when calculating false positives and false negatives, as both will have the same value $\frac{P_i N_i}{P_i+N_i}$ for each i. Consequently, certain measures may yield identical values. Notably, the disparities between True Positive Rate (TPR) and Positive Predictive Value (PPV) will be equal, given that false positives and false negatives are equivalent. In this case, $P[p = 0|A = 0] = c_0$, $P[p = 0|A = 1] = c_1$, with $c_0, c_1 \in [0, 1]$, $P[p = 1|A = 0] = 1 - c_0$, $P[p = 1|A = 1] = 1 - c_1$.

An example of how confusion matrices could be visualized during learning at a specific node can be seen in Figure 6.1.

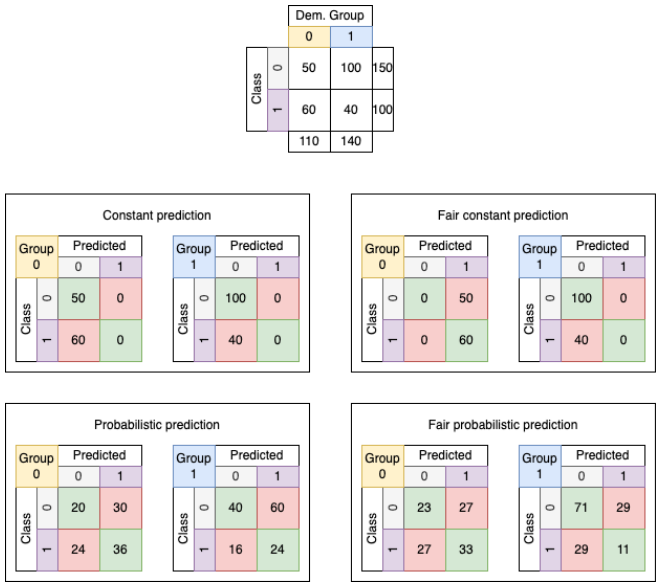


Figure 6.1.: Different criteria for calculating confusion matrices for a node, whose training instances with their respective classes and groups appear in the upper table.

For the reasons previously described, the criterion for calculating confusion matrices at each intermediate node that we will use in our implementation will be the latter one: fair probabilistic prediction.

The pseudocode of the algorithm is actually the same as Algorithm 2, with the only difference being the consideration of this new modified impurity criterion, where fair probabilistic prediction will be used for calculating confusion matrices at each node.

6.2. SWOT Analysis

In this subsection, the different strengths, weaknesses, opportunities, and threats of this algorithm will be analized. These analyses, also known as SWOT analyses for their acronym, are two-dimensional analyses commonly conducted in the business world to study the quality of a particular proposal. Before conducting the analysis, we must first specify what each of these concepts means within our context.

- **Strengths**: These are the internal strong points where the algorithm excels. They are characteristics that make it a viable option for use.
- **Weaknesses**: These are the internal weak points where the algorithm falters. They are characteristics that make one consider whether it should be used or not.
- **Opportunities**: These are external characteristics that potentially make the algorithm stronger. In this case, we will consider all improvement possibilities and potential future development lines for this algorithm.
- **Threats**: These are external characteristics that potentially make the algorithm weaker. In this case, we will consider all possible competitors that rival the algorithm and might lead to reconsidering which algorithm to use.

SWOT analyses are two-dimensional because these characteristics can be grouped into the following two dimensions:

- **Goodness**: Are these characteristics helpful (strengths and opportunities) or harmful (weaknesses and threats) in terms of achieving our goal?
- **Origin**: Are these characteristics internal to the algorithm (strengths and weaknesses) or external to it (opportunities and threats)?

Once this analysis is defined, the characteristics identified for the FDT algorithm can be examined in Figure 6.2.

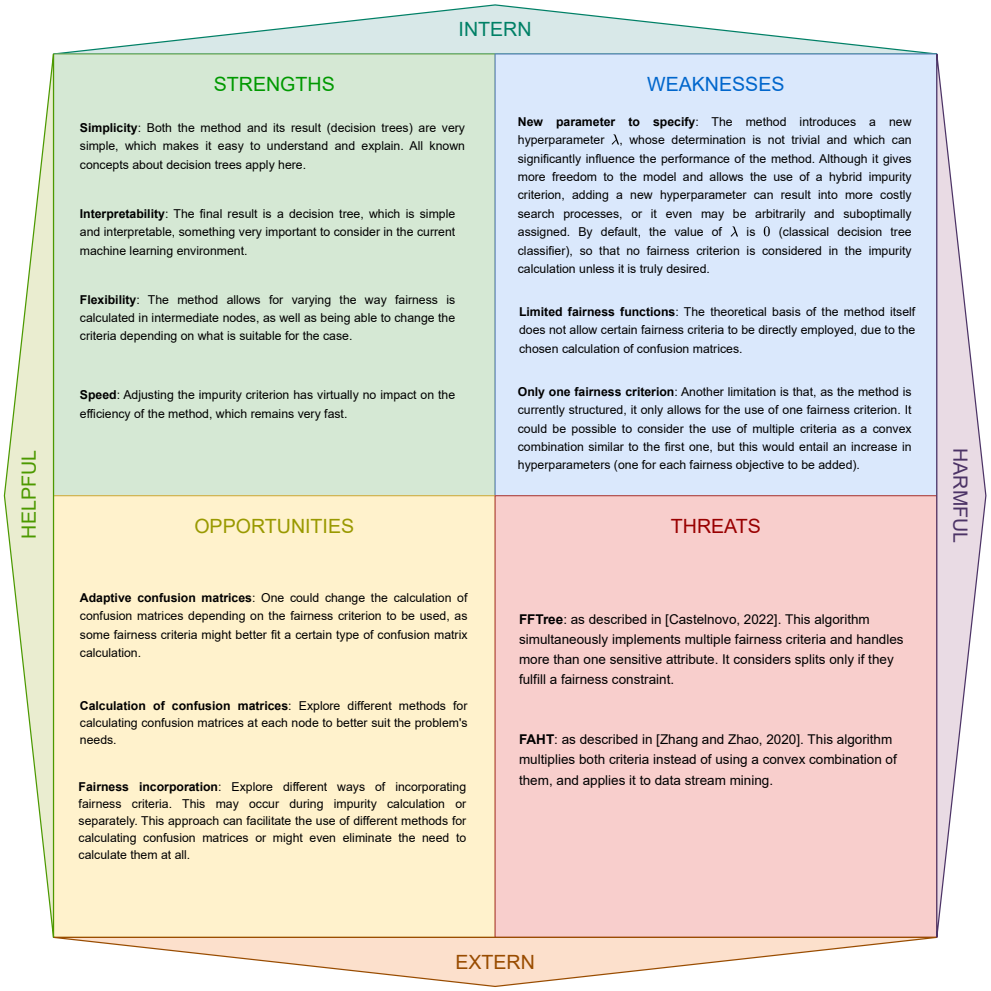


Figure 6.2.: SWOT Analysis for the FDT algorithm. The references for the threat algorithms are as follows: [Castelnovo, 2022] y [Zhang and Zhao, 2020].

7. Second algorithm: Fair Genetic Pruning

In this chapter, the second developed algorithm, named Fair Genetic Pruning (FGP), will now be discussed. This algorithm is based on genetic metaheuristics, decision trees, and pruning techniques applied to them. The basic idea is to construct the largest decision tree possible and then create a space for pruning it, from which individuals are derived as pruned versions of the base tree.

7.1. Description of the algorithm

In this section, each of the algorithm's characteristics will be described in detail in an orderly manner, those being decision space, representation of individuals, population initialization, crossover criterion, mutation criterion, distance between decision trees, and generational replacement.

7.1.1. Decision space

To define the decision space of our problem, we first need to understand what constitutes a solution for the problem we are dealing with and explore possible characterizations of it..

We start with a binary matrix tree, which serves as the foundation for our algorithm. This matrix tree is created without restrictions in terms of size. This approach leads to the construction of a decision tree that perfectly classifies our training sample, but it will likely overfit the training set, resulting in a drop in precision in any real deployment environment (as well as for validation and test sets). Fairness metrics measured on this matrix tree will obviously not be as good either, considering the fairness impossibility theorems. This tree will be rather large; in fact, it is the optimal tree with respect to the impurity criterion used, perfectly classifying the training sample. This means that any other tree using the same impurity criterion will classify our training sample worse in terms of accuracy and will be smaller in size. The key property that we will take advantage of with this tree is that it is the largest we can build with the available training samples (and it is optimized with respect to the impurity criterion).

The main goal of this algorithm is to find the subtrees of this matrix tree which have the optimal balance in terms of the considered criteria, those being presumably accuracy, fairness and/or tree size criteria. Our decision space will then be all the possible subtrees having the same root node as our matrix tree. It is important to note that the root node has to be the same, as we can build other subtrees having different root nodes, but in that case we would start the classification process in a branch of the matrix tree, in which only a subset of training samples fall, which does not make any sense. The classification process should start always the same way, so the root node has to be always the same.

It is important to note that a subtree of the matrix tree concerning the already cited requirements can also be seen as a set of prunings applied to the matrix tree. This is because

applying a pruning over a tree results in a new tree, having the same structure as the base one, but deleting all the subtree structure after a given node. All subtrees of the base tree can be constructed by applying a set of prunings over the base tree. For this reason, the decision space can also be seen as the possible trees that can be obtained after applying a set of prunings over the matrix tree as well. We will now explore this fact.

7.1.2. Representation of individuals

As we have mentioned, the algorithm will utilize a matrix tree, which will serve as the basis for constructing population individuals, being subtrees of the same sharing the same root. Additionally, all these subtrees can also be defined as sets of prunings performed on the matrix tree. Using these characteristics, an equivalence can be established between a subtree and the set of prunings that characterizes it. In this way, we will establish two ways of representing population individuals, which will be used in the algorithm depending on which one is more suitable for each task to be performed:

- **Pruning representation**: Each individual in our population can be represented as a set of prunings applied to the matrix tree. We can leverage the fact that our matrix tree is a binary tree to create a simple mathematical representation of its nodes and to define what constitutes a pruning.

  Definition 7.1.1 (Representation of nodes in a binary tree)**.** A binary tree, which is a directed graph with only one root node, can have its nodes represented by an ordered list of values $[n_1, n_2, \ldots, n_r]$, where $n_i \in \{0, 1\}$ for all $i \in \{1, \ldots, r\}$, representing a path from the root node. Starting from the root node, a value of $n_1 = 0$ means that we will travel through its left child, and a value of $n_1 = 1$ means that we will move to its right child. We continue this procedure recursively from the node where we arrive after following all the previous path indications. Finally, we reach the node given by the value n_r.

  Definition 7.1.2 (Pruning over a matrix tree)**.** A pruning of the matrix tree involves removing the children nodes and links from a particular node. The node at which the pruning occurs is specified by an ordered list of values $[n_1, n_2, \ldots, n_r]$, where $n_i \in \{0, 1\}$, $\forall i \in \{1, \ldots, r\}$. This list indicates the path to the node where pruning will be performed. Thus, we identify this operation as pruning $[n_1, n_2, \ldots, n_r]$.

  Using this definition, we can represent any individual as a set of prunings. The matrix tree would be represented by $[\,]$, while other subtrees would be represented by the prunings that have been performed. Care must be taken, as there could be redundant prunings (for example, the pruning $[0, 1]$ would be meaningless if the pruning $[0]$ has already been done), and for this reason, criteria must be established to avoid these mistakes. Fortunately, the representation is prepared to be sorted efficiently using lexicographic order. This will not only allow us to avoid redundant prunings with efficient checks but also to perform other optimizations at the code level.

  Examples of this representation can be seen in Figure 7.1. These representations are already in their minimal form (without redundant prunings) and sorted lexicographically, conditions that will always be maintained to avoid errors.

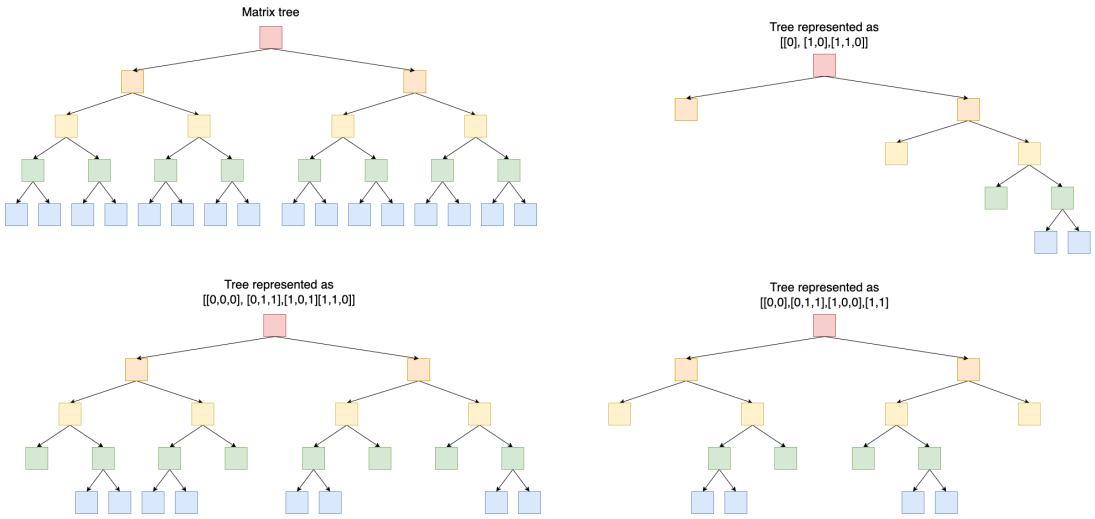


Figure 7.1.: Example of pruning representation of individuals given a matrix tree.

- **Leaves representation**: We can even leverage the representation of nodes and prunings to define the structure of our matrix tree. A key observation is that, as can be seen in Figure 7.1, the pruning representation can actually be seen as a representation in which only the new leaves of the tree appear, which did not belong to the matrix tree before. This characteristic allows us to extend the notation and represent the matrix tree using a notation that only includes the representation of its leaves. If we were to look at it from a pruning perspective, we could thus consider the matrix tree as an "infinitely large" tree, which is pruned at the leaves it ultimately ends up having.

  With this consideration, we can define another notation as seen in Figure 7.2, where each of the leaves that the considered tree ultimately has are encoded, which is why it is called leaves representation. This representation is often very redundant, as pruning representation allows us to save many elements. However, it can be useful in certain situations, such as in the mutation criterion that will be discussed later on.

7.1.3. Population initialization

To create the initial population, we need to define the matrix tree as mentioned earlier. For this task, all of our training examples need to be utilized, and the largest (unrestricted in terms of size) tree has to be trained. Once our matrix tree is established, we can begin defining our initial population in terms of it.

For our initial population, we will have a set of subtrees as defined in previous section 7.1.2 of fixed size n_i. This population size will not vary over time, so it is a parameter that has to be previously specified.

The goal of defining this initial population is to create a diverse set of individuals that will aid us in searching for good solutions as generations pass. To achieve this, we will follow the

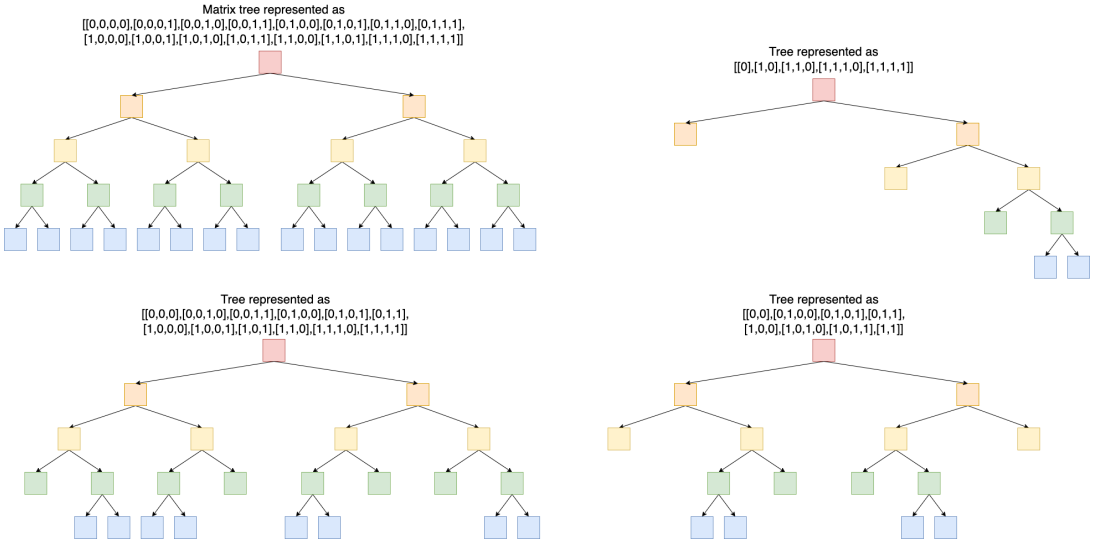


Figure 7.2.: Example of leaves representation of the same individuals and matrix tree as shown in Figure 7.1.

following criteria:

- **Original tree**: The original matrix tree (without any prunings) will be included in the initial population.
- **Pruning probabilities**: For the remaining individuals up to size n_i, we will consider the matrix tree and traverse it from the root. The probability of each node being pruned will be assigned independently of its depth, making all nodes equally likely to be pruned. Let us denote this probability of pruning a certain node as α. In this scenario, we must make a crucial observation: if an ancestor of a given node has already been selected for pruning, that node will not be able to be selected for pruning again. This implies that the probability of any deeper node must be greater than that of any shallower one.

  The probability of pruning nodes at depth 1 will be α. For nodes at depth 2, the probability cannot simply be α, because if its parent has been pruned, it cannot be selected for pruning. Therefore, the actual probability we have to assign is x, satisfying $\alpha = x(1-\alpha) \Rightarrow x = \frac{\alpha}{1-\alpha}$. We can iteratively apply this process to obtain that the probability we have to assign to a node at depth n is $\frac{\alpha}{(1-\alpha)^{n-1}}$, in order to be selected for pruning with probability α.

  However, we must note that $f(n) = \frac{\alpha}{(1-\alpha)^{n-1}}$ is monotonically increasing with respect to n, and it can exceed 1. In fact, $f(n) = 1 \Leftrightarrow \frac{\alpha}{(1-\alpha)^{n-1}} = 1 \Leftrightarrow \alpha = (1-\alpha)^{n-1} \Leftrightarrow \log(\alpha) = (n-1)\log(1-\alpha) \Leftrightarrow n = 1 + \frac{\log(\alpha)}{\log(1-\alpha)} = 1 + \log_{1-\alpha}(\alpha)$. This implications are bidirectional as $0 < \alpha < 1$.

This means that for any tree of depth n, we can assign a probability α which satisfies $1 > \frac{\alpha}{(1-\alpha)^{n-1}} \Leftrightarrow 1 > \alpha + \alpha\frac{1}{n-1}$. To numerically solve it, it is better to express it in terms of the inequality $\alpha - (1-\alpha)^{n-1} < 0$, which can be solved very efficiently.

- **Final pruning probability**: As a general criterion, given a matrix tree of depth n, our value α will be half of the solution to the equation $x - (1-x)^{n-1} = 0$.

We can see an example of this in Figure 7.3.

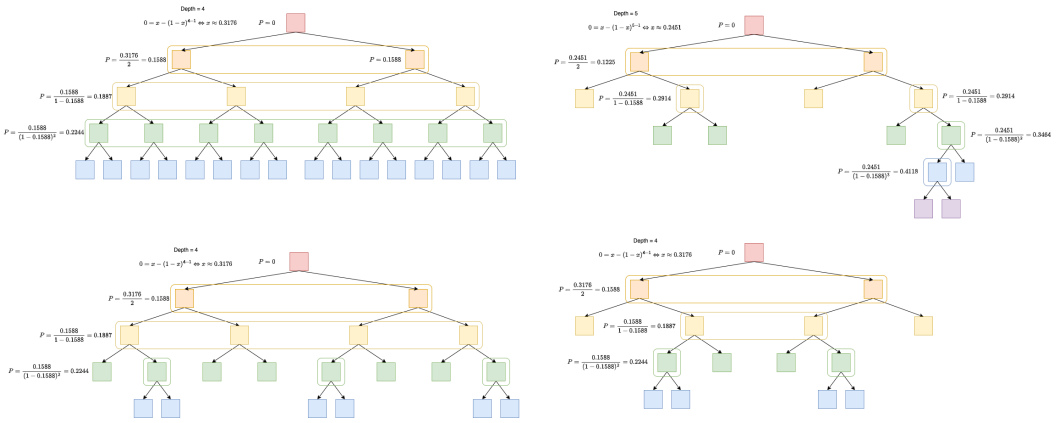


Figure 7.3.: Initial probabilities associated with the pruning of each of the nodes of the matrix trees displayed.

7.1.4. Crossover between individuals

Crossover between solutions is a crucial aspect to consider when defining a genetic evolutionary algorithm. Once we have a population of solutions, we will need to create a population of descendant solutions. To do this, we will need to specify how parent solutions are selected (selection criterion), and how the crossover between them is performed, if it is indeed carried out.

In our case, for each pair of child solutions, we will need to select two parent solutions. To select each parent individually, we will randomly select 2 individuals from the current population. We will evaluate how fit these individuals are for reproduction (in terms of the objective function) and select the better one. If neither is better than the other, one will be randomly selected. Once both parents are chosen through this procedure, they may or may not reproduce, with a probability p_c initially defined. If no crossover occurs, the two parent solutions are returned to belong to the new population of child solutions. In our case, $p_c = 1$, as elitism will be used for generational replacement, considering all parent and child solutions.

On the other hand, in the event that a crossover occurs, we must specify how it will be performed. The procedure for carrying out a crossover will be as follows:

- **Select probability**: Firstly, a value $\gamma \in [0.4, 0.6]$ will be considered. This parameter will control the assignment of prunings from the parents to the children. γ will represent

the probability of assigning a pruning from the parents to the first child, and thus the probability of assigning it to the second child will be $1 - \gamma$. This probability is randomly chosen and the value 0.5 is not used to generate more varied individuals.

- **Decide to which child the first pruning will belong**: We will consider the pruning representations of both individuals. We will begin by taking, using lexicographic order, the first pruning item from among the two parents' representations. It will be assigned to the first child with probability γ, and if not assigned, it will be assigned to the other individual. There is one exception to this rule: if an attempt is made to assign a pruning to a tree at a point that has already been pruned, the other individual will be attempted to be assigned instead. For example, if the first child already has the pruning $[0]$, it cannot be assigned the pruning $[0, 1]$; therefore, an attempt will be made to assign it to the other child.
- **Iterate**:This process is iteratively repeated until both parent representations are finished.

We can observe the pseudocode associated with the process of obtaining a population of child solutions from a given population and the crossover of trees in Algorithm 3.

7.1.5. Mutation criterion

A mutation over an already created solution consists of altering that solution and creating another similar one. The purpose of mutation is to create more variability in the search process, allowing the algorithm to explore other zones of the search space that would not normally be explored. For this reason, this aspect enhances exploration rather than exploitation of our search space.

When discussing similarity, we need to introduce a distance metric that allows us to calculate similarities across the search space. To maintain consistency in terms of similarity, it is common to establish a certain boundary that enables the generation of similar solutions around a given one. Using the distance metric, we can consider a topological ball centered at our current solution, with the boundary defined by a predefined radius. A mutation involves randomly generating a solution within that ball.

In our case study, our search space is the space of subtrees of a given matrix tree sharing the root node, which can also be viewed as the space of different pruning sets over that matrix tree. For this purpose, we need to introduce a distance metric over this space. We can take advantage of the fact that individuals from this space are decision trees to consider a distance between these kinds of objects.

7.1.6. Distance between trees

There are various distances between trees that can be utilized. Some of the most classical ones, such as Tree Edit Distance (TED), involve calculating the optimal number of operations of insertion/deletion over one tree to transform it into the other tree. Other metrics have been developed, for instance, in the context of phylogenetic trees, which may include operations like contraction and rearrangement of nodes. Additionally, there are methods known as kernel methods, which translate trees into another scalar feature space, allowing the calculation

Algorithm 3: Children population generation and crossover criterion.

Input: Objective function $f = (f_1, \ldots, f_n)$, previous population P.
Parameters: crossover probability (p_c), crossover parameter (γ).
Output: Children population P'.

```
Create a children population P' = [].
for Half the population size n_i/2 do
    Select 2 random individuals P_{1,1} and P_{1,2}.
    Select the best between them (P_1) with respect to the objective function f.
    Do the same to select another parent (P_2).
    if random uniform number in [0,1) < p_c then
        Create two empty representations for children P'_1 and P'_2.
        Join both P_1 and P_2 pruning representations, P_t, preserving order.
        Assign a random probability γ ∈ [0.4, 0.6] to P'_1.
        for Each element p in P_t, chosen with respect to lexicographical order do
            if The node represented in p has already been pruned in P'_1 then
                if p has not been pruned in P'_2 then
                    Assign p to P'_2.
                else
                    Discard p.
            else if The node represented in p has already been pruned in P'_1 then
                if p has not been pruned in P'_2 then
                    Assign p to P'_2.
                else
                    Discard p.
            else
                if Random uniform number in [0,1] < γ then
                    Assign p to P_1.
                else
                    Assign p to P_2.
        Add P'_1 and P'_2 to children population P'.
    else
        Add P_1 and P_2 to children population P'.
return Children population P'.
```

of a dot product between the representations of both trees. The resulting value can then be used to compare them.

For our algorithm, a modification of the Tree Edit Distance criterion will be utilized to define a specific neighborhood for decision trees. Given a decision tree, its neighborhood will consist of all trees that can be generated by applying contraction/expansion operations on a specific leaf (pruning), up to a maximum of a levels. To do this, first, a node will be chosen. Subsequently, it will be decided whether to contract/expand, and how many levels (x) randomly between 1 and $\left\lceil \frac{\text{Max depth}}{10} \right\rceil$, where Max depth is the maximum depth reached in the matrix tree. A contraction involves pruning its x-th predecessor, and an expansion (only for prunings) involves undoing a pruning and advancing downwards a total of x positions. It is important to note that, for example, if pruning $[0]$ becomes pruning $[0,1,0]$, the entire branch hanging from node $[0,1]$ and the entire branch hanging from node $[0,1,1]$ are not pruned, so they will reach the end of the matrix tree. This is done to avoid drastically increasing the number of prunings performed, since in the case of extending a node by x levels, at most 2^x new prunings would be introduced.

We can see an example of this with a radius distance of 1 in Figure 7.4. All possible mutations in that figure can be seen in Figure 7.5.

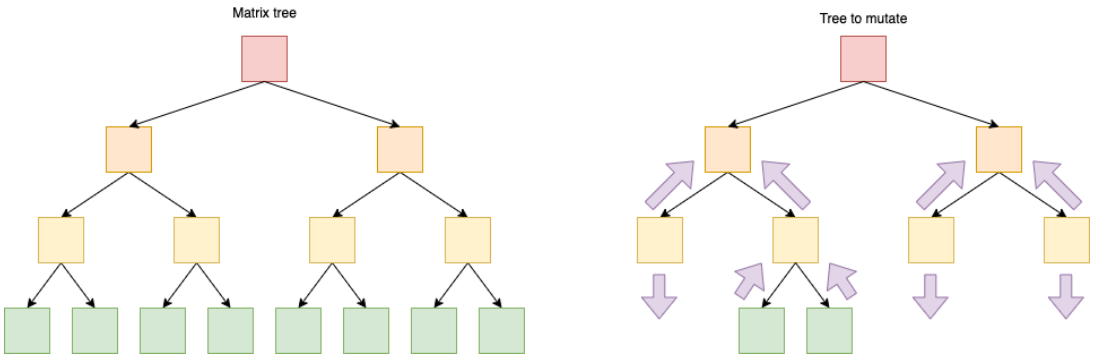


Figure 7.4.: Representation of a matrix tree and a given tree to mutate for distance = 1. Purple arrows indicate the possible paths that can be traversed for mutation.

Having considered this distance criterion between trees, it is clear that the leaves representation would be more beneficial for its implementation, as it utilizes all the actual leaves of the tree to be mutated. It is much more efficient to implement a method to translate between the two representations, pruning representation and leaves representation, than to work with the pruning representation using this mutation criterion.

The pseudocode associated with the mutation process of an individual can be found in Algorithm 4.

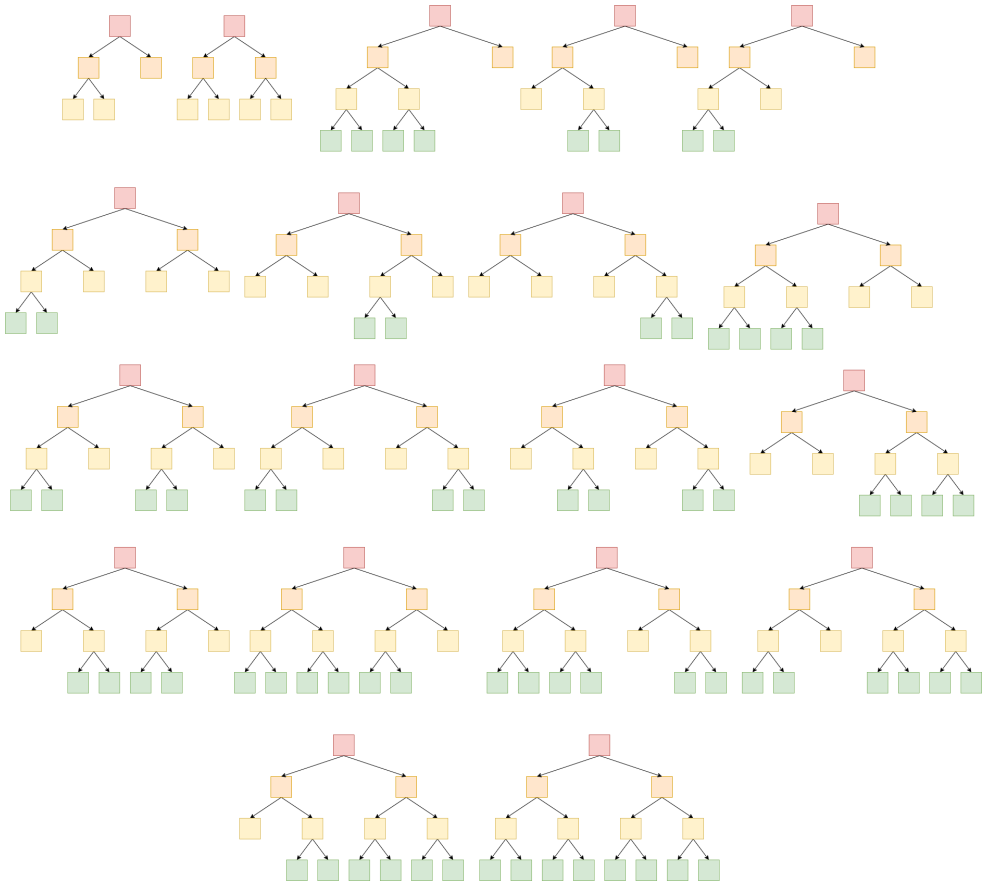


Figure 7.5.: All possible mutations of the tree shown in figure 7.4.

Algorithm 4: Mutation of an individual.

Input: Individual to mutate I.
Parameters: Mutation probability (p_m).
Output: Mutated solution I'.

if *Random uniform number in the range* $[0, 1] < p_m$ **then**
  $L =$ leaf representation of I.
  Select $l \in L$ with uniform probability.
  Decide with uniform probability to go up (if not root node) or down to any child (if not leaf node).
  Randomly select the amount of levels $x \in \{1, \ldots, \left\lceil \frac{\text{Max depth}}{10} \right\rceil\}$.
  Change the value of l for which was selected, and insert it with respect to lexicographical order in I'.
  Convert I' to pruning representation.
else
  $I' \leftarrow I$.
return *Mutated solution* I'.

7.1.7. Generational replacement

The generational replacement will be done in a similar manner to how it is performed in the NSGA-II algorithm, which will be later studied in detail. To do this, all solutions from the current population, plus all solutions from the offspring population (after crossover and mutation), are considered and sorted similarly to how it is done in NSGA-II. Dominance fronts are established, and solutions are selected from there. In the dominance front that would fill the population size, the crowding distance is calculated to select individuals with greater diversity.

7.2. SWOT Analysis

The SWOT analysis for this algorithm can be seen in Figure 7.6.

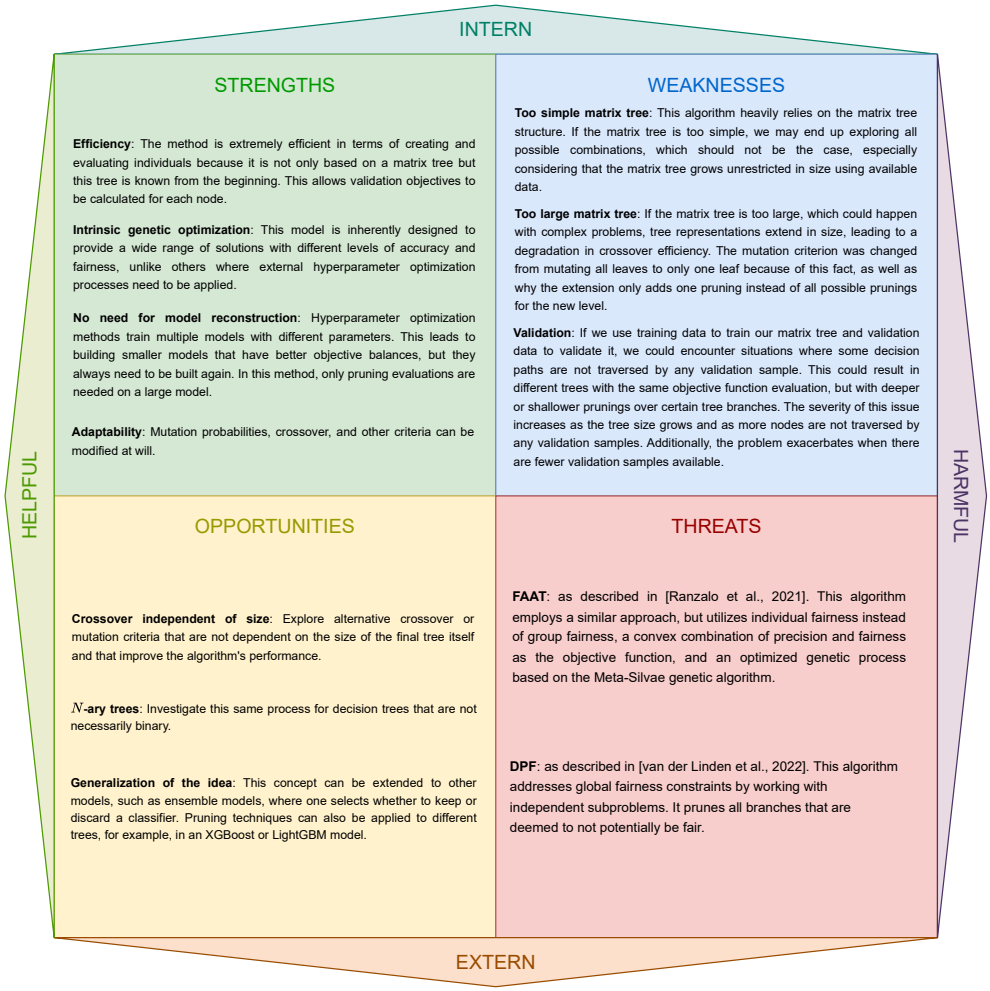


Figure 7.6.: SWOT Analysis for the FGP algorithm. The references for the threat algorithms are as follows: [Ranzato et al., 2021] y [van der Linden et al., 2022].

8. Third algorithm: Fair LightGBM

In this chapter, the last developed algorithm, based on the LightGBM algorithm, will be discussed. LightGBM is conceptually a Gradient Boosting Decision Tree (GBDT), but with some extra features that enhance its scalability in terms of high dimensionality, accounting for both features and instances in our datasets.

Gradient Boosting in Machine Learning concepts will be introduced, with a focus on Gradient Boosting Decision Trees (GBDTs) conceptually. Related algorithms such as XGBoost and LightGBM will be discussed. Particularly, LightGBM will be the main focus, and it will be explained how we can modify the gradient function to incorporate fairness into it.

8.1. Introduction to Gradient Boosting in Machine Learning

In this section, we are going to introduce basic concepts for Gradient Boosting (GB). We will begin by introducing the mathematical setup needed to understand what boosting entails for parametrized functions. Then, we will extend it to using a function space approach and explore why it does not work when we have a finite data sample. Finally, we will discuss regularization.

8.1.1. Mathematical setup

In order to conceptually understand what gradient boosting is, we have to clarify the context in which we are working [Friedman, 2001]. To begin with, in every machine learning problem, the primary goal is usually to predict the value of a random variable y, also known as the response variable, which is initially unknown, using the values of some known variables, $X = \{x_1, ..., x_n\}$, referred to as explanatory variables. In theory, there exists a function $F : \mathbb{R}^n \to \mathbb{R}$ which maps X into y, and our aim is to estimate that function F from a finite set of known data points $\{X_i, y_i\}$, where $i \in \{1, ..., n\}$. The estimation is performed in such a way that it minimizes the expectation of a loss function, typically defined by the user as $L(y, F(X))$, over all possible distributions of (X, y).

$$F* = \arg\min_F E_{y,X} L(y, F(X)) = \arg\min_F E_x[E_y(L(y, F(X))|X] \tag{8.1}$$

Some frequent regression loss functions are $(y - F(X))^2$ and $|y - F(X)|$, which are named squared error and absolute error, respectively. For classification, the negative log-likelihood function $-(\sum_{i=1}^{n} y_i \log(F(X)) + (1 - y_i) \log(1 - F(X)))$, where $y \in 0, 1$ and F is a function that returns probabilities for belonging to the positive class, typically the sigmoid function, is one of the most commonly employed. It will be studied in section 8.3.

The main problem is that the space in which to find the function F is infinitely dimensional, so some restrictions have to be imposed on it in order for the problem to be tractable. For that reason, one of the most common approaches is to use a parametrized class of functions,

denoted as $F(X;P)$, where $P = \{p_1, ..., p_m\}$ is a set of parameters for that function, and its values uniquely identify one function.

The function to be studied is a parameterized function, which follows an additive expansion expression:

$$F(X;\{\beta_k, A_k\}) = \sum_{k=1}^{K} \beta_k h(X;A_k) \tag{8.2}$$

The function $h(x;a_k)$ is itself another parametrized function, utilizing parameters $A = \{a_1, \ldots, a_{m_2}\}$. If we intend to employ this expansion in a practical environment, the function h must be simple enough to be approximated multiple times within a given timeframe. Many widely used machine learning models utilize this additive expansion, including neural networks, radial basis functions, and support vector machines. In our scenario, the function h will be a simple decision tree, and the parameters will represent the splitting variables.

When using parametrized functions, the problem of function estimation can be rewritten using the following notation:

$$\begin{aligned} P^* &= \arg\min_P \phi(P), \\ \phi(P) &= E_{y,x} L(y, F(X;P)) \\ F^*(X) &= F(X;P^*) \end{aligned} \tag{8.3}$$

We do have to apply numerical optimization methods in order to solve this equation in most cases, as it continues to be a difficult problem analytically. An expression which is commonly used is:

$$P* = \sum_{k=0}^{K} p_k \tag{8.4}$$

So in this sense, we start with an initial guess p_0, and subsequently refine the value of these parameters, taking into account all previous steps. The calculation of each subsequent p_k is determined by the optimization method. One of the simplest and most frequently used methods to refine this value is the steepest descent, which involves calculating the derivative of the ϕ function with respect to the parameters:

$$g_m = \{g_{j,m}\} = \left\{ \left[\frac{\partial \phi(P)}{\partial P_j} \right]_{P=P_{m-1}} \right\}, P_{m-1} = \sum_{i=0}^{m-1} p_i \tag{8.5}$$

The step taken will be $p_m = -\rho_m g_m$, where:

$$\rho_m = \arg\min_\rho \phi(P_{m-1} - \rho g_m) \tag{8.6}$$

The negative gradient $-g_m$ defines the steepest descent direction, and ρ_m is the optimal decreasing value along that direction.

8.1.2. Numerical optimization using function spaces

Let us suppose that we are not dealing with parametrized functions, but instead we are considering a function space. Returning to the initial expression:

$$\phi(F) = E_{y,X} L(y, F(X)) = E_X[E_y[L(y, F(X))]|X] \tag{8.7}$$

We can express for a certain X:

$$F^*(X) = E_y[L(y, F(X))|X] \tag{8.8}$$

Despite being in an infinite-dimensional space of functions, datasets only contain a finite amount of data. Therefore, we will have only a finite set of $F(X_i)$, where $i \in \{1, \dots, N\}$. We will optimize using the numerical optimization procedure discussed in section 8.1.1, so the solution will be:

$$F^*(X) = \sum_{m=0}^{M} f_m(X) \tag{8.9}$$

where $f_0(X)$ is the initial guess, and the subsequent ones are the boosts over that initial guess. For steepest-descent:

$$f_m(X) = -\rho_m g_m(X) \tag{8.10}$$

where:

$$\begin{gathered} g_m(x) = \left[\frac{\partial \phi(F(X))}{\partial F(X)}\right]_{F(X)=F_{m-1}(X)} = \left[\frac{\partial E_y[L(y, F(X))|X]}{\partial F(X)}\right]_{F(X)=F_{m-1}(X)} \\ F_{m-1}(X) = \sum_{i=0}^{m-1} f_i(X) \end{gathered} \tag{8.11}$$

And finally, assuming that we can interchange differentiation and integration, which is not true in general, we can obtain the final expression:

$$g_m(X) = E_y\left[\frac{\partial L(y, F(X))}{\partial F(x)}|X\right]_{F(X)=F_{m-1}(X)} \tag{8.12}$$

and ρ_m is determined by the line search:

$$\rho_m = \arg\min_{\rho} E_{y,X} L(y, F_{m-1}(X) - \rho g_m(X)) \tag{8.13}$$

8.1.3. The problem with finite data samples

The issue with having a finite data sample using the previous approach is that we have to estimate the distribution of (X, y) with a finite data sample, $\{X_i, y_i\}$, where $i \in 1, \dots, N$. $E_y[\cdot|X]$ is generally not accurately estimable given only the finite number of values X_i, and those known data points have little to no relevance, as we would be interested in estimating $F^*(X)$ at values of X different from the known ones. For this reason, we need to impose a certain level of smoothness on the actual function F in order to infer F^* using the available information. We will almost certainly encounter some error, but our ability to work with that data will significantly change. The most common way to impose smoothness is, again, by considering a parametrized class of functions to search from. In this sense, we will search for

parameters that match:

$$\{\beta_m, A_m\} = \arg\min_{\{\beta'_m, A'_m\}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta'_m h(x_i; A'_m)\right) \tag{8.14}$$

Typically, there are situations where this expression becomes intractable, even with the steepest-descent optimization procedure. In such cases, we can resort to a greedy stagewise approach. We can calculate the parameters using the following expression:

$$(\beta_m, A_m) = \arg\min_{\beta, A} \sum_{i=1}^{N} L(y_i, F_{m-1}(X_i) + \beta h(x_i; A)) \tag{8.15}$$

Where the function F_m is calculated using only the previous one (which internally contains all previous additions):

$$F_m(X) = F_{m-1}(X) + \beta_m h(X; A_m) \tag{8.16}$$

And this is done $\forall i \in \{1, \ldots, M\}$. This procedure is called boosting, and a typical loss function to use in this context is the log loss function, which will be studied later. The function h is called a weak classifier or base learner and is typically a decision tree with restricted growth, aiming to avoid over-generalization and focus on specific regions of the decision space. In this case, given any approximate solution $F_{m-1}(X)$, the function $\beta_m h(X; A_m)$ can be seen as the best greedy correction to the approximator function, adapting to the data points. The value $h(X; A)$ can be considered as a direction, the steepest one, and the parameter β is the best value in that direction.

The value of the gradient function is calculated as follows:

$$-g_m(X_i) = -\left[\frac{\partial L(y_i, F(X_i))}{\partial F(X_i)}\right]_{F_(X)=F_{m-1}(X)} \tag{8.17}$$

This gradient is defined only at the data points $\{X_i\}$ and cannot be generalized to other X values. One way to enable generalization is to select from the parametrized class the function $h(X; A_m)$ that is most similar to $-g_m \in \mathbb{R}^N$. This function from the parametrized function space correlates the most with $-g_m(X)$ over the data distribution. It can be obtained from the following expression:

$$A_m = \arg\min_{A, \beta} \sum_{i=1}^{N} [-g_m(X_i) - \beta h(X_i; A)]^2 \tag{8.18}$$

This parameter A_m can be used to calculate $h(x; A_m)$ instead of using $-g_m(x)$, which is the one used in the steepest-descent strategy. Specifically, the line search is performed:

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, F_{m-1}(X_i) + \rho h(X_i; A_m)) \tag{8.19}$$

With this calculation, the next approximation to the real function can be calculated by updat-

ing the previous one as follows:

$$F_m(x) = F_{m-1}(x) + \rho_m H(X; A_m) \tag{8.20}$$

So what we are doing here is fitting a weak smooth classifier $h(X; A)$ to the responses of the data points we have, $\{\tilde{y}_i = -g_m(X_i)\}$, for all i in the range from 1 to N. This enables us to transform the problem into a least-squares minimization problem, followed by a single parameter optimization. This implies that for any $h(X; A)$ for which a feasible least-squares algorithm exists to solve the last equation of ρ_m, we can use this approach to minimize any differentiable loss $L(y, F)$ in our forward additive modeling. This leads to the following algorithm, using the steepest-descent algorithm:

Algorithm 5: Gradient Boost.

Input: L, $X_i, y_i, i \in \{1, ..., N\}$, M, h

1 $F_0(X) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2 **for** $m \in \{1, ..., M\}$ **do**
3 $\quad \tilde{y}_i = -\left[\frac{\partial L(y_i, F(X_i))}{\partial F(X_i)}\right]_{F(X)=F_{m-1}(X)}, i \in \{1, ..., N\}$
4 $\quad A_m = \arg\min_{A,\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(X_i; A)]^2$
5 $\quad \rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(X_i) + \rho h(X_i; A_m))$
6 $\quad F_m(X) = F_{m-1}(X) + \rho_m h(X; A_m)$
7 **end**
8 **return** F_M

If the weak model h is a decision tree, we will call this algorithm Gradient Boost Decision Tree (GBDT).

In line 4 of this algorithm the estimation of the parameters is done by a lest-squares procedure, but it is not the only one that could be used. It is used by default as it has good computational properties and can be generally done in an efficient way, but any other could also be employed to do that same task of finding the best parameters for the weak classifier.

8.1.4. Regularization

The process of fitting training data helps us find a function that approximates the data well, assuming that the function to predict is smooth enough. We are able to achieve good results, up to a certain point. Overfitting is a common problem, characterized by fitting too closely to the training data, which can hinder the generalization ability of the learned function to data outside the training set. To address this issue, various techniques have been proposed, known as regularization techniques.

In the context of additive models, the number of components M may seem like the most natural regularization parameter. A higher value of M typically results in more accurate models for the training data, as the expectation of the loss function decreases. However, using fewer terms for regularization may lead to better results. Nonetheless, there are studies indicating that other regularization techniques often perform better than this approach [Copas, 1983]. Particularly, one technique involves adding a regularization parameter ν, typically referred

to as the learning rate, during each boost of the function, as follows:

$$F_m(X) = F_{m-1}(X) + \nu\rho_m h(X; A_m), 0 < \nu \leq 1 \tag{8.21}$$

This technique is often called shrinkage, and with it, there are two direct regularization parameters in the model, M and ν. Both parameters are directly related, as lower values for ν cause the best value for M to potentially increase, as updates affect the model less. The tradeoff between these two parameters has been studied and documented [Friedman, 2001].

8.1.5. Example of Gradient Boosting algorithm: XGBoost algorithm

As an example of a powerful algorithm based on the gradient boosting ideas described before using trees, we have the outstanding eXtreme Gradient Boosting, or XGBoost algorithm, created in 2016 [Chen and Guestrin, 2016]. It is a popular and powerful algorithm based on GBDT, known for its speed, scalability, and performance on large datasets. Some of its main features will now be listed:

- **Regularization and Pruning**: XGBoost includes regularization parameters to control overfitting. It introduces L1/L2 penalties, calculated using the residuals of the leaves [Johnson and Zhang, 2014].
- **Handles Sparse Data**: XGBoost can efficiently handle sparse datasets using the weighted quantile sketch algorithm.
- **Parallel Learning and Out-of-Core Computing**: The algorithm features a block structure for parallel learning, which allows it to scale using multicore machines or clusters. It also supports out-of-core computing, utilizing data structures based on disk storage rather than memory storage, enabling it to handle very large datasets.
- **Optimization Parameters**: XGBoost includes a set of parameters that can help optimize the learning procedure, controlling all the aspects mentioned above, apart from other ones.

8.2. LightGBM algorithm

LightGBM, developed by Microsoft, is an open-source gradient boosting algorithm designed for efficiency with large datasets [Ke et al., 2017]. It was introduced in 2017 with the goal of improving training speed and predictive performance in gradient boosting algorithms. It is a very popular algorithm that has achieved state-of-the-art results for many problems across diverse knowledge fields. It includes scalability tweaks that help the algorithm be utilized in almost any situation, including big data, sparse data, efficiently handling instance weights, taking advantage of parallel and distributed computation scenarios, and more.

Its main features are as follows:

- **Histogram-based split points**: The most time-consuming aspect when considering a GBDT model is finding the optimal split points during the training of each individual decision tree. To address this, a method based on histograms is utilized. This method involves binning continuous features into discrete bins and then constructing

feature histograms from these bins to determine the best split points. This approach is considerably more memory-efficient and speeds up training.

- **Gradient-based One-Side Sampling (GOSS)**: A new method for instance sampling is proposed. This method assigns a weight to each instance based on its gradient. Instances with small gradients typically have low error, indicating that the model is already well-trained on them. However, removing all such instances could alter the data structure. To address this, the algorithm retains the top $\alpha\%$ instances with the highest gradients. Instead of discarding the remaining instances, it randomly subsamples a proportion β from the remaining $(100 - \alpha)\%$. This approach enhances model performance by reducing the number of instances while intelligently selecting which instances to keep, without significantly compromising model generalization capabilities.

- **Exclusive Feature Bundling (FEB)**: This feature is crucial for large-scale datasets and involves reducing the number of features with minimal loss. It entails creating feature bundles for mutually exclusive features, which ultimately produce the same histograms. However, there are two main issues:
  - **Greedy bundling**: Determining which features should be bundled together is an NP-hard problem, as it can be reduced to the graph coloring problem. Therefore, a greedy strategy is employed. Additionally, in the final implementation, a low conflict rate is assumed to encourage further bundling, thereby enhancing algorithm performance without significantly sacrificing accuracy.
  - **Bundling formation**: In order to merge features together to create the same histogram, a straightforward yet effective approach is to apply an offset to each or some features. This ensures that values from different features fall into different bins of the histogram. For example, if one feature takes values in the range $[0, 1)$ and another in the range $[0, 2)$, an offset can be added to the second feature so that it takes values in the range $[1, 3)$, thereby achieving the desired property

  Utilizing this approach, sparse features can be bundled together, eliminating zero-valued computation. While this can be highly advantageous for high-dimensional sparse datasets, it may degrade performance for small or non-sparse datasets. Despite this issue, the calculation remains relatively efficient ($\mathcal{O}(z)$, where z represents non-zero data).

- **Leaf-wise tree growth**: LightGBM employs this strategy instead of depth-wise tree growth, as utilized by C4.5, CART, or XGBoost. This approach expands the leaf with the maximum loss reduction, rather than attempting to expand all nodes at the same level in each step. While it can reduce loss more effectively than depth-wise growth, it may lead to overfitting by producing more complex and deeper trees. However, the complexity can be controlled using regularization parameters, which significantly aid the algorithm in finding optimized trees.

In addition to its optimized speed, efficient handling of large datasets, and memory efficiency, LightGBM's implementation also integrates high parallelism and distributed learning, along with a wide array of parameters for tuning and usage. Its performance in terms of speed and accuracy ranks among the top in current data science competitions and it is widely employed in numerous real-world applications [Xing et al., 2024, Zhuang et al., 2024, Truong et al., 2024].

In summary, LightGBM is a powerful algorithm based on GBDTs, with features that facilitate efficient adaptation to nearly any situation in terms of speed, accuracy, and scalability. For these reasons, it has been chosen as the base algorithm for our latest method.

8.3. The log loss function

We will now discuss the Log loss function. This is a very common loss function mainly used in the context of binary classification.

The log loss function has the following structure:

$$L(y,x) = \log\text{loss}(y,x) = -\left(y\ln\left(\frac{1}{1+e^{-x}}\right) + (1-y)\ln\left(1-\frac{1}{1+e^{-x}}\right)\right) \tag{8.22}$$

The value y is expected to take values in $\{0,1\}$, while the value of x is expected to take values in $\mathbb{R}$. Inside the log loss function, another very important function appears, which is the sigmoid function. It is expressed as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{8.23}$$

It is important to highlight some good properties of this function, such as being strictly monotonic as well as:

$$\lim_{x\to\infty}\sigma(x) = 1,\ \lim_{x\to-\infty}\sigma(x) = 0,\ \text{and also } \sigma(0) = \frac{1}{2} \tag{8.24}$$

So this function maps values from $\mathbb{R}$ to the range $[0,1]$, which can be interpreted as a probability or degree of certainty of belonging to the positive class. With this definition, we can redefine our function in terms of y and σ as follows:

$$L(y,\sigma) = -\left(y\ln\sigma + (1-y)\ln\left(1-\sigma\right)\right) \tag{8.25}$$

Additionally, the derivative of the sigmoid function can be calculated as follows:

$$\frac{\partial\sigma(x)}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x)) \tag{8.26}$$

Using this intermediate function, we can calculate both the derivative and Hessian functions of our loss function with respect to x using the chain rule:

$$\begin{gathered}\frac{\partial L(y,x)}{\partial x} = \frac{\partial L(y,\sigma)}{\partial\sigma}\frac{\partial\sigma(x)}{\partial x} = \frac{\partial L(y,\sigma)}{\partial\sigma}\sigma(x)(1-\sigma(x)) = \\ \left(-\frac{y}{\sigma(x)} + \frac{1-y}{1-\sigma(x)}\right)\sigma(x)(1-\sigma(x)) = -y(1-\sigma(x)) + (1-y)\sigma(x) = \sigma(x) - y\end{gathered} \tag{8.27}$$

and for the second-order derivative:

$$\frac{\partial^2 L(y,x)}{\partial^2 x} = \frac{\partial}{\partial x}\frac{\partial L(y,x)}{\partial x} = \frac{\partial}{\partial x}(\sigma(x)-y) = \frac{\partial}{\partial x}\sigma(x) = \sigma(x)(1-\sigma(x)) \tag{8.28}$$

By employing log loss as our loss function, defining a LightGBM model and finding optimized models for this metric is straightforward. Log loss is a natural and interpretable loss function with very simple first and second-order derivatives, which enhance performance.

8.4. Introducing a fainess-aware loss function

In order to define a fairness-aware loss function, we will create a continuous extension of our classical fairness functions. Let us begin with the difference in the False Positive Rate (FPR) metric. We need to consider that the classical metric is defined by the following expression:

$$\mathrm{FPR}_{\mathrm{diff}} = \left| \frac{\mathrm{FP}_1}{\mathrm{FP}_1 + \mathrm{TN}_1} - \frac{\mathrm{FP}_0}{\mathrm{FP}_0 + \mathrm{TN}_0} \right| \tag{8.29}$$

This metric is commonly used in the context of classification, where our predicted values are either 0 or 1. However, in this case, we have a continuous output (which will then be transformed into actual labels). We can consider the values of our prediction after applying the sigmoid function $(\sigma(x))$ as a score that resembles the probability of that sample belonging to the positive class. However, it does not necessarily have to be treated as a strict probability, as we do not impose any probability function constraint. With these scores, we can generalize this function and others by first generalizing the concepts of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN).

We will use FP as an example to illustrate this. To define FP in classical classification terms, we will use an auxiliary indicator function, $\mathbb{1}_{i,j,k}$:

$$\mathbb{1}_{i,j,k}(x,y,p) = \begin{cases} 1 & \text{if } x = i \wedge y = j \wedge p = k \\ 0 & \text{in any other case} \end{cases} \tag{8.30}$$

We can subsequently define the functions $\mathbb{1}_{i,j,-}$, $\mathbb{1}_{i,-,k}$ and $\mathbb{1}_{-,j,k}$ in the same manner as before, without considering the values of p, y and x respectively. For instance, $\mathbb{1}_{i,j,-}$ will be defined as:

$$\mathbb{1}_{i,j,-}(x,y,p) = \mathbb{1}_{i,j,-}(x,y) = \begin{cases} 1 & \text{if } x = i \wedge y = j \\ 0 & \text{in any other case} \end{cases} \tag{8.31}$$

Using this definition, one can express FP as a function as follows. Let x_i be the prediction for sample i, y_i its actual class, and p_i the value of its protected attribute. Let $X = (x_1, \ldots, x_N)$ be the column vector of predicted values, $Y = (y_1, \ldots, y_N)$ the column vector of ground truth outputs, and $P = (p_1, \ldots, p_N)$ the column vector of protected attributes. The definition can be as follows:

$$\mathrm{FP}(X,Y) = \sum_{i=0}^{N} \mathbb{1}_{1,0,-}(x_i, y_i) \tag{8.32}$$

The sum just defined gives us the number of false positives within our sample in a classification context. Now, we would like to extend this definition to problems with classification scores. In this case, if X is the column vector of scores, we will define FP as follows:

$$\mathrm{FP}(X,Y) = \sum_{i=0}^{N} \mathbb{1}_{-,0,-}(y_i) x_i \tag{8.33}$$

To illustrate the meaning of the previous expression, let us consider an example. Suppose a score of 0.2 is assigned to an instance belonging to class 0. In this scenario, we have an error of 0.2 with respect to the actual class. This is because we are predicting it belongs to class 1 with a strength of 0.2, while the score of it belonging to class 0 is 0.8. For this reason, the value for FP will be 0.2.

We have to clarify that, in our context, when using the log loss function, the scores for the prediction are obtained after applying the sigmoid function to the values of x, as explained in section 8.3.

Let us now consider a vectorial definition of FP, TP, FN, and TN using the σ function vector-wise, which will help us to calculate the derivative and hessian of the loss function about to be defined. Considering a vector of terms $((x_1, y_1, p_1), \ldots, (x_N, y_N, p_N))$, then we can define the extension of the σ function vector-wise as follows: $\mathbb{1}_{i,j,k}(((x_1, y_1, p_1), \ldots, (x_N, y_N, p_N))) = (\mathbb{1}_{i,j,k}(x_1, y_1, p_1), \ldots, \mathbb{1}_{i,j,k}(x_N, y_N, p_N))$.

The rest of $\mathbb{1}$ function extensions are defined analogously. Using the above definition of FP using scores, we can consider a vector version of that definition:

$$\text{FP}(X, Y) = \mathbb{1}_{-,0,-}(Y)^T X \tag{8.34}$$

In this definition, there is a matrix product (which is equivalent to a dot product) of values of $\mathbb{1}_{-,0,-}(Y)$ and X. What is important to note is that $\mathbb{1}_{-,0,-}(Y)$ is constant with respect to the vector of scores X, which is why we will use an abuse of notation writing $\mathbb{1}_{-,0,-}(Y)^T = \mathbb{1}^T_{-,0,-}$. Using this notation, the final expression of our metrics will be:

$$\text{FP}(X, Y) = \mathbb{1}^T_{-,0,-} X, \text{TP}(X, Y) = \mathbb{1}^T_{-,1,-} X \text{FN}(X, Y) = \mathbb{1}^T_{-,1,-}(1 - X), \text{TN}(X, Y) = \mathbb{1}^T_{-,0,-}(1 - X) \tag{8.35}$$

With this vector notation, our FPR difference can be written in vectorial form using scores as follows:

$$\text{FPR}_{\text{diff}}(X, Y, P) = \left| \frac{\mathbb{1}^T_{-,0,1} X}{\mathbb{1}^T_{-,0,1} X + \mathbb{1}^T_{-,0,1}(1 - X)} - \frac{\mathbb{1}^T_{-,0,0} X}{\mathbb{1}^T_{-,0,0} X + \mathbb{1}^T_{-,0,0}(1 - X)} \right| \tag{8.36}$$

There is a key observation here, which is that $\mathbb{1}^T_{-,0,1} X + \mathbb{1}^T_{-,0,1}(1 - X) = \mathbb{1}^T_{-,0,1} 1$, where 1 represents the N-dimensional vector where all values are 1. This expression is indeed equivalent to the amount of instances i where $y_i = 0$ and $p_i = 1$. Using this property, the final expression for our function is:

$$\text{FPR}_{\text{diff}}(X, Y, P) = \left| \frac{\mathbb{1}^T_{-,0,1} X}{\mathbb{1}^T_{-,0,1} 1} - \frac{\mathbb{1}^T_{-,0,0} X}{\mathbb{1}^T_{-,0,0} 1} \right| \tag{8.37}$$

Another important property of this definition is the value of its derivative. In order to calculate it, we have to take into account that X is a vector whose components are always non-negative, as well as images from any σ function. Using theses:

$$\begin{aligned}\frac{\partial \text{FPR}_{\text{diff}}(X,Y,P)}{\partial X} &= \frac{\partial}{\partial X}\left(\left|\frac{\mathbb{1}^T_{-,0,1}X}{\mathbb{1}^T_{-,0,1}1} - \frac{\mathbb{1}^T_{-,0,0}X}{\mathbb{1}^T_{-,0,0}1}\right|\right) = \frac{\partial}{\partial X}\left(\left|\frac{(\mathbb{1}^T_{-,0,0}1\mathbb{1}^T_{-,0,1} - \mathbb{1}^T_{-,0,1}1\mathbb{1}^T_{-,0,0})X}{\mathbb{1}^T_{-,0,1}1\mathbb{1}^T_{-,0,0}1}\right|\right) \\ &= \frac{1}{\mathbb{1}^T_{-,0,1}1\mathbb{1}^T_{-,0,0}1}\frac{\partial}{\partial X}(|(\mathbb{1}^T_{-,0,0}1\mathbb{1}^T_{-,0,1} - \mathbb{1}^T_{-,0,1}1\mathbb{1}^T_{-,0,0})X|) \\ &= \frac{|\mathbb{1}^T_{-,0,0}1\mathbb{1}_{-,0,1} - \mathbb{1}^T_{-,0,1}1\mathbb{1}_{-,0,0}|}{\mathbb{1}^T_{-,0,1}1\mathbb{1}^T_{-,0,0}1} = \left|\frac{\mathbb{1}_{-,0,1}}{\mathbb{1}^T_{-,0,1}1} - \frac{\mathbb{1}_{-,0,0}}{\mathbb{1}^T_{-,0,0}1}\right| = C\end{aligned} \tag{8.38}$$

which is a constant vector with respect to X (not with respect to Y and P), so the second-order derivative with respect to X will be equal to the vector 0. Furthermore, despite the function being within an absolute value, it is differentiable for any value of X since all its terms are non-negative. These properties are very advantageous for its incorporation into the loss function, considering that it will need to be differentiated.

Taking all of this into account, we are now able to define our fairness-aware loss function using the extension of the FPR_{diff} metric. The function we currently have is as follows:

$$L_f(Y,X,P) = -(1-\lambda)\left(Y\ln\left(\tfrac{1}{1+e^{-X}}\right) + (1-Y)\ln\left(1-\tfrac{1}{1+e^{-X}}\right)\right) + \lambda\left|\frac{\mathbb{1}^T_{-,0,1}\frac{1}{1+e^{-X}}}{\mathbb{1}^T_{-,0,1}1} + \frac{\mathbb{1}^T_{-,0,0}\frac{1}{1+e^{-X}}}{\mathbb{1}^T_{-,0,0}1}\right| \tag{8.39}$$

where $\lambda \in [0,1]$. It can be defined in terms of the sigmoid function (using Σ instead of σ following vector notation), as follows:

$$\begin{aligned}L_f(Y,\Sigma,P) &= -(1-\lambda)\left(Y\ln\Sigma + (1-Y)\ln(1-\Sigma)\right) + \lambda\left|\frac{\mathbb{1}^T_{-,0,1}\Sigma}{\mathbb{1}^T_{-,0,1}1} - \frac{\mathbb{1}^T_{-,0,0}\Sigma}{\mathbb{1}^T_{-,0,0}1}\right| \\ &= (1-\lambda)L(Y,\Sigma) + \lambda\text{FPR}_{\text{diff}}(\Sigma,Y,P)\end{aligned} \tag{8.40}$$

Considering $L(Y,\Sigma)$ as the logistic function discussed in section 8.3. This function exhibits a similar structure to that utilized in the FDT algorithm, constituting a convex combination between the traditional loss function, $L(Y,\Sigma)$, and the desired fairness criterion, here represented by $\text{FPR}_{\text{diff}}(\Sigma,Y,P)$. While this definition may appear straightforward, it is essential to remember that it required a continuous extension definition of the fairness criterion. Moreover, this definition proves highly advantageous for computing the first and second-order derivatives essential for executing the algorithm. Let us delve into their expressions, starting with the first order derivative:

$$\begin{aligned}\frac{\partial L_f(Y,\Sigma,P)}{\partial X} &= \frac{\partial L_f(Y,\Sigma,P)}{\partial \Sigma}\frac{\partial \Sigma(X)}{\partial X} \\ &= \left((1-\lambda)\frac{\partial L(Y,\Sigma,P)}{\partial \Sigma} + \lambda\frac{\partial \text{FPR}_{\text{diff}}(\Sigma,Y,P))}{\partial \Sigma}\right)\frac{\partial \Sigma(X)}{\partial X} \\ &= \left((1-\lambda)\left(-\frac{Y}{\Sigma(X)} + \frac{1-Y}{1-\Sigma(X)}\right) + \lambda C\right)(\Sigma(X)(1-\Sigma(X)) \\ &= (1-\lambda)(\Sigma(X)-Y) + \lambda C\Sigma(X)(1-\Sigma(X))\end{aligned} \tag{8.41}$$

This represents yet another convex combination, this time between the derivative of the

log loss function and the derivative of the FPR_{diff} function. Calculating the second-order derivative is similarly straightforward:

$$\begin{aligned}\frac{\partial^2 L_f(Y,\Sigma,P)}{\partial^2 X} &= \frac{\partial}{\partial X}((1-\lambda)(\Sigma(X)-Y)+\lambda C\Sigma(X)(1-\Sigma(X))) \\ &= (1-\lambda)(\Sigma(X)(1-\Sigma(X))+\lambda C\left((\Sigma(X)(1-\Sigma(X))^2-\Sigma(X)^2(1-\Sigma(X))\right)\end{aligned} \tag{8.42}$$

With these expressions, integrating this new objective into our LightGBM algorithm becomes a simple task.

8.5. SWOT Analysis

The SWOT analysis for this algorithm can be seen in Figure 8.1.

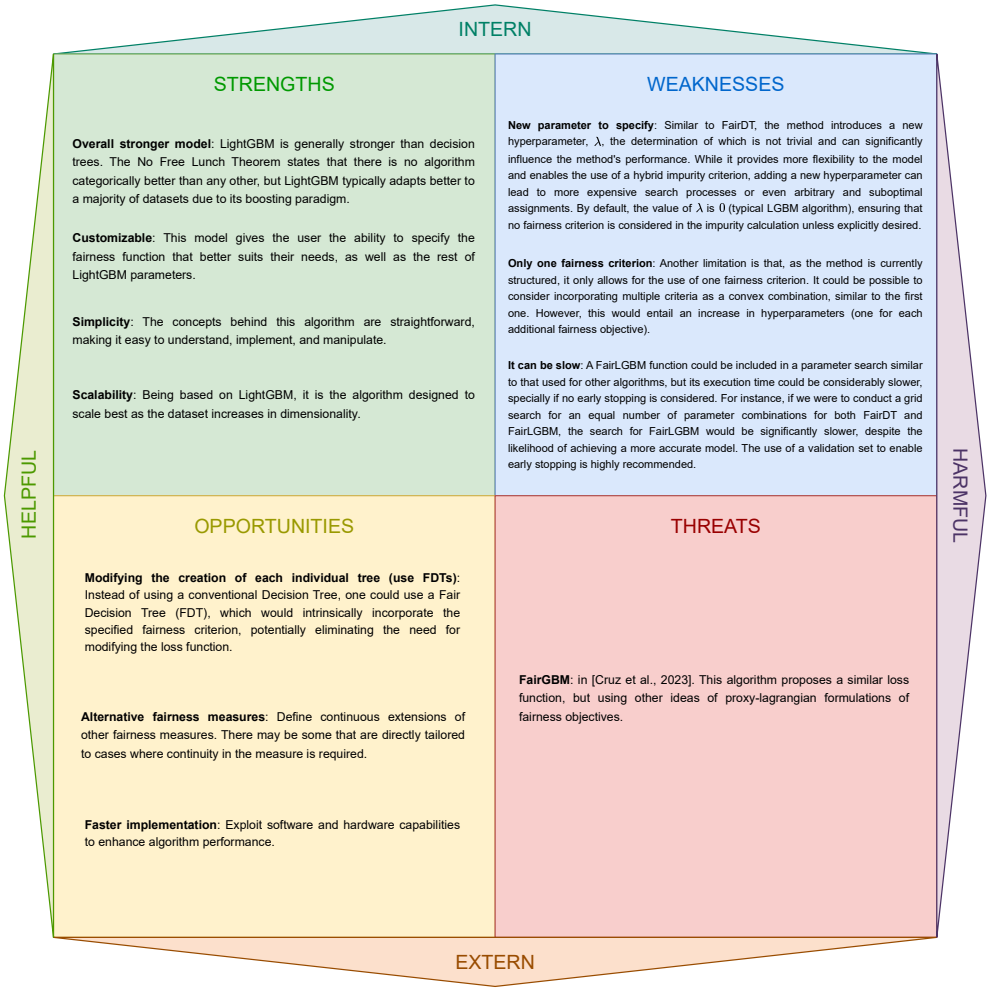


Figure 8.1.: SWOT Analysis for the FLGBM algorithm. The references for the threat algorithms are as follows: [Cruz et al., 2023].

Part IV.

Experimentation

In this part, the experimentation conducted to test the presented algorithms will be discussed, including dataset descriptions and implemented code.

9. Experimental framework

In this chapter, we will discuss the experimental framework to test the developed algorithms, which methodology will be employed, and how they results will be evaluated.

In order to test all of these algorithms, we will have to define an experimental setup that enables us to explore and further comprehend their individual capabilities. For this purpose, we will explain in detail all datasets to which these methods have been applied, along with our experimentation framework and the tests conducted on them.

9.1. Overview of the experimentation

The experimentation will involve testing the 3 algorithms in a controlled environment and conducting individual and comparative studies of each. Additionally, we will include a baseline control algorithm for comparison, which is the decision tree (DT), so in reality, we will be using 4 algorithms.

To conduct these tests, 10 datasets will be selected, upon which each of the algorithms will be tested. These datasets will vary in terms of dimensions, context, and content, allowing us to address a diverse range of cases.

Let us remember that we are in a multi-objective optimization context, so we will not be able to run each algorithm just once for each dataset; we will have to run them several times to find a Pareto-optimal set of solutions. To achieve this, it is proposed to include each algorithm within a process of searching for these solutions. To do this, we will divide our algorithms into two groups:

- **DT, FairDT, and FairLGBM algorithms**: they will be included in a hyperparameter optimization process using the multi-objective genetic algorithm NSGA-II. Each individual in the solution population will be represented by a set of hyperparameters for the respective algorithm, so that the decision space will consist of different values for these algorithm hyperparameters. In this manner, the goal is to find the set of hyperparameters that, after training the algorithm with the same training dataset, achieve a better balance among the considered objectives on a validation dataset.

- **Fair Genetic Pruning algorithm**, a multi-objective genetic search process will be considered, as proposed in the algorithm's description, with the criteria for representing individuals, creating the initial population, crossover, mutation, and replacement outlined in their respective sections FALTA. After generating the matrix tree with the training set, the aim will be to generate pruned trees that achieve the best balance among objectives with the validation set.

9.2. Datasets

Datasets are a key part of our experimentation, as we are focusing on a specific problem that can be found in a wide range of different contexts and situations. For this reason, it is crucial for us to select and identify these datasets where discrimination can occur, as well as the protected attributes they contain that can be the subject of discrimination.

To find a dataset with these characteristics, one cannot arbitrarily choose a dataset and select an attribute that might apparently lead to unfairness. A study must be conducted on the influence of such attributes on the specific problem, which is a task typically requiring professionals in humanitarian or sociological fields. Therefore, it is beyond our scope to choose any protected attribute, and thus, datasets that have been previously studied and known to have some form of injustice will be used.[Fabris et al., 2022].

Datasets selected for this study can be found in the GitHub repository `https://github.com/juliettm/datasets`, which, as of January 30, 2024, is a private repository. It contains datasets that are not open for public use and can only be accessed by explicit request to the authors. Access can be requested and will be granted for research purposes. For the evaluation of this work, only the datasets in their preprocessed and anonymized version will be provided. Our goal deviates from deeply describing and understanding each particular dataset and its fairness implications, but it is important to at least understand their context. For that reason, we will now outline the prediction task and provide some other useful information about the datasets used in our study:

- **Adult dataset**:
  - **Dataset description**: This dataset contains information about U.S. citizens gathered during the March 1994 U.S. Current Population Survey. It includes demographic and socioeconomic dimensions, with features such as education, profession, age, sex, race, personal, and financial condition. It is commonly used as a benchmark dataset for fairness in machine learning.
  - **Prediction task**: Predict whether a given individual earns above $50,000 annually or not.
  - **Protected attribute**: Race.
  - **Dataset shape after preprocessing**: 45222 instances and 14 attributes.
  - **Additional references**: [Becker and Kohavi, 1996]
- **Compas dataset**:
  - **Dataset description**: This dataset contains information about the COMPAS algorithm and its application to citizens of Broward County, Florida, during a time period ranging from 2013 to 2014. Selected attributes for use in our experiments were chosen as outlined in [Friedler et al., 2019], which are the following: sex, age, age_cat, race, juv_fel_count, juv_misd_count, juv_other_count, priors_count, c_charge_degree, c_charge_desc, decile_score y score_text. It is commonly used as a benchmark dataset for fairness in machine learning.

  - **Prediction task**: Predict whether a certain individual will recidivate (and may the police notice that recidivism) within a time period ranging from their release until 2 years afterward.
  - **Protected attribute**: Race.
  - **Dataset shape after preprocessing**: 5278 instances, and 9 attributes.
  - **Additional references**: [Julia et al., 2016] `https://github.com/propublica/compas-analysis`

- **Diabetes dataset**:
  - **Dataset description**: This dataset contains records of patients diagnosed with diabetes who underwent laboratory tests, medications, and stayed for up to 14 days in hospitals. The data was collected from 1999 to 2008 at 130 US hospitals, including information from their integrated delivery networks. It includes demographic information, diagnoses, and other types of hospital procedures.
  - **Prediction task**: Predict if a patient will be readmitted within 30 days of discharge.
  - **Protected attribute**: Sex.
  - **Dataset shape after preprocessing**: 46176 instances and 16 columns.
  - **Additional references**: [Clore and Strack, 2014, Strack et al., 2014]

- **Dutch dataset**:
  - **Dataset description**: This dataset contains information derived from the 2001 census conducted by the Dutch Central Bureau for Statistics to collect data about family composition, economic activities, levels of education, and occupation of Dutch citizens and foreigners from various countries of origin.
  - **Prediction task**: Predict whether a certain individual has a high-income or low-income profession.
  - **Protected attribute**: Sex.
  - **Dataset shape after preprocessing**: 60420 instances and 10 attributes.
  - **Additional references**: [Centraal Bureau voor de Statistiek (CBS) (Statistics Netherlands), 2018]

- **German dataset**:
  - **Dataset description**: "This dataset contains financial information of loan applicants from 1973 to 1975. It includes details about their financial situation, credit history, and personal situation.
  - **Prediction task**: Predict whether a given individual has a low or high risk of repaying a certain loan.
  - **Protected attribute**: Age.
  - **Dataset shape after preprocessing**: 1000 instances and 10 attributes.
  - **Additional references**: [Hofmann, 1994]

- **Insurance dataset**:

  - **Dataset description**: This dataset contains information from a popular Italian car insurance comparison website, including quotes provided by nine companies, collected in 2020. It includes information such as gender, age, vehicle details, and a summary of claim history.
  - **Prediction task**: The prediction task is to predict whether the insurance charges are above or below 40000€.
  - **Protected attribute**: Sex.
  - **Dataset shape after preprocessing**: 1338 instances and 9 attributes.
  - **Additional references**: [Fabris et al., 2021]

- **Obesity dataset**:
  - **Dataset description**: This dataset includes data for estimating obesity levels in individuals from the countries of Mexico, Peru, and Colombia, based on their eating habits and physical condition.
  - **Prediction task**: The prediction task is to determine whether the individual is obese (with obesity type I, II, or III) or not (with insufficient weight, normal weight, overweight level I, or II).
  - **Protected attribute**: Gender.
  - **Dataset shape after preprocessing**: 2111 instances and 23 attributes.
  - **Additional references**: [obe, 2019]

- **Parkinson dataset**:
  - **Dataset description**: This dataset contains biomedical voice measurements collected at home from 42 individuals with early-stage Parkinson's disease who were recruited for a six-month trial of a telemonitoring device for remote symptom progression monitoring. Attributes include information such as subject age, gender, and recruitment date.
  - **Prediction task**: Predict whether the total UPDRS score for a certain patient's voice recording is below 17.1 or not.
  - **Protected attribute**: Sex.
  - **Dataset shape after preprocessing**: 5875 instances and 19 columns.
  - **Additional references**: [Tsanas and Little, 2009]

- **Ricci dataset**:
  - **Dataset description**: This dataset contains information on 118 New Haven (Connecticut) firefighter promotion tests, including the scores and race of the individuals. The case escalated to the US Supreme Court labor case on discrimination, Ricci vs. DeStefano (2009), due to disparate impact.
  - **Prediction task**: The prediction task in this case is to predict whether a given individual will pass a promotion exam (scoring at least 70 points) or not.
  - **Protected attribute**: Race.
  - **Dataset shape after preprocessing**: 118 instances and 5 attributes.

  - **Additional references**: [pro, 2009], https://github.com/algofairness/fairness-comparison/tree/master/fairness/data/raw

- **Student dataset**:
  - **Dataset description**: This dataset contains information on student achievement from two Portuguese secondary schools, collected using questionnaires and school reports. The data attributes include student grades, demographic, social, and school-related features.
  - **Prediction task**: The prediction task is to predict whether the individual's final year grade during the 3rd term for the Portuguese subject will be greater or lower than 12.
  - **Protected attribute**: Sex.
  - **Dataset shape after preprocessing**: 649 instances and 39 attributes.
  - **Additional references**: [Cortez, 2014]

These datasets belong to diverse knowledge fields, such as finance, legal, labor, or medicine, and collectively constitute a potentially good set of datasets for testing our algorithms. They exhibit a good variety in terms of the number of instances, attributes, and contents. None of them are excessively large in either aspect, but they provide a diverse range of datasets to evaluate our algorithms. A scatterplot showing the number of features and samples for each dataset can be seen in Figure 9.1.

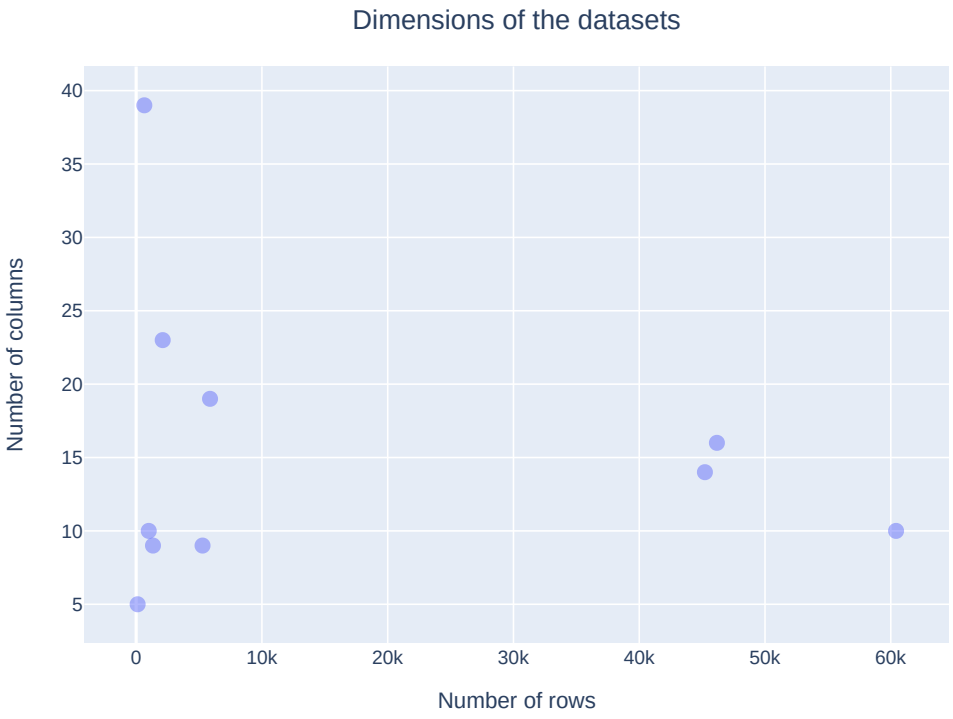


Figure 9.1.: Rows and columns for all datasets included in the experimentation process.

All datasets were preprocessed uniformly, meaning that we will apply the same principles during preprocessing to all datasets. Our objective is not to conduct extensive preprocessing to achieve the very best results possible for each dataset, but rather to preprocess them in a consistent manner to observe general behavioral patterns in our algorithm executions. This will help us determine the effectiveness of the algorithms for their proposed tasks. The preprocessing strategy we will follow is similar to that proposed in [Valdivia et al., 2020]. It involves selecting attributes that contribute to good results, transforming categorical attributes into numerical ones, and converting binary labels into 0 or 1 values. If categorical attributes have an implicit order, they will be numerically encoded; otherwise, a one-hot encoding set of attributes will be generated.

9.3. Decision space, hyperparameters

The decision space cannot be standardized as a common set across all methods because it varies depending on the algorithm chosen and the parameters that control its learning process. Therefore, the parameters considered for the optimization process (for algorithms using the NSGA-II hyperparameter optimization process, given that for Fair Genetic Pruning the decision space is the space of all possible pruning sets) will now be explained:

- **DT algorithm**:
  - **criterion**: This parameter represents the function that measures the quality of the split of an intermediate node in the decision tree. The decision tree generates new nodes from those already present at lower levels as this value becomes smaller. There are two possible values: 'gini' and 'entropy'.
  - **max_depth**: This parameter specifies the maximum depth that the tree can reach. Deeper trees tend to be more complex. The possible values it can take are $\{n \in \mathbb{N} : n \leq m\}$, where m is the depth of the tree learned for the training sample without any growth restriction. (This tree will always belong to the initial population, so this value can be easily set).
  - **min_samples_split**: This parameter specifies the minimum number of samples required to split a node. A larger value makes it simpler for a node to split. The possible values considered are $\{n \in \mathbb{N} : 2 \leq n \leq 40\}$.
  - **max_leaf_nodes**: This parameter sets the maximum number of leaf nodes that the tree can have. A larger value increases the complexity of the tree, allowing for more diverse paths until reaching a final classification. The possible values are $\{n \in \mathbb{N} : n \leq l\}$, where l is the number of leaf nodes of the tree learned for the training sample without any growth restriction. (This tree will always belong to the initial population, so this value can be easily set).
  - **class_weight**: This parameter assigns a weight to each class, which is useful for imbalanced datasets. The possible values that will be used are $\{n \in \mathbb{N} : 1 \leq n \leq 9\}$. This means that the negative class will have a weight of n, and the positive class will have a weight of $10 - n$ ($n = 5$ for equal weights).
- **FairDT algorithm**: Same parameters as for Decision Trees (DT) with the following additional modifications/additions:

  - **criterion**: Instead of using gini and entropy, it utilizes the newly created functions gini_fair and entropy_fair. (Using gini/entropy_fair with a fair parameter of 0 is equivalent to using gini/entropy).
  - **fair_param**: This parameter controls the importance of fairness in the learning process. A low value promotes a fairer model, while a high value could overshadow the traditional impurity criterion, which ideally should lead the learning process. The possible values we will use are $\{x \in \mathbb{R} : 0 \leq x \leq 1\}$.

- **FairLGBM algorithm**:
  - **num_leaves**: This parameter controls the complexity of the tree model. If set to $2^{(}$max_depth$)$, it aims to obtain the same number of leaves as a depth-wise tree. However, in practice, as LightGBM uses leaf-wise trees, trees are typically much deeper than depth-wise trees for a fixed number of leaves. This parameter serves as a regularization parameter. Possible values we will use are $\{n \in \mathbb{N} : 2 \leq n \leq 62\}$.
  - **min_data_in_leaf**: This is another regularization parameter. Its optimal value highly depends on the number of training instances and num_leaves. It controls the minimum amount of data needed for a split to occur. Possible values we will use are $\{n \in \mathbb{N} : 0 \leq n \leq 20\}$.
  - **max_depth**: This parameter explicitly limits the maximum depth the tree can reach. Possible values that will be used for this parameter are $\{n \in \mathbb{N} : n \leq m\}$, where m is the maximum depth among all trees learned for an unrestricted LightGBM model in terms of size. (This LightGBM model will belong to the initial population, so this value can be easily set).
  - **learning_rate**: This parameter is crucial in LightGBM, as it determines the step size at each iteration while moving toward the minimum of the loss function. Typically, a smaller learning rate requires more iterations to converge, resulting in a slower training process but potentially a more accurate model. Possible values that will be used are $\{x \in \mathbb{R} : 0.01 < x < 0.2\}$.
  - **n_estimators**: This parameter specifies the number of boosting stages to perform, and thus the number of estimators trained. More estimators can lead to better learning performance, but it can also lead to overfitting. The values selected are $\{n \in \mathbb{N} : 50 \leq n \leq 200\}$.
  - **feature_fraction**: This parameter controls the fraction of randomly selected features used for training each individual tree. Possible values that will be used are $\{x \in \mathbb{R} : \frac{1}{\text{Num Features}} \leq x \leq 1\}$.
  - **fair_param**: This parameter controls the importance of fairness in the learning process. A low value promotes a fairer model, while a high value could overshadow the traditional impurity criterion, which ideally should lead the learning process. The possible values we will use are $\{x \in \mathbb{R} : 0 \leq x \leq 1\}$.

9.4. Objective space

The target space will be described by the functions we will try to optimize. These functions will not have a high correlation with each other, since then optimizing one of them would

optimize the others, potentially simplifying the model and considering a single objective in that case.

For the study to be conducted, we will only consider the measures of **inverted G-mean** and difference in false positive rate **($\mathbf{FPR_{diff}}$)**. With these two objectives, we aim to find models that achieve high classification accuracy while also ensuring compliance with a specific fairness objective.

However, many other objective functions can be considered. We will present the objectives considered and implemented (even if not used in the final experimentation) in different families, which are: accuracy objectives, fairness objectives, and interpretability objectives.

- **Error objectives**: We aim for our predictions to closely match reality, minimizing the errors made in them.
  - **Error rate**: This is a classic error objective. It is defined as $\frac{\text{FP+FN}}{\text{P+N}}$. This objective is to be minimized and can take values in the range $[0, 1]$.
  - **Inverted G-mean**: We will use the geometric mean criterion, which combines the measures of the true positive rate ($P[p = 1|Y = 1]$) and the true negative rate ($P[p = 0|Y = 0]$). By using the geometric mean criterion, we ensure that a significant improvement in this function requires a joint improvement in both measures. The G-mean measure is defined as $\sqrt{P[p = 1|Y = 1]P[p = 0|Y = 0]} = \sqrt{\text{TPR} \cdot \text{TNR}}$, and our aim is to maximize it. However, since we treat the problem as a minimization one, we will use inverted G-mean = $1 - \sqrt{\text{TPR} \cdot \text{TNR}}$. Possible values for this measure belong to the range $[0, 1]$.
- **Fairness objectives**: We aim for our system to generate fair predictions without discrimination based on an individual's sensitive attribute. Since there are various mutually exclusive families of fairness, as we have already outlined, we will define several measures to address them.
  - **Difference in FPR (False Positive Rate, $\mathbf{FPR_{diff}}$)**: We aim to minimize the difference between false positives ($P[p = 1|Y = 0]$) conditioned on the value of both classes, i.e., the difference between $P[p = 1|Y = 0, A = 0]$ and $P[p = 1|Y = 0, A = 1]$. We will consider the absolute difference between the conditionings on each value of the sensitive attribute, i.e., $|P[p = 1|Y = 0, A = 0] - P[p = 1|Y = 0, A = 1]|$. Using an absolute difference instead of a relative one is preferable for several reasons: possible values for this measure are bounded to the range $[0, 1]$, and the minimum is reached when the difference is 0 (indicating the same probability regardless of the value taken in the sensitive attribute). Additionally, considering a relative difference may equate values such as 0.02 and 0.01 with 0.5 and 1, which might not be preferable. This measure corresponds to the family of equalized odds.
  - **Difference in PPV (Predictive Positive Value, $\mathbf{PPV_{diff}}$)**: We aim to minimize the difference in individuals who, having been predicted as positives, actually were positives ($P[Y = 1|p = 1]$), with respect to both classes, i.e., $P[Y = 1|p = 1, A = 1]$ and $P[Y = 1|p = 1, A = 0]$. Similar to before, we will consider the absolute difference, thus minimizing $|P[Y = 1|p = 1, A = 1] - P[Y = 1|p = 1, A = 0]|$. The minimum occurs when both probabilities are equal. This measure corresponds

to the family of predictive rate parity. Possible values for this measure belong to the range $[0, 1]$.

- **Difference in PNR (Predictive Negative Rate, $\mathbf{PNR_{diff}}$)**: In this case, we aim to minimize the difference between the probabilities that an individual is predicted as negative ($P[p = 0]$), conditioned on whether the individual takes one value or another in the sensitive attribute, i.e., $P[p = 0|A = 0]$ and $P[p = 0|A = 1]$. Similar to before, we are interested in the absolute difference between the two: $|P[p = 0|A = 0] - P[p = 0|A = 1]|$. This measure corresponds to the demographic parity family. Possible values for this measure belong to the range $[0, 1]$.

- **Interpretability objectives** (not suitable for FairLGBM): We will aim for our system to be as simple as possible, as this greatly enhances its interpretability and understanding. This will allow users to comprehend it, enabling them to make decisions based on the method's outcome consciously and responsibly.
  - **Number of leaves**: This is a measure that indicates the complexity and interpretability of the model quite well. The higher the number of leaves, the greater the number of decision paths that can be taken when classifying an individual, making the interpretation of each of these paths, or the tree as a whole, much more complex.
  - **Weighted average leaf depth based on the individuals falling into each leaf**: A higher depth for a particular leaf implies that it has passed through a greater number of intermediate decision processes (intermediate nodes), making the interpretation of that decision path more complicated. However, we are interested in knowing how many instances (during training or validation) fell into that leaf, as this will give us a relative importance of that leaf. If the majority of examples fell into shallow leaves, and only exceptions fell into very deep leaves, the complexity in decision-making for a new individual is likely to be lower than if the opposite were true.

Once all the implemented objectives have been reviewed, it is important to note several nuances. The implemented code is prepared to use any subset of them for our executions, allowing for partial optimization studies or attempting a total optimization with all objectives. Due to all the aforementioned problems that inherently come with increased dimensionality, it can be very interesting from a more efficient optimization perspective to restrict the number of objectives that are deemed necessary or that the user considers at each moment.

9.5. Evaluation of models and runs

In order to evaluate the results obtained from the optimization process, a validation and testing system will be devised. Our dataset will be partitioned for testing, with 75% of the data randomly selected for training and the remaining 25% for testing. This data partition ratio is a standard practice. For validation, a partition with the same proportion as for testing will be performed, so that 75% of the training data will be exclusively used for training, while 25% will be used for validating the learned models. Therefore, the actual distribution will be 56.25% of the data for training, 18.75% for validation, and 25% of the total data for testing. This partition will be conducted using a random seed to ensure reproducibility of

the experimentation. These seeds will also control other random processes of the algorithms.

The use of a random seed could potentially bias the results: it is possible that for a specific seed, the dataset partition may have a specific structure and the results may take a particular form, which may not be the case if another seed had been chosen. That's why 10 executions will be performed for each algorithm and dataset with different random seeds (ranging from 100 to 109).

Each of these runs will generate a set of Pareto-optimal solutions for the specified algorithm and dataset. These solutions are Pareto-optimal with respect to the objective function evaluated on the validation set, since throughout the genetic optimization process, only the training data (for training models) and validation data (for evaluating models) are accessible. Once this Pareto-optimal set is obtained with respect to the validation set, the considered models will be tested using the test set.

Once all these algorithms obtained from the 10 runs for each dataset are evaluated with the test set (respective to each partition), all the solutions will be considered together, and the Pareto-optimal algorithms will be selected using the test results. By using this procedure, we first find the best algorithms during the search process using the validation information, and once we have those algorithms, we select the best ones using the information from the test set.

Additionally, quality measures will be calculated on the set of Pareto-optimal solutions itself, so that we can compare the solutions obtained by various algorithms for the same dataset. Other quality measures will be calculated for each generation of each run, such as the time taken to perform the run, and other general measures on the structure of the algorithms found in each generation.

10. Implementation details

In this chapter, various aspects of experimentation at the implementation and deployment levels will be discussed. Details will be provided on the software developed, as well as the hardware used for testing the methods.

10.1. Software specifications for executions

In this section, the structure of the software developed for carrying out the executions will be explained. Specifically, the general structure and contents of the folders and files created will be explained.

The code in its entirety is included in the following repository for public access: `https://github.com/Daalma7/FairTreesAlgorithms`. We will now proceed to explain its main contents.

The main structure of the repository can be seen in Figure 10.1. The contents of the items shown in the figure can be seen below:

- **GeneticPruning**: Folder containing the code related to the algorithm with the same name.
- **HyperparameterOptimization**: Folder containing all the code related to the hyperparameter optimization process for the DT, FairDT, and FairLGBM algorithms using NSGA-II.
- **exec_scripts**: Folder containing the execution scripts of the methods for the conducted experimentation.
- **README.md**: File with a simple description of the project.
- **environment.txt**: File explaining how to set up an environment for running the code, installing necessary packages.

The contents of the GeneticPruning folder can be seen in Figure 10.2. The contents of the files shown in the figure can be seen below:

- **genetic.py**: File that defines the entire genetic optimization process, including all selection, crossover, mutation, and replacement criteria.
- **individual.py**: File that defines 2 very important classes: on one hand, the structure of the matrix tree, and on the other hand, an individual itself, consisting of a set of prunings on the matrix tree. All methods for representation and calculation of metrics on individuals are implemented here.

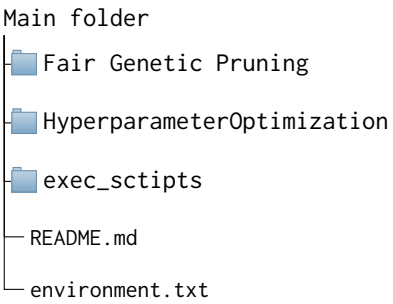


Figure 10.1.: Main folder of the developed code.

- **main.py**: Main file that controls the execution of the method, accepting parameters such as the training set or the number of generations and population size.
- **ml.py**: File containing all logic related to machine learning. Training of individuals, validation, or saving information about the algorithm's performance and results.
- **test.py**: File for testing.

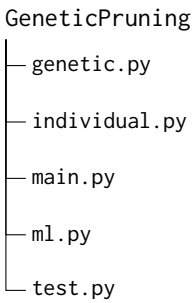


Figure 10.2.: Contents of folder GeneticPruning.

The main contents of the HyperparameterOptimization folder can be seen in Figure 10.3. The contents of the files shown in the figure can be seen below:

- **algorithms**: Folder where different multi-objective optimization algorithms are defined.
  - **nsga2**: Folder containing the code of the NSGA-II algorithm.
  - **evolution.py**: Main file where the main evolutionary process of NSGA-II is defined.
  - **utils.py**: File defining all functions related to NSGA-II, such as initial population creation, mutation, crossover...
- **bin**: Folder with different execution scripts
  - **main.py**: File that controls the main execution of the multi-objective optimization process, with the algorithm and parameters specified.

- **general**: Folder containing general classes and utilities.
  - **individual.py**: File containing the definition of individual classes that will be used in the optimization process.
  - **ml.py**: File containing the definition of all machine learning processes, objective calculation, and model hyperparameter control.
  - **population.py**: File containing the definition of population classes of individuals for the optimization process.
  - **problem.py**: File containing the class that defines an optimization problem, as well as various general utilities. Centralizes the generation of individuals and calculation of objectives.
- **models**: Folder containing the definition of the two models created to run in this optimization process: FairLGBM and FairDT.
  - **FLGBM (FLGBM.py)**: Definition of the FairLGBM model. It is defined in a single file.
  - **FairDT**: Folder containing the definition of the FairDT model. It actually extends the DecisionTreeClassifier model from the scikit-learn library, adding the relevant fairness modifications. It has a package structure.

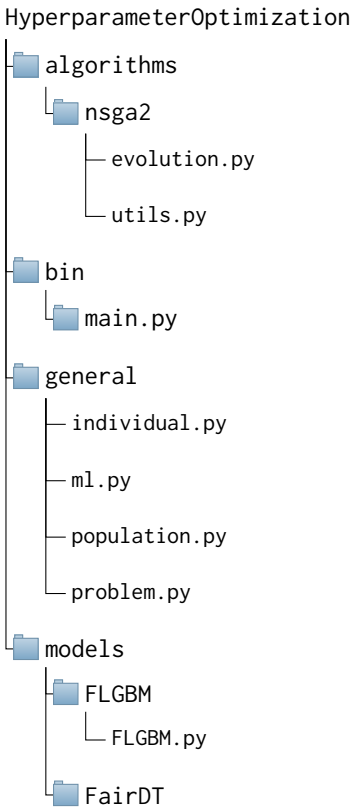


Figure 10.3.: Contents of folder HyperparameterOptimization.

The main contents of the exec_scripts folder can be seen in Figure 10.4. The contents of the files shown in the figure can be seen below:

- **CalculateMeasures**: Folder containing the code related to measure calculations and plots.
  - **calculatemeasures_aux.py**: Functions for measure calculations, plots, and total Pareto-optimal sets (for a dataset using all available models).
  - **main.py**: Main file for executing measure calculations and plots.
  - **qualitymeasures.py**: File containing the definition of quality measures on Pareto-optimal solutions, as defined in the section Quality metrics over a solution set FALTA.
- **exescript_x**: Script that controls the executions indicated by those ending in DT, FDT, FLGBM, and FGP, managing the execution of the corresponding algorithms. The one ending in Measures controls the execution of the measure calculation file.

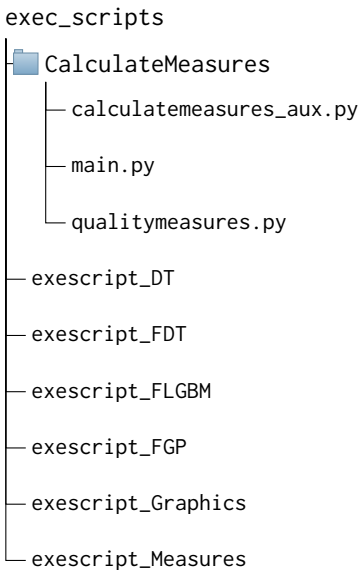


Figure 10.4.: Contents of folder GeneticPruning.

10.2. Implementation details for algorithms

In this section, some details about the implementation that are believed to be relevant we will discussed:

- **Parallelism**: The creation of individuals in each population has been implemented using CPU-level parallelism utilizing Python's joblib module, and its Parallel and delayed methods. Both in the creation of the initial population and in the crossover and training of offspring individuals, parallel processing will be performed. This can

be done perfectly, since the specification of parameters (either randomly in the initial population, or through selection and crossover in the offspring populations) and the training of the considered individuals are completely independent. This makes the entire process much more efficient, as it avoids unnecessary sequential calculations, resulting in a very noticeable improvement in execution time.

- **No validation repetition**: Suppose the crossover probability is not 1, and therefore the same parent individuals are returned to the offspring population. In that case, the individuals will not be retrained, also saving computation time.
- **Saving individuals when necessary**: The individuals generated throughout the optimization process will be stored in result files. Storing these individuals, for example, after each generation, could be very costly, as writing and/or reading result files could create a bottleneck in the program. Therefore, it is considered a criterion to write individuals only at the end of the hyperparameter optimization process (after execution). Files will be saved for all individuals that have been considered, as well as for the Pareto optimal set.

Next, we will discuss details about the implementation of each algorithm:

10.2.1. Implementation of FairDT algorithm

For the implementation, the version of the DecisionTreeClassifier algorithm from the scikit-learn library (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) was modified. To achieve this, the code available on GitHub was downloaded (https://github.com/scikit-learn/scikit-learn/tree/f07e0138bfee41cd2c0a5d0251dc3fe03e6e1084/sklearn/tree, accessed: April 2022), and relevant files were modified, primarily the Python and Cython files such as _criterion.pyx, .pxd, _classes.py, and _tree.pyx. For successful compilation, it is necessary to install the cython package in a version prior to 3.x.x (version 0.29.37 was used in this case). There is a file named build.sh within the folder containing the developed FairDT code, which contains the necessary code to execute in order to compile the algorithm package and make it usable.

The result is the new DecisionTreeClassifier class, which functions like the scikit-learn version but with some modifications. It includes the fairness criteria gini_fair and entropy_fair, in addition to the gini and entropy criteria. These new criteria allow for the use of the fairness parameter and function; if gini or entropy are selected, these fairness criteria will not be used. Then, there is the parameter f_lambda, which is the combination parameter specified in the previous section. Lastly, the fair_fun parameter allows us to select the fairness function to apply. By default, it is set to fpr_diff, but it can also be any of the other objectives specified in the objectives section (fpr_diff, ppv_diff, pnr_diff). Additionally, two more were implemented: tpr_diff and tnr_diff.

10.2.2. Implementation of Fair Genetic Pruning algorithm

The implementation of these methods has been entirely done using the Python language. Classes have been created to represent the matrix tree, each individual (a set of prunings over the matrix tree/subtree sharing the root node), and the genetic optimization process itself.

Thanks to the different representations implemented and discussed during the algorithm explanation (section FALTA), the method's efficiency is enhanced. To represent the matrix tree, classes were created to allow faster access to key information. For instance, dictionaries identifying the number of instances divided by class and protected attribute falling into each node, the total number of instances, and an array containing the representation of the initial leaves of this matrix tree were implemented. This stored information scales linearly with the size of the matrix tree (in terms of the number of nodes), and the efficiency improvement for methods affected by the storage of this information is quite noticeable, as it avoids performing many repeated calculations. In the case of methods using dictionaries, efficiency improves to $\mathcal{O}(1)$, while for those using the array, efficiency becomes $\mathcal{O}(n)$, where n is the length of the array.

The DecisionTreeClassifier class from the scikit-learn library has been utilized in order to learn the decision tree. The tree definition within the library, along with its methods within the structure itself, have been leveraged to access the required information. The implementation of the NSGA-II method is entirely analogous to that done for the hyperparameter optimization methods.

10.2.3. Implementation of FairLGBM algorithm

For the implementation, the code from the LightGBM library (`https://lightgbm.readthedocs.io/en/stable/`). was used. To achieve this, the functionality of this code was included within a class called FairLGBM, which contains the basic functionalities that this algorithm can work with. In this code, the loss function is modified, as well as other functions.

To modify the loss function, it is necessary to specify the *objective* parameter in the LightGBM parameter list, assigning it a defined function. This function should accept two parameters: an array with predictions and a LightGBM dataset containing the information to learn from. The function should return the gradient and Hessian of the loss function. This has been implemented in the bce_fair_loss function. In addition to that, a bce_fair_eval function has been implemented. This function serves for calculating the loss function itself, in processes that, for example, include early stopping or validation sets.

In our case, we will indeed use early stopping with the validation set to obtain more accurate models and save many training steps. This fact is crucial when performing hyperparameter optimization processes, since the same base algorithm is trained many times. However, training can also be done without specifying these sets.

10.3. Hardware specifications for executions

In this section, the hardware used for the experimentation will be explained. The technical specifications of the equipment used for the experimentation are as follows:

- **Case**: Corsair Graphite 760T Black Full Tower.
- **Power supply**: 1000W ATX.
- **Motherboard**: ROG Corsair VIII Hero Socket AM4.

- **Processor**: Ryzen 9 5950X 3.4GHz.
- **Cooling**: Corsair Hydro H100X.
- **RAM**: 2 x 16GB DDR4 3200MHz PC4-25600 - CL16 - 1.35V.
- **SSD**: 1TB 2.5″.
- **Graphics Card**: Asus TUF RTX 3080 10GB.

Part V.

Results and Conclusions

In this part, all the results obtained through the experimentation will be showcased and discussed. Conclusions of the project as a whole will be drawn.

11. Results

In this chapter, the results obtained during the experimentation will be presented. The generated graphs and diagrams will be explained, along with their interpretation.

11.1. Análisis del efecto del parámetro λ sobre el algoritmo FDT

En primer lugar, vamos a realizar un pequeño estudio sobre el efecto del parámetro λ sobre la estructura de los árboles de decisión para el modelo FDT. En las figuras 11.1 , 11.2, 11.3 y 11.4 se pueden ver distintos valores sobre características estructurales de los árboles FDT en función del parámetro λ. En particular, para cada conjunto de datos y semilla aleatoria (que controla la partición de datos) se han entrenado FDTs con el conjunto de entrenamiento sin restricciones en términos de tamaño, pero con diferentes valores del parámetro λ, variando entre 0 y 1, en intervalos constantes de 0.01 (un total de 101 valores distintos de λ). Fijando el conjunto de datos y el valor de λ, los resultados obtenidos para cada partición de datos se han resumido, mostrando el valor medio de las ejecuciones como un punto, y una barra mostrando un intervalo de confianza centrado en la dicho punto al 95% de confianza de que los valores de la medida estén en dicho intervalo. En la figura 11.1 se pueden ver los resultados de precisión en en entrenamiento y test, en la figura 11.2 se puede observar información análoga pero para la medida FPR_{diff}, en la figura 11.3 se puede ver la profundindad de los árboles entrenados, y en la figura 11.4 se puede ver el número de hojas de estos árboles.

El efecto del parámetro λ que se puede ver en estas figuras es claro. En la figura 11.1 se puede ver cómo a medida que se aumenta el valor del parámetro λ, disminuye la precisión. Esta reducción no se realiza de manera constante, y se puede ver cómo para valores relativamente bajos de λ (en torno al 0.1) se mejoran un poco los resultados en test. En general empieza una reducción abrupta a partir del valor 0.6.

Los valores que se muestran en la figura 11.1 son muy llamativos. Se puede ver cómo a medida que aumenta el valor de λ, al principio se empeora el valor de FPR_{diff}, lo cual no es esperado. Sin embargo este fenómeno no ocurre de manera no constante, existen ciertos valores en los que puntualmente mejora (que son justo los que queremos encontrar). En torno al valor 0.6, pega un salto muy pronunciado, reduciéndose igualmente de manera abrupta en torno al valor 0.8, habiendo algunos casos en los que se reduce muchísimo su valor, obteniendo aquí sí valores inferiores a todos los anteriores. Sin embargo, por lo que vimos en la figura 11.1, también se produce una reducción considerable en la precisión, tal y como se esperaba.

Por último, en las figuras 11.3 y 11.4 se observan tendencias análogas, pero tremendamente influyentes en nuestra experimentación. Conforme aumenta el valor de λ, los árboles encontrados son más simples (menos profundidad y hojas). A partir del valor 0.8, los árboles

son extremadamente simples, no deseados normalmente. Este hecho, en un proceso de optimización multiobjetivo es indeseable, puesto que independientemente del resto de parámetros (que normalmente limitan el crecimiento del árbol), si el valor de λ es bajo, los árboles que se entrenarán serán muy simples. Pero como el valor objetivo en FPR_{diff} son los más óptimos, aparecerán siempre como individuos no dominados, y una gran combinación de hiperparámetros (espacio de decisión) dará lugar a los mismos tipos de árboles, no dominados. En una experimentación inicial realizada, se vio cómo el algoritmo tendría a acumular este tipo de árboles. Nuestro objetivo es no sólo encontrar estos árboles, sino una exploración más completa del espacio objetivo, por lo que se limitará la cantidad de soluciones con los mismos valores objetivo que habrá en cada generación, tal y como se explica en la sección **??** FALTA. Este efecto también se ha observado en el algoritmo FLGBM.

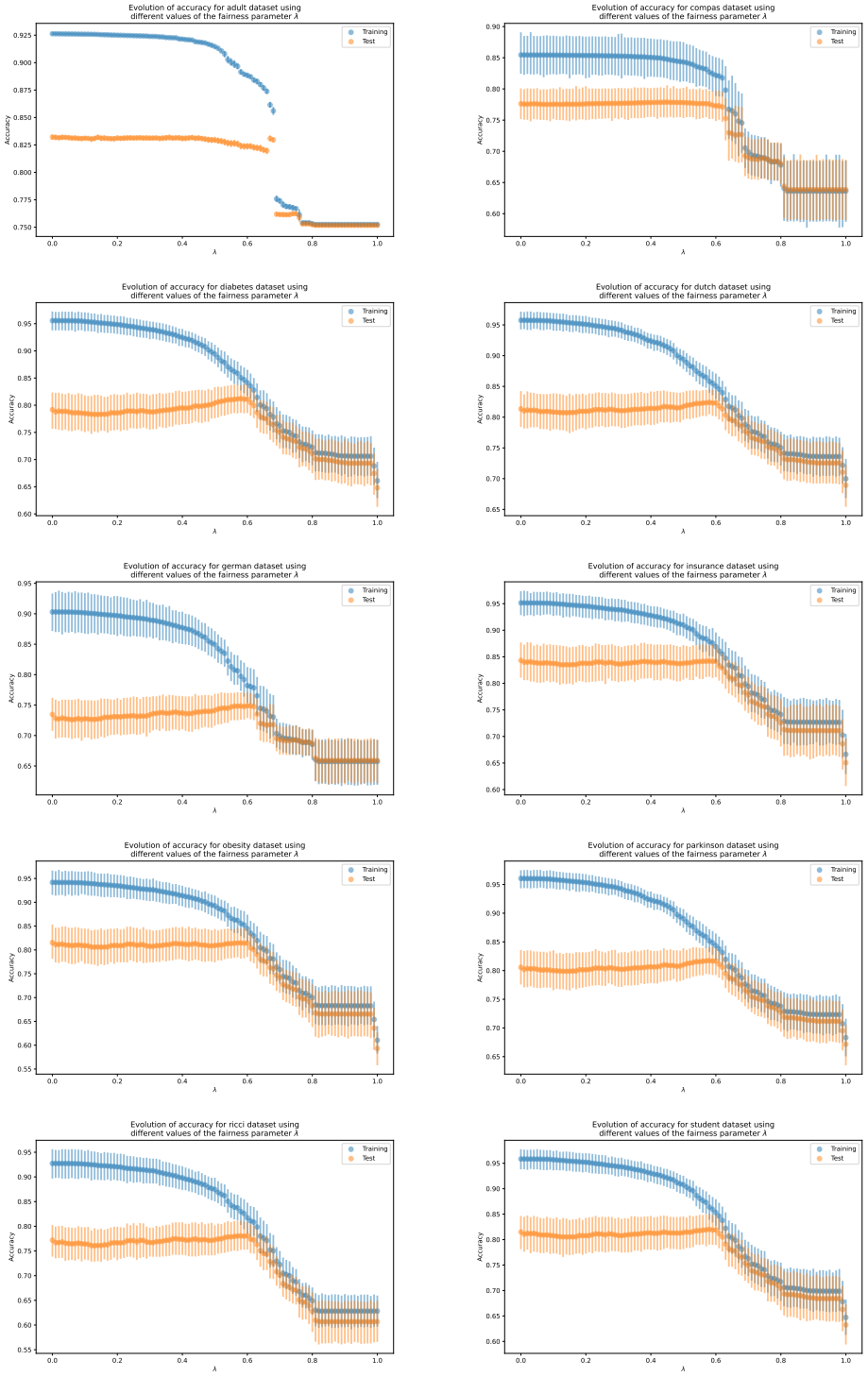


Figure 11.1.: Evolution of accuracy varying the fairness parameter

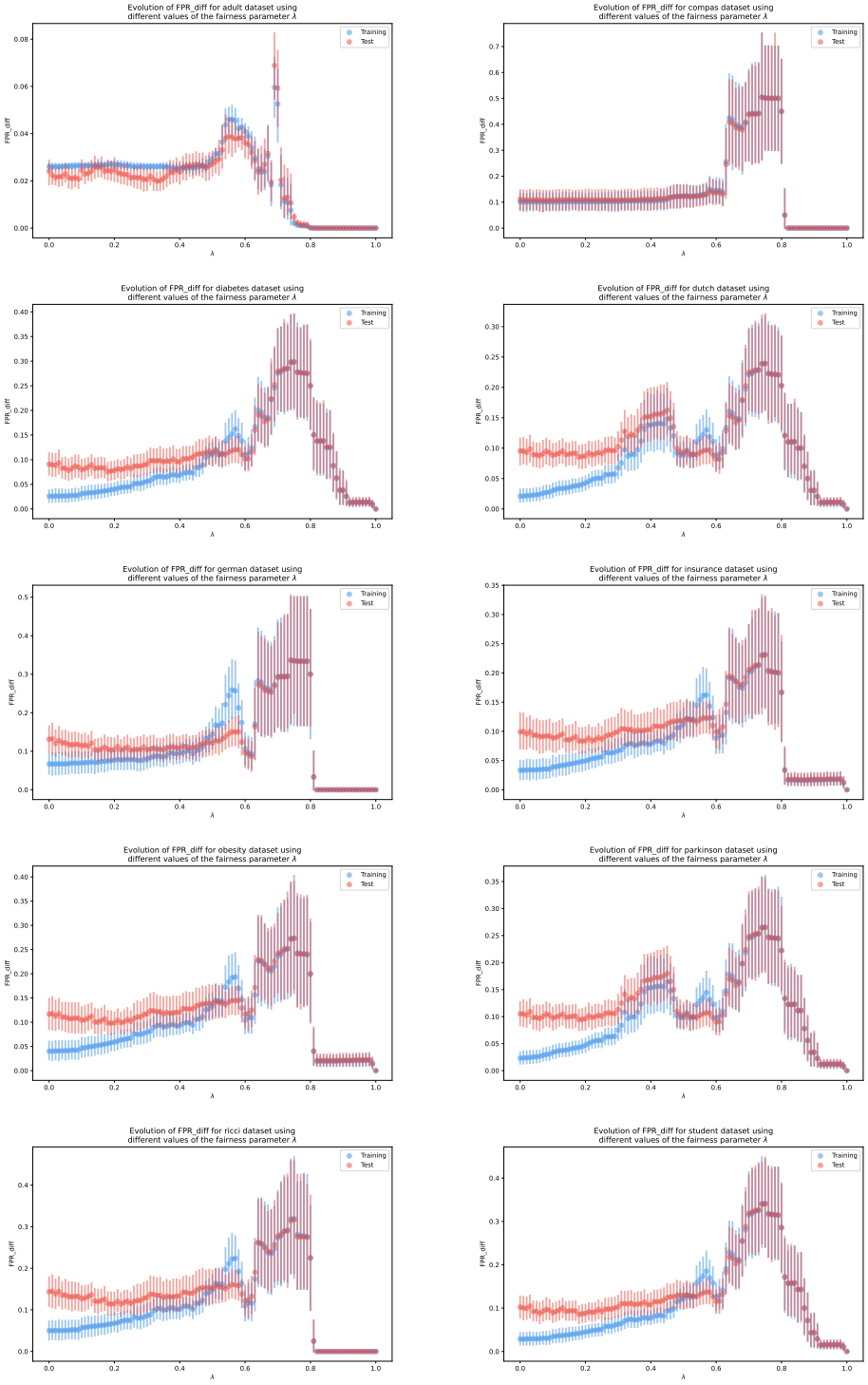


Figure 11.2.: Evolution of FPR_{diff} varying the fairness parameter

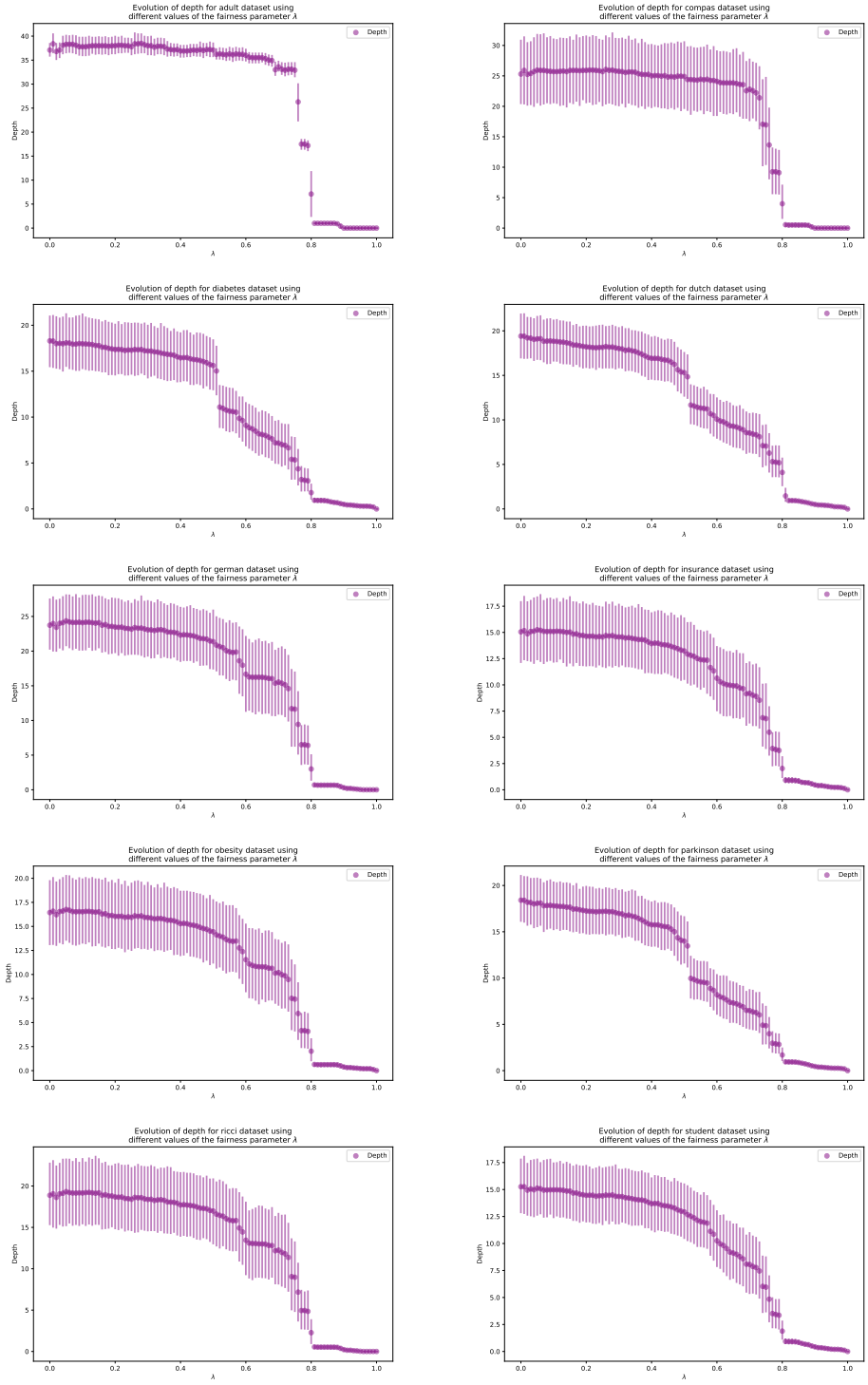


Figure 11.3.: Evolution of depth varying the fairness parameter

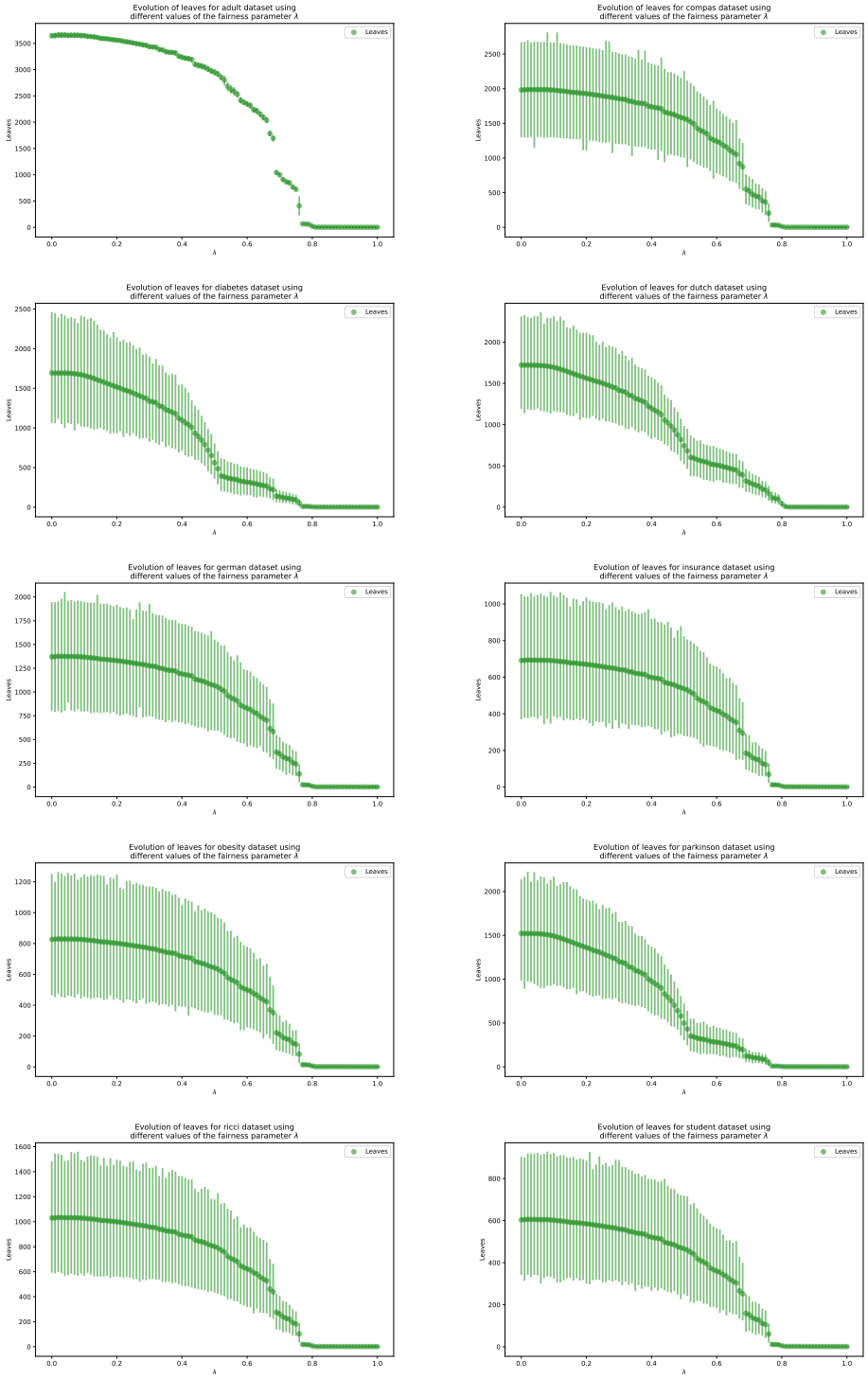


Figure 11.4.: Evolution of the number of leaves varying the fairness parameter

11.2. Gráficas y tablas comparativas de resultados de la experimentación

We will continue by presenting general results regarding the execution of the algorithms. From Figure 11.5 to Figure 11.34, various result graphs for each dataset can be found. We will explain the content of each of these graphs:

- **Pareto fronts**: Figures 11.5, 11.8, 11.11, 11.14, 11.17, 11.20, 11.23, 11.26, 11.29 and 11.32 contain information about the Pareto-optimal sets of solutions.
  - **Scatterplot showing average Pareto fronts by algorithm**: The top-left graph shows a scatterplot where the Pareto front with the test results found by each execution of each algorithm is plotted using a soft tone. With them, an average Pareto-optimal set has been created for each algorithm. The number of individuals in this average set will be the average of the number of Pareto-optimal individuals found in each execution. On the other hand, the values in each objective will be calculated using percentiles. Having two objectives o_1, o_2, and a population of n individuals, then the objective values of these individuals will be $\left\{\left(p_{o_0}\left(\frac{i}{n-1}\right), p_{o_1}\left(1-\frac{i}{n-1}\right)\right), i \in \{0,\ldots,n-1\}\right\}$, where the functions p_{o_0} and p_{o_1} are the percentiles functions of each objective calculated over the population of Pareto-optimal solutions from all executions for that algorithm and dataset. In this way, “average" values are established for all executions. These average values appear in a stronger tone. Additionally, to indicate the approximate shape of the Pareto front, a Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) has been used, producing a continuous, smooth, and generally good approximation for this specific case.
  - **Scatterplot showing general average Pareto front**: The top-right graph is a scatterplot that uses the results from the left graph to select the Pareto front from all values of the average Pareto front of each algorithm. Additionally, the remaining values of the Pareto front of each algorithm that do not belong to the general Pareto front are shown in a softer tone. The same Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) cubic monotonic interpolation is used.
  - **Pieplot showing proportion of individuals**: The bottom-left graph corresponds to the proportion of individuals per algorithm that are found in the final Pareto-optimal set. It is important to note that these individuals will be the average generated individuals.
  - **Violinplot showing the ratio between test and validation results**: The bottom-center graph shows the ratio between test and validation objectives for the Pareto-optimal solutions. These ratios will be calculated over all real Pareto-optimal individuals found in each execution. As the results are in the range $[0, 1]$, and the lower they are, the better, if the violinplot takes values greater than 1, that means there is overfit for that specific objective and algorithm. If the values are less than 1, test results are better than validation ones.
  - **Coverage heatmap**: The bottom-right graph shows a ranking of coverage between the average Pareto fronts created for each algorithm. A high value in a position is better for the algorithm of the row and worse for the algorithm of the column.

- **Structure of individuals across generations**: Figures 11.6, 11.9, 11.12, 11.15, 11.18, 11.21, 11.24, 11.26, 11.30 and 11.33 contain information about different metrics across generations for each algorithm. The metrics presented are relevant metrics regarding the structure of the models comprising the generation. The results shown represent the average per generation using information from all executions. For each measure, the maximum, mean, minimum, and standard deviation values are displayed in a shaded area. The measures presented for each algorithm are as follows:
  - **FairDT and FairLGBM**: Number of leaves, depth, depth weighted by the number of instances that fall into each leave, and unbalance (calculated as depth of the shallower leaf divided by the depth of the deepest leaf)
  - **Fair Genetic Pruning**: Same values as above, as well as the amount of prunings performed.

  Using this graph, we can verify whether the general structure of the models learned by the optimization process converges or not, and if it does, how quickly it does so, along with the nature of that structure.
- **Rankings of quality metrics**: Figures 11.7, 11.10, 11.13, 11.16, 11.19, 11.22, 11.25, 11.28, 11.31 and 11.34 show a comparison between the quality measures on the sets of Pareto-optimal solutions found by each algorithm. These measures are calculated over the test set. Additionally, it is indicated whether a low or high value is better for each measure. This graphs are very informative in order to compare the performance of the algorithms for these datasets.

Additionally, Figure 11.35 shows a graph with a ranking performed on each of the quality measures. In particular, if an algorithm has achieved the best score among the 4 for a specific quality measure and dataset, it will receive 3 points. If it achieved the second place, it will receive 2 points, if third, 1 point, and if it ranked the last, it will receive no points. The ranking has been conducted by summing up all the points obtained for each dataset.

Results for Adult dataset

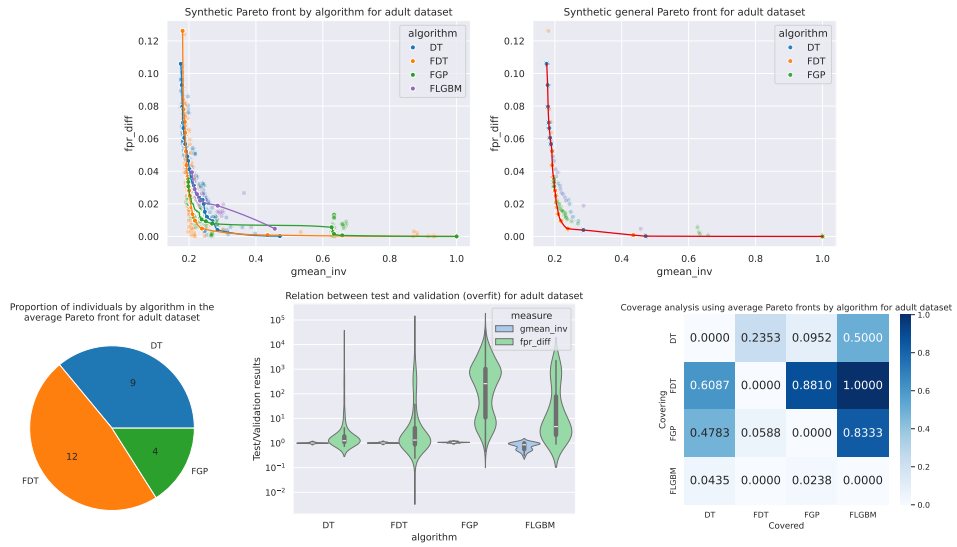


Figure 11.5.: General metrics for Pareto optimal solutions for adult dataset.

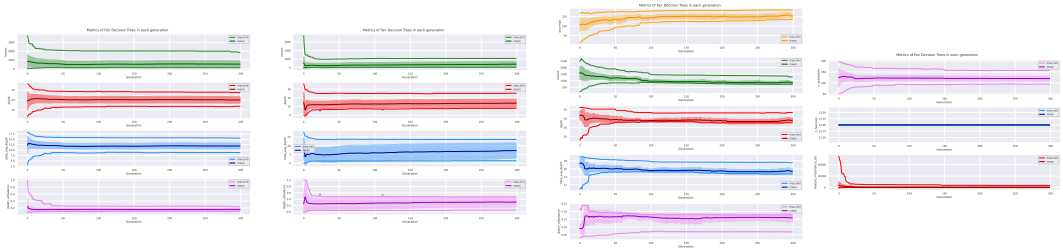


Figure 11.6.: Evolution of individuals dimensions through generations for adult dataset.

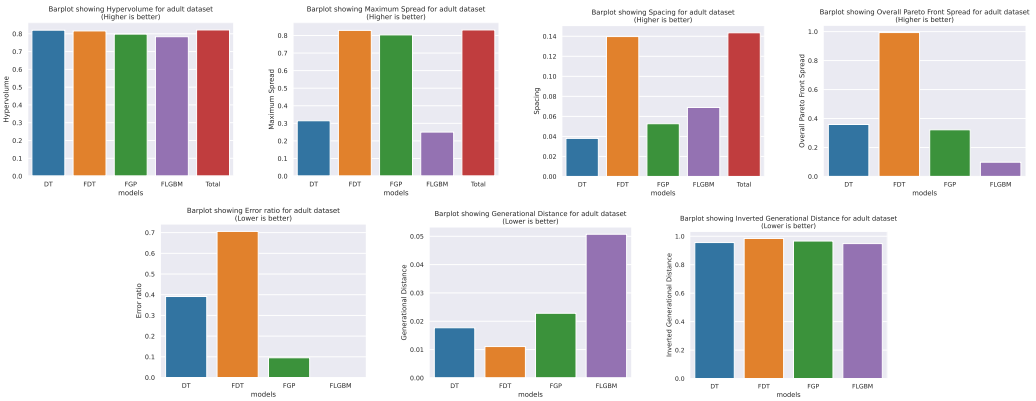


Figure 11.7.: Pareto front quality metrics for adult dataset.

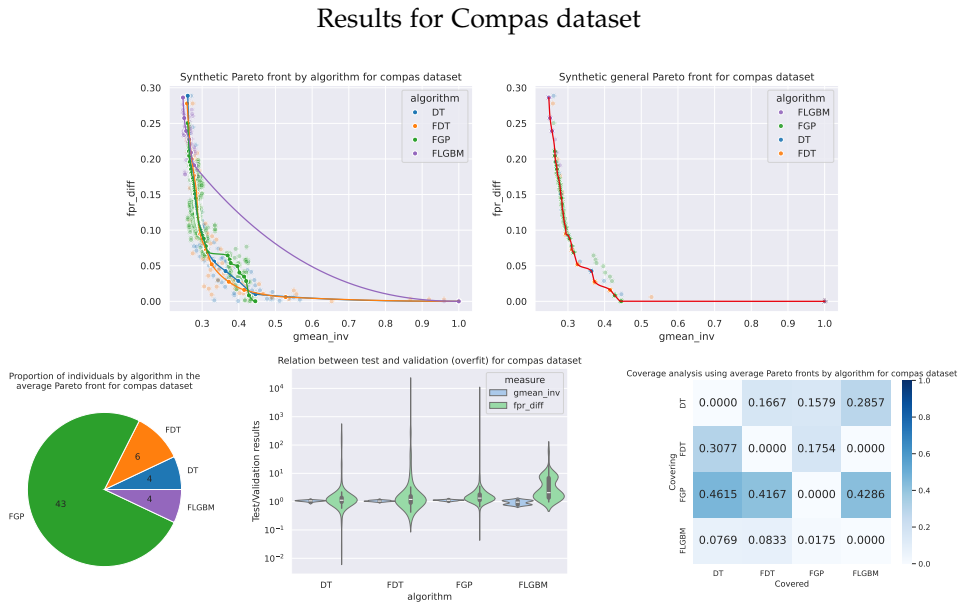


Figure 11.8.: General metrics for Pareto optimal solutions for compas dataset.

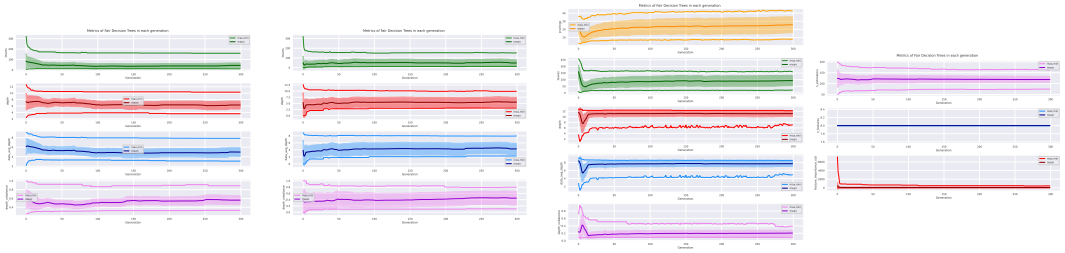


Figure 11.9.: Evolution of individuals dimensions through generations for compas dataset.

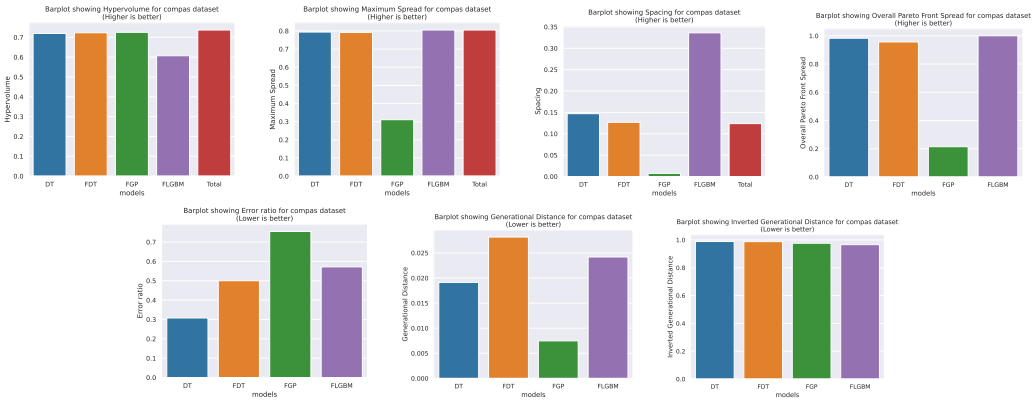


Figure 11.10.: Pareto front quality metrics for compas dataset.

Results for Diabetes dataset

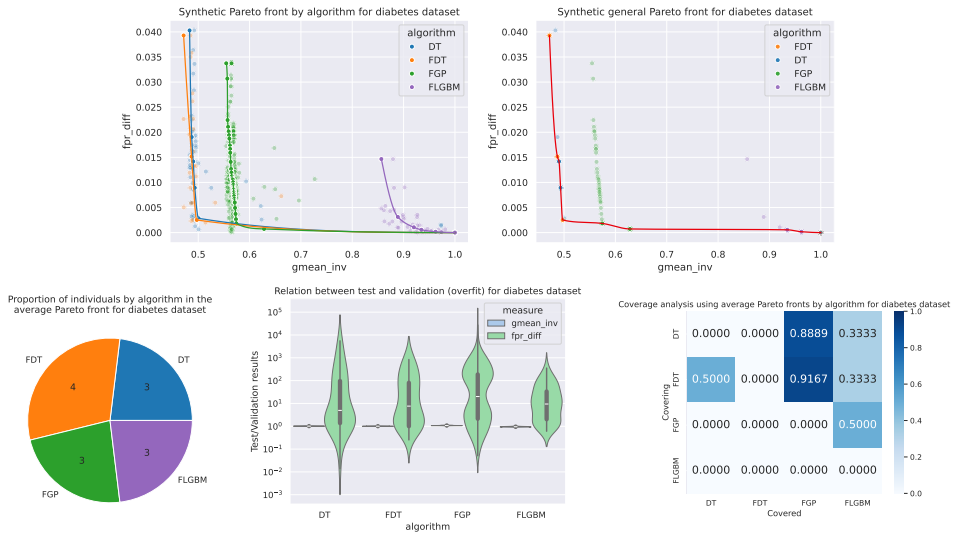


Figure 11.11.: General metrics for Pareto optimal solutions for diabetes dataset.

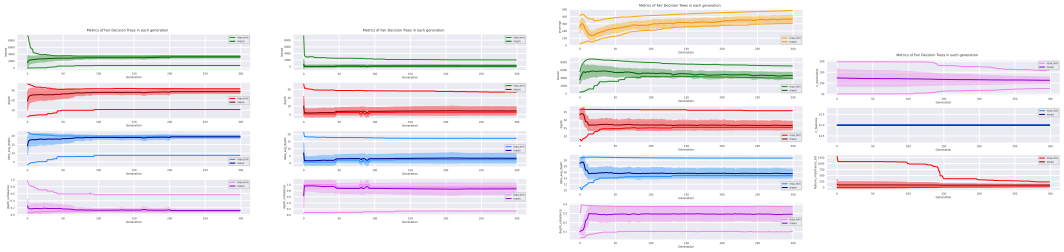


Figure 11.12.: Evolution of individuals dimensions through generations for diabetes dataset.

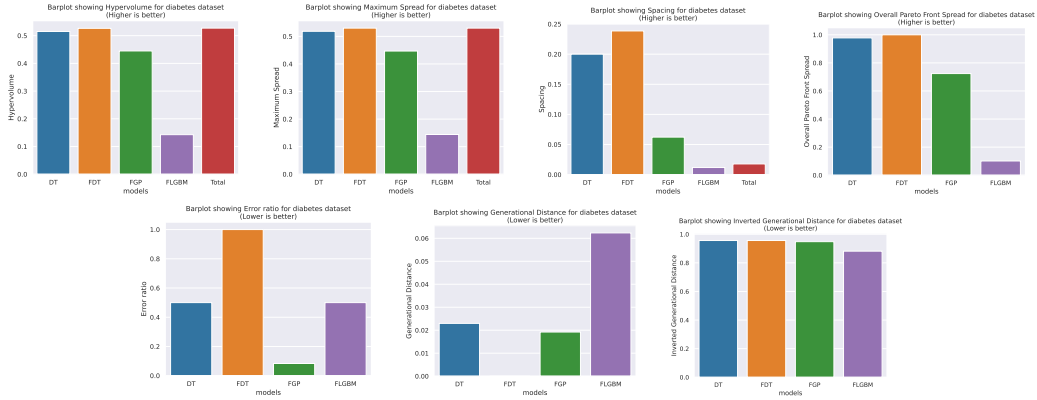


Figure 11.13.: Pareto front quality metrics for diabetes dataset.

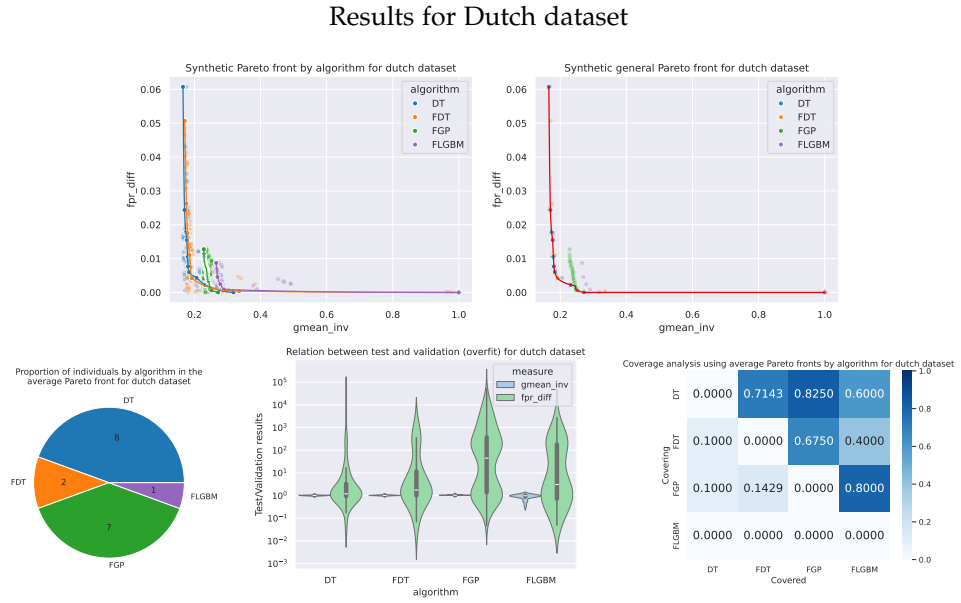


Figure 11.14.: General metrics for Pareto optimal solutions for dutch dataset.

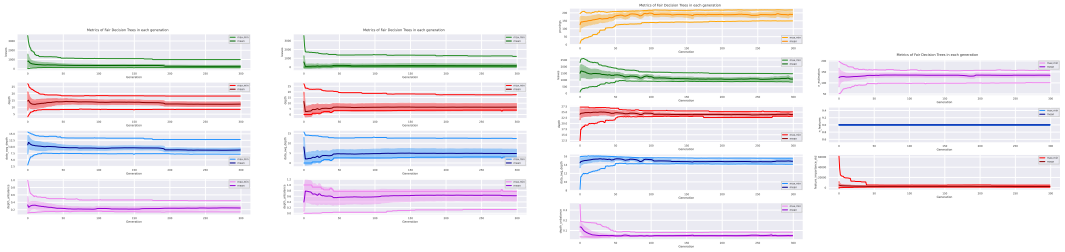


Figure 11.15.: Evolution of individuals dimensions through generations for dutch dataset.

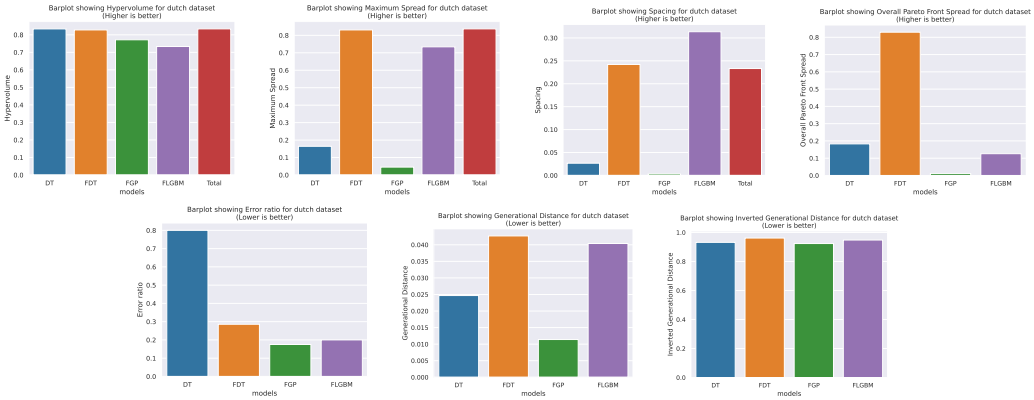


Figure 11.16.: Pareto front quality metrics for dutch dataset.

Results for German dataset

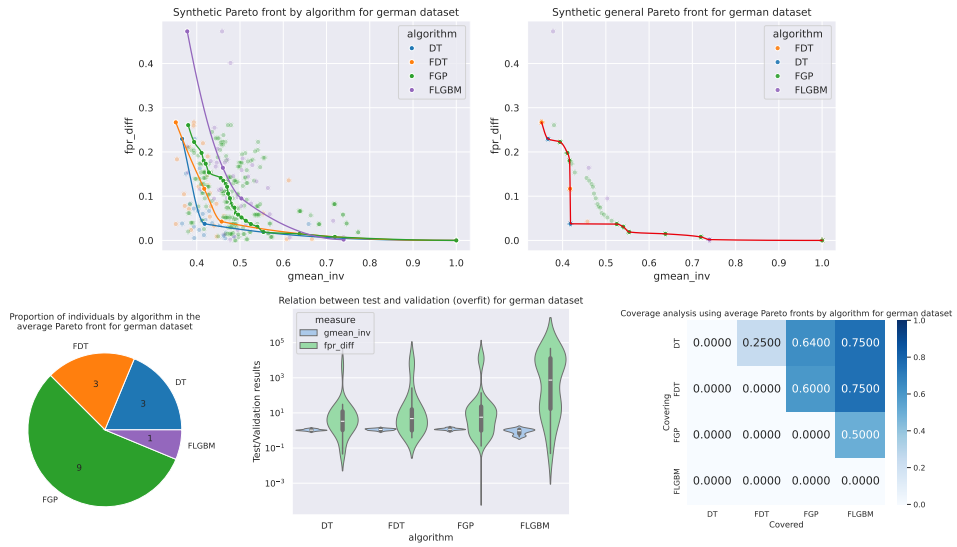


Figure 11.17.: General metrics for Pareto optimal solutions for german dataset.

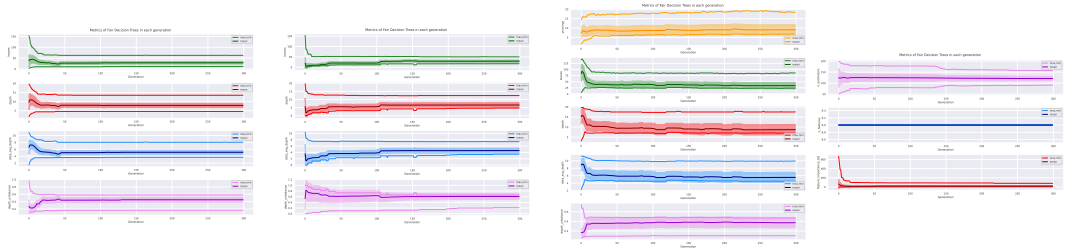


Figure 11.18.: Evolution of individuals dimensions through generations for german dataset.

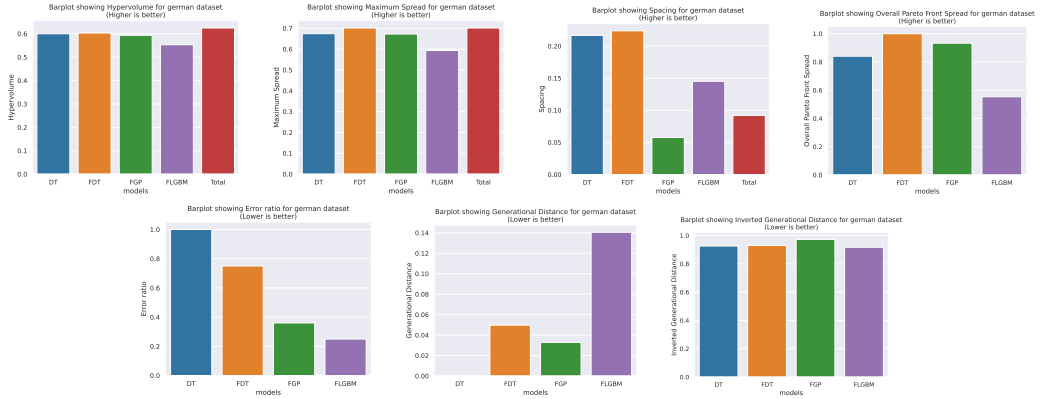


Figure 11.19.: Pareto front quality metrics for german dataset.

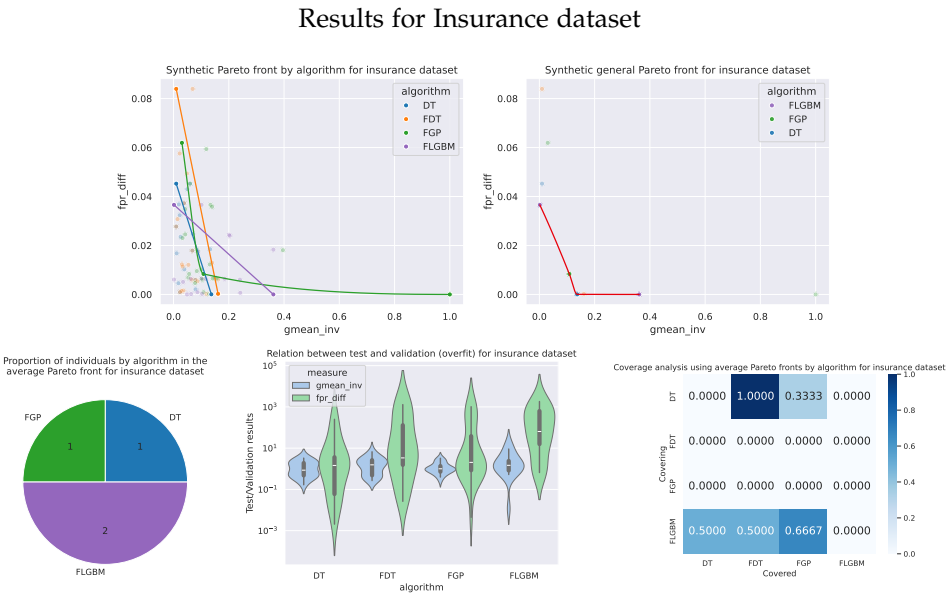


Figure 11.20.: General metrics for Pareto optimal solutions for insurance dataset.

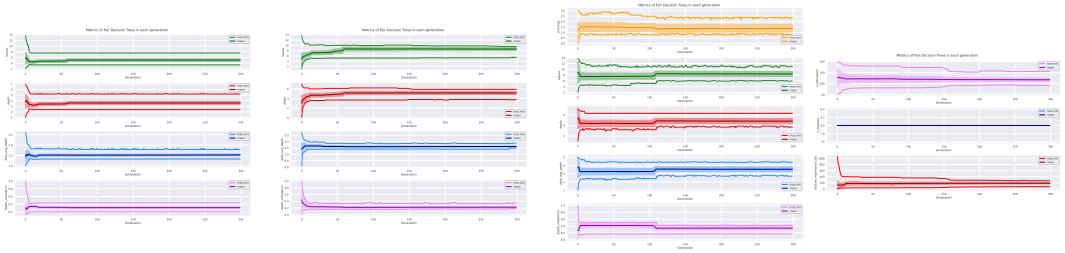


Figure 11.21.: Evolution of individuals dimensions through generations for insurance dataset.

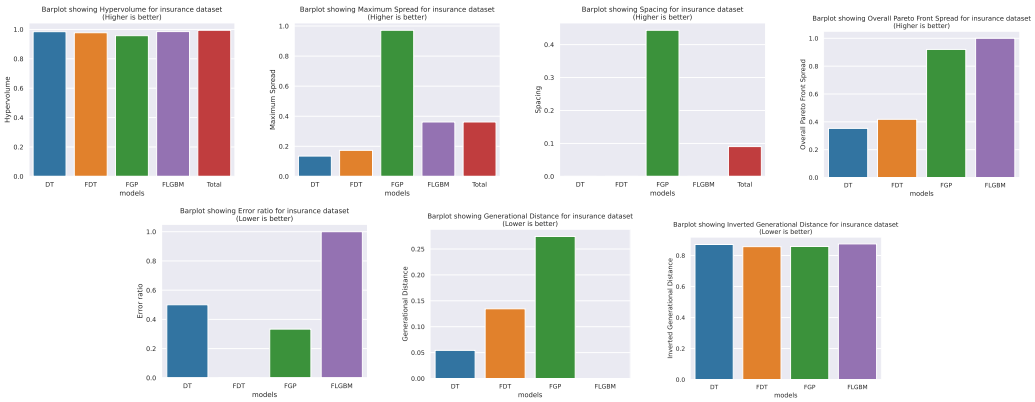


Figure 11.22.: Pareto front quality metrics for insurance dataset.

Results for Obesity dataset

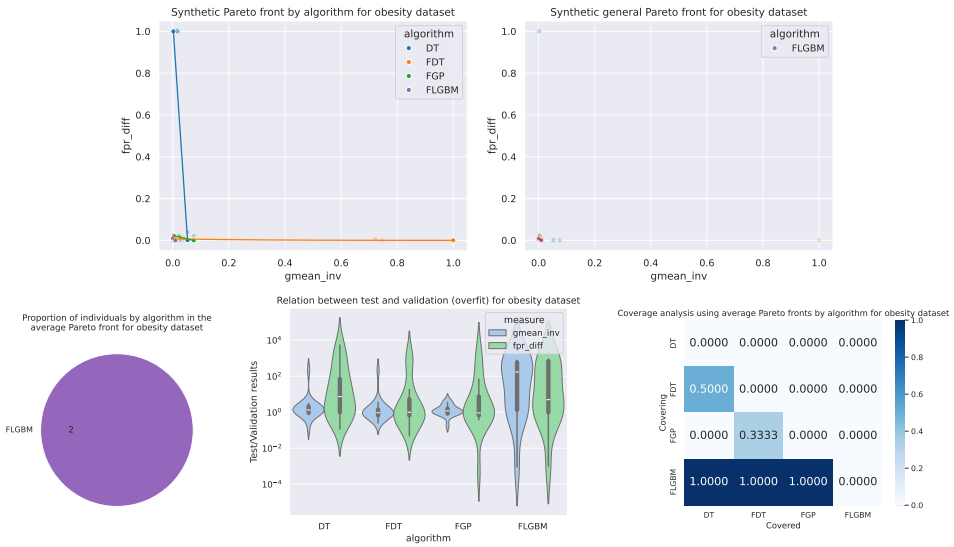


Figure 11.23.: General metrics for Pareto optimal solutions for obesity dataset.

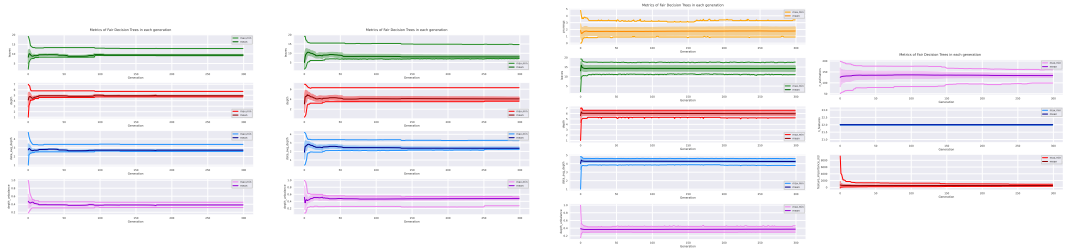


Figure 11.24.: Evolution of individuals dimensions through generations for obesity dataset.

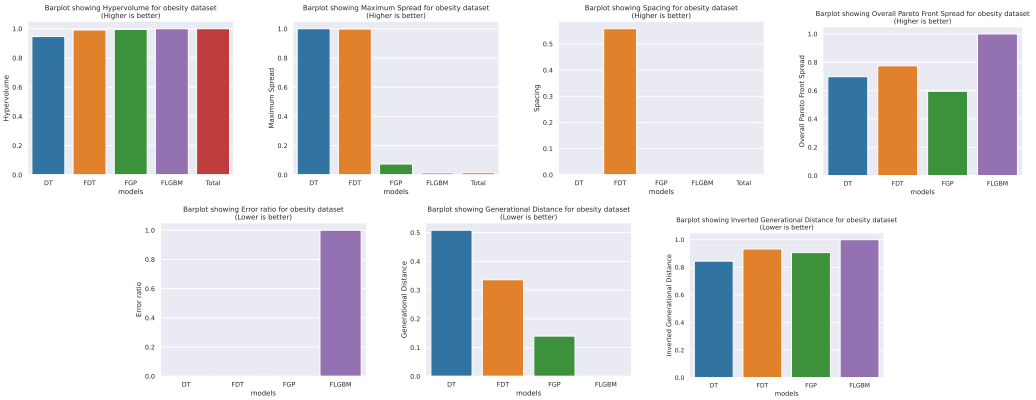


Figure 11.25.: Pareto front quality metrics for obesity dataset.

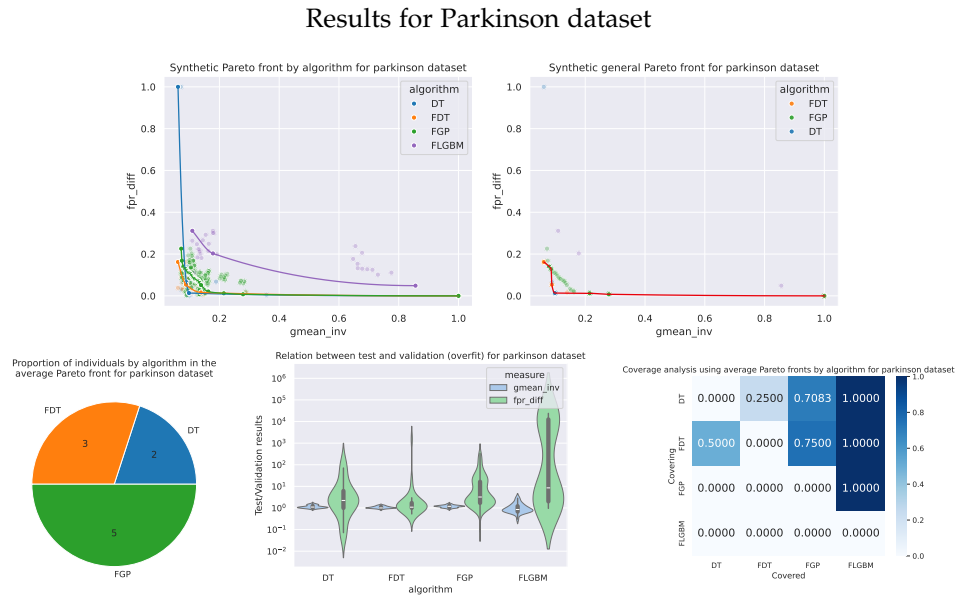


Figure 11.26.: General metrics for Pareto optimal solutions for parkinson dataset.

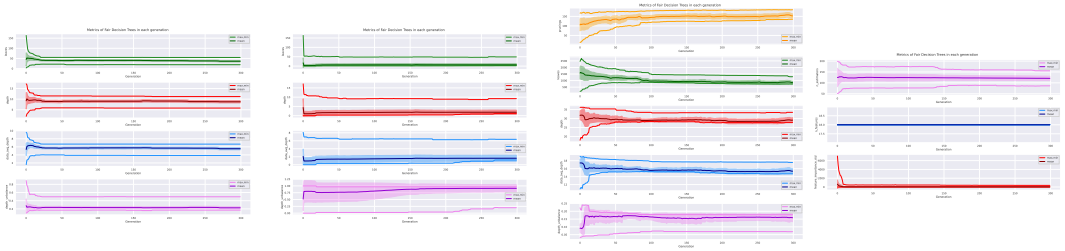


Figure 11.27.: Evolution of individuals dimensions through generations for parkinson dataset.

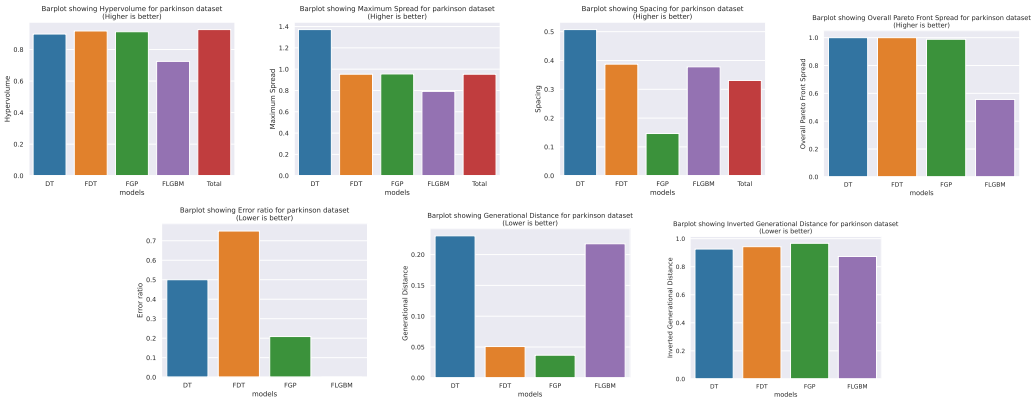


Figure 11.28.: Pareto front quality metrics for parkinson dataset.

Results for Ricci dataset

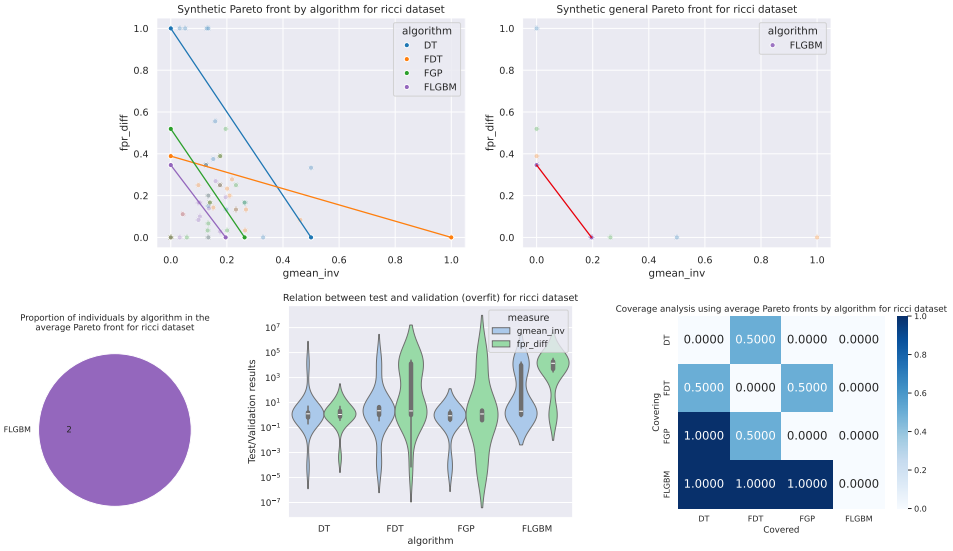


Figure 11.29.: General metrics for Pareto optimal solutions for ricci dataset.

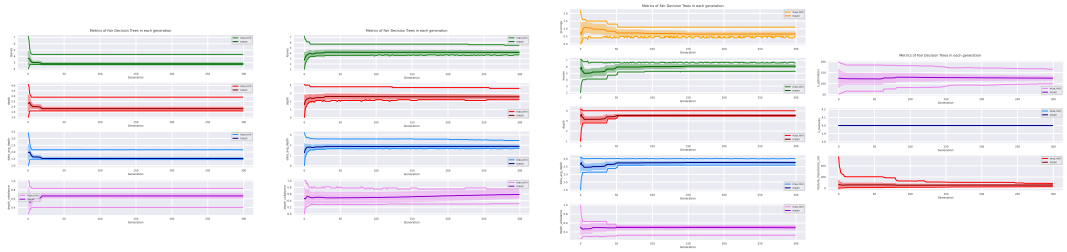


Figure 11.30.: Evolution of individuals dimensions through generations for ricci dataset.

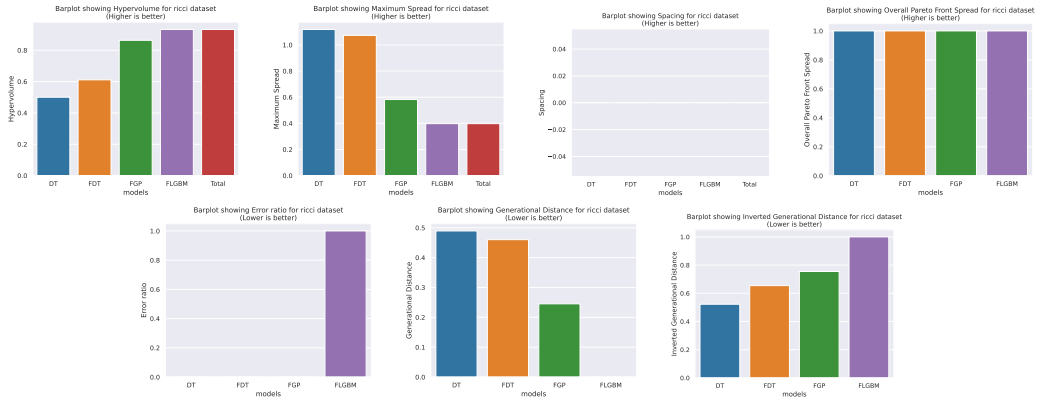


Figure 11.31.: Pareto front quality metrics for adult dataset.

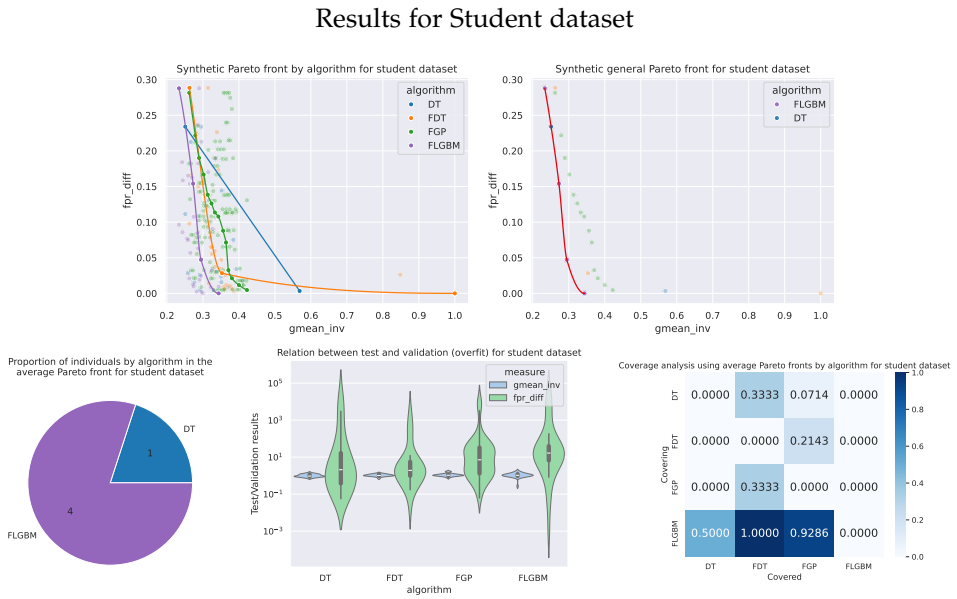


Figure 11.32.: General metrics for Pareto optimal solutions for student dataset.

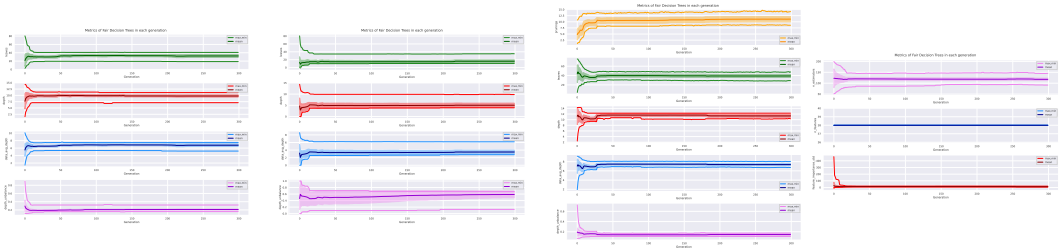


Figure 11.33.: Evolution of individuals dimensions through generations for student dataset.

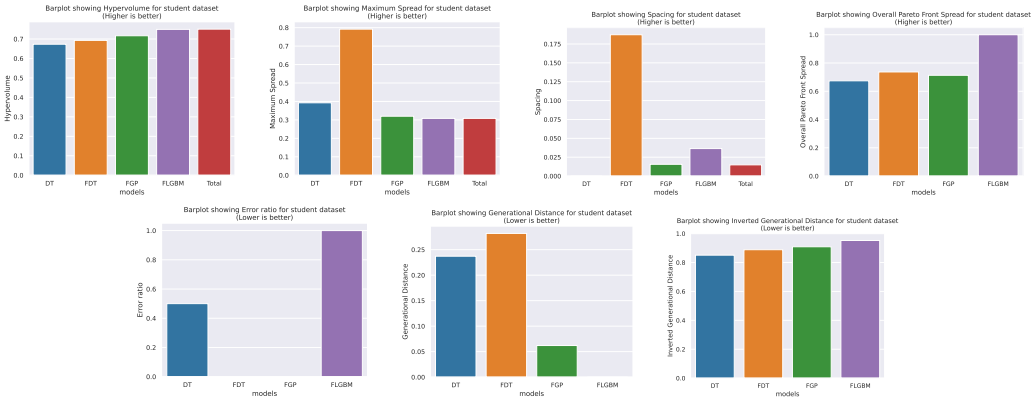


Figure 11.34.: Pareto front quality metrics for student dataset.

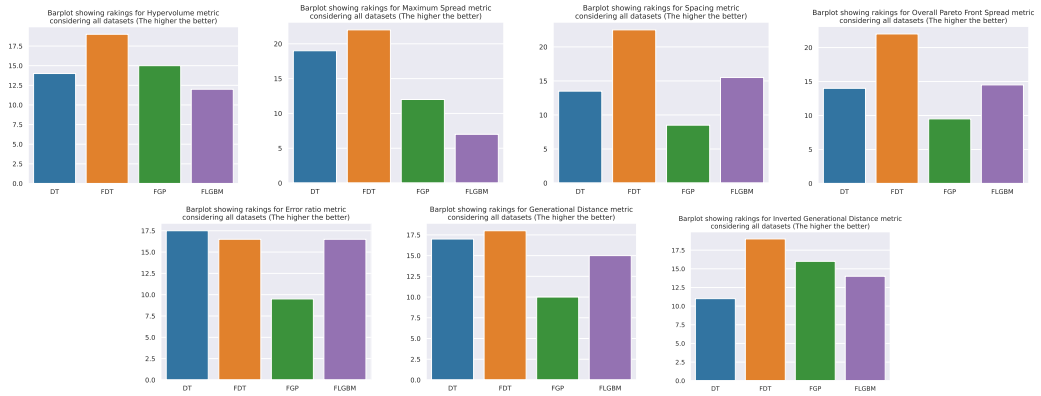


Figure 11.35.: Rankings of quality metrics among all datasets for the studied algorithms.

We can extract the following information from these graphs:

- **Shapes of the Pareto fronts**: The obtained Pareto fronts have reasonably good shapes. Each of them are distinct, and it can be clearly seen how each algorithm yields a different shape in the found Pareto front. This fact, is very positive because it shows that each algorithm contributes with solutions having different approaches and diverse balances.
- **Performance**: All algorithms have performed reasonably well; none has clearly outperformed or underperformed any other. Depending on the problem and the situation, a particular algorithm may have found better or worse solutions, which follows the No Free Lunch Theorem.
- **Validation and test**: It can also be observed that, in general, there has been a good balance in the validation and test results. However, it is true that, in general, the FairLGBM algorithm has consistently obtained scores closer to 0 across all datasets.
- **Structural convergence**: In general, the algorithms have exhibited fairly rapid convergence in the structure of the models found. As can be seen in the evolution graphs of the dimensions, the structures of individuals usually stabilize before 50 generations. This is the case except for the Fair Genetic Pruning algorithm, where it can be observed that as generations progress, the number of prunings performed generally increases. This may be because configurations with a higher number of prunings at deeper levels are being explored. Everything indicates that this algorithm would likely improve its results if allowed to optimize for a longer period.
- **General results**: The results have varied considerably across datasets, and there is no algorithm that consistently performs better for all. Let us now move on to analyze each algorithm separately:
  - **DT Algorithm**: This algorithm has achieved good results in the Diabetes, German, Parkinson, and Student datasets, with relevance also in Dutch. This algorithm has won in Overall Pareto Front Spread in the final ranking, but not by much difference compared to the second (1 point), in Generational Distance, and tied with FDT in Inverted Generational Distance. It is an algorithm that tends to cover

many solutions from others. Overall, it is a fairly reliable and consistently decent base algorithm.

- **FDT Algorithm**: This algorithm has achieved good results in the Insurance and Obesity datasets. This algorithm has won in Hypervolume, Spacing, and Error Ratio in the final ranking and tied for the top position with FDT in Inverted Generational Distance. It is an algorithm that tends to cover many solutions from others. Overall, it is also a fairly reliable algorithm since it has contributed at least 1 solution to each Pareto front.
- **FGP Algorithm**: This algorithm has achieved good results in the Adult and Compas datasets, which are highly studied in the area of fairness in machine learning. It has also provided good solutions in Dutch. This algorithm has not won in any final ranking, yet it has obtained good solutions in certain specific problems. Consistently, it has been able to generate a large number of solutions. It can be said that it may exhibit good behavior depending on the problem.
- **FLGBM Algorithm**: This algorithm has achieved good results in the Ricci dataset, which is a small dataset. Generally, it has found extreme solutions, always finding cases where an almost perfect FPR_{diff} is obtained, at the expense of an almost zero gmean_inv. It usually obtains extreme solutions, although it is capable of performing well depending on the dataset.

Additionally, a table with the results of the quality measures for the average Pareto fronts is included. This way, the results can be seen numerically, allowing for a better comparison between them. The measures for the overall average Pareto front are also included. These results can be observed in Table **??**.

Another pertinent analysis performed was a comparative analysis using statistical tests. In this case, all results found for each algorithm and execution (data partition) will be used. For each dataset and partition, the results obtained for each algorithm will be considered, the total Pareto front will be created, and all the quality measures for these sets of solutions will be calculated. Therefore, we will have a total of 100 paired measurements for each quality metric and algorithm (10 datasets with 10 different partitions each). The Wilcoxon signed-rank test for paired samples will be used to compare the results obtained. This study will compare the algorithms in pairs to determine if there are significant differences between the results of the populations. Specifically, the null hypothesis H_0 will be that the differences are symmetric around $\mu = 0$, while the alternative hypothesis H_1 will be that differences are symmetric around $\mu \neq 0$.

These results can be seen in Table 11.2. f the test result is statistically significant, (p-value < 0.05, in the upper diagional) it will show which algorithm performed better and which performed worse (lower diagonal). If the algorithm in the row has better results, it will be indicated with the symbol $\oplus$, while if it performed worse, it will be indicated with the symbol $\ominus$. Similarly, even if the results are not significant, if the algorithm in the row has better results in 60% of the occasions, it will be indicated with the symbol $+$. If it is worse in 60% of the occasions, it will be indicated with the symbol $-$. If none of the above applies, it will be indicated with the symbol $=$, indicating that they have performed comparably well an equal number of times.

Table 11.1.: Average Pareto fronts quality metrics for each dataset

Data	Quality Metric	DT	FDT	FGP	FLGBM	P.Front
adult	Hypervolume	0.82034	0.81609	0.79819	0.78365	0.82160
	Spacing	0.03803	0.13970	0.05266	0.06888	0.14347
	Maximum Spread	0.31459	0.82895	0.80314	0.25024	0.83141
	Overall PF Spread	0.35867	0.99353	0.32206	0.09826	1.00000
	Error Ratio	0.60870	0.29412	0.90476	1.00000	0.00000
	GD	0.01770	0.01107	0.02282	0.05072	0.00000
	Inverted GD	0.04430	0.01546	0.03366	0.05154	0.00000
compas	Hypervolume	0.71954	0.72288	0.72521	0.60700	0.73684
	Spacing	0.14699	0.12689	0.00733	0.33606	0.12375
	Maximum Spread	0.79360	0.79147	0.31081	0.80482	0.80482
	Overall PF Spread	0.98275	0.95644	0.21424	1.00000	1.00000
	Error Ratio	0.69231	0.50000	0.24561	0.42857	0.00000
	GD	0.01912	0.02816	0.00747	0.02419	0.00000
	Inverted GD	0.01149	0.01228	0.02364	0.03403	0.00000
german	Hypervolume	0.60005	0.60315	0.59345	0.55319	0.62464
	Spacing	0.21663	0.22369	0.05767	0.14519	0.09205
	Maximum Spread	0.67418	0.70131	0.67241	0.59357	0.70131
	Overall PF Spread	0.84075	1.00000	0.93309	0.55357	1.00000
	Error Ratio	0.00000	0.25000	0.64000	0.75000	0.00000
	GD	0.00000	0.04972	0.03286	0.14054	0.00000
	Inverted GD	0.07399	0.06969	0.02749	0.08287	0.00000
ricci	Hypervolume	0.50000	0.61111	0.86338	0.93229	0.93229
	Spacing	0.00000	0.00000	0.00000	0.00000	0.00000
	Maximum Spread	1.11803	1.07296	0.58210	0.39760	0.39760
	Overall PF Spread	1.00000	1.00000	1.00000	1.00000	1.00000
	Error Ratio	1.00000	1.00000	1.00000	0.00000	0.00000
	GD	0.48945	0.46020	0.24526	0.00000	0.00000
	Inverted GD	0.47763	0.34570	0.24526	0.00000	0.00000
obesity	Hypervolume	0.94652	0.99105	0.99469	0.99991	0.99991
	Spacing	0.00000	0.55784	0.00000	0.00000	0.00000
	Maximum Spread	1.00037	0.99825	0.07410	0.01306	0.01306
	Overall PF Spread	0.69795	0.77473	0.59558	1.00000	1.00000
	Error Ratio	1.00000	1.00000	1.00000	0.00000	0.00000
	GD	0.50847	0.33592	0.13987	0.00000	0.00000
	Inverted GD	0.15534	0.06756	0.09360	0.00000	0.00000
insurance	Hypervolume	0.98458	0.97744	0.95737	0.98524	0.99425
	Spacing	0.00000	0.00000	0.44354	0.00000	0.09041
	Maximum Spread	0.13493	0.17311	0.97164	0.36160	0.36160
	Overall PF Spread	0.35300	0.41835	0.92012	1.00000	1.00000
	Error Ratio	0.50000	1.00000	0.66667	0.00000	0.00000
	GD	0.05434	0.13457	0.27426	0.00000	0.00000
	Inverted GD	0.12891	0.14273	0.14160	0.12514	0.00000

student	Hypervolume	0.67324	0.69308	0.71666	0.74957	0.75074
	Spacing	0.00000	0.18728	0.01556	0.03641	0.01482
	Maximum Spread	0.39256	0.79194	0.31990	0.30821	0.30821
	Overall PF Spread	0.67383	0.73667	0.71309	1.00000	1.00000
	Error Ratio	0.50000	1.00000	1.00000	0.00000	0.00000
	GD	0.23718	0.28190	0.06193	0.00000	0.00000
	Inverted GD	0.14918	0.11152	0.09110	0.04759	0.00000
diabetes	Hypervolume	0.51511	0.52644	0.44481	0.14270	0.52719
	Spacing	0.20000	0.23855	0.06227	0.01178	0.01774
	Maximum Spread	0.51843	0.52992	0.44675	0.14405	0.52992
	Overall PF Spread	0.97803	1.00000	0.72397	0.10117	1.00000
	Error Ratio	0.50000	0.00000	0.91667	0.50000	0.00000
	GD	0.02297	0.00000	0.01920	0.06234	0.00000
	Inverted GD	0.04370	0.04365	0.05171	0.11809	0.00000
parkinson	Hypervolume	0.89796	0.91730	0.91250	0.72466	0.92616
	Spacing	0.50729	0.38727	0.14640	0.37797	0.33064
	Maximum Spread	1.37152	0.95255	0.95501	0.79157	0.95255
	Overall PF Spread	1.00000	1.00000	0.98858	0.55629	1.00000
	Error Ratio	0.50000	0.25000	0.79167	1.00000	0.00000
	GD	0.23044	0.05076	0.03671	0.21756	0.00000
	Inverted GD	0.07404	0.05718	0.03301	0.12688	0.00000
dutch	Hypervolume	0.83370	0.82794	0.77164	0.73328	0.83389
	Spacing	0.02662	0.24216	0.00321	0.31359	0.23323
	Maximum Spread	0.16434	0.83088	0.04512	0.73406	0.83688
	Overall PF Spread	0.18282	0.82952	0.01089	0.12603	1.00000
	Error Ratio	0.20000	0.71429	0.82500	0.80000	0.00000
	GD	0.02467	0.04268	0.01139	0.04039	0.00000
	Inverted GD	0.06809	0.03846	0.07631	0.05217	0.00000

Estos resultados complementan los vistos en los rankings de gráficos previos. De esta manera se puede ver cómo el algoritmo FDT gana al algoritmo DT en todas las medidas salvo en error ratio (eso sí, con un valor obtenido de 0.0453, cerca del no rechazo) y en generational distance donde obtuvo peores valores, y en hipervolumen obtuvieron unos resultados similares. FGP y FLGBM obtuvieron resultados más variados, aunque pierden consistentemente contra FDT (salvo en error ratio y generational distance, donde FLGBM se comportó de forma similar a FDT, y FGP ganó en generational distance a FDT).

11.3. Gráficas de valores de los hiperparámetros en los conjuntos Pareto optimales

Let us also include images showing the optimal values of the hyperparameters using violinplots, and their correlation using heatmaps. We will explain the content of the graphs below:

Table 11.2.: Paired samples Wilcoxon signed-rank test for all quality metrics using all runs and datasets

(a) Wilcoxon signed-rank test for Hypervolume metric

	DT	FDT	FGP	FLGBM
DT		7.57×10^{-1}	4.92×10^{-6}	1.98×10^{-6}
FDT	$=$		1.14×10^{-6}	3.16×10^{-9}
FGP	$\ominus$	$\ominus$		6.34×10^{-7}
FLGBM	$\ominus$	$\ominus$	$\ominus$	

(b) Wilcoxon signed-rank test for Spacing metric

	DT	FDT	FGP	FLGBM
DT		4.17×10^{-6}	1.66×10^{-1}	2.34×10^{-3}
FDT	$\oplus$		2.82×10^{-7}	4.82×10^{-9}
FGP	$=$	$\ominus$		1.87×10^{-2}
FLGBM	$\oplus$	$\ominus$	$\oplus$	

(c) Wilcoxon signed-rank test for Maximum Spread metric

	DT	FDT	FGP	FLGBM
DT		1.26×10^{-4}	3.93×10^{-1}	6.69×10^{-5}
FDT	$\oplus$		6.05×10^{-6}	5.49×10^{-12}
FGP	$=$	$\ominus$		7.96×10^{-6}
FLGBM	$\ominus$	$\ominus$	$\ominus$	

(d) Wilcoxon signed-rank test for Overall Pareto Front Spread metric

	DT	FDT	FGP	FLGBM
DT		1.59×10^{-6}	7.59×10^{-2}	4.88×10^{-6}
FDT	$\oplus$		9.43×10^{-8}	1.74×10^{-10}
FGP	$=$	$\ominus$		2.12×10^{-4}
FLGBM	$\ominus$	$\ominus$	$\ominus$	

(e) Wilcoxon signed-rank test for Error ratio metric

	DT	FDT	FGP	FLGBM
DT		4.86×10^{-2}	7.01×10^{-10}	4.64×10^{-1}
FDT	$\ominus$		8.42×10^{-13}	3.99×10^{-1}
FGP	$\ominus$	$\ominus$		9.70×10^{-9}
FLGBM	$=$	$=$	$\oplus$	

(f) Wilcoxon signed-rank test for Generational Distance metric

	DT	FDT	FGP	FLGBM
DT		2.14×10^{-2}	1.95×10^{-1}	9.56×10^{-1}
FDT	$\ominus$		4.41×10^{-1}	1.22×10^{-1}
FGP	$=$	$=$		2.34×10^{-1}
FLGBM	$=$	$=$	$=$	

(g) Wilcoxon signed-rank test for Inverted Generational Distance metric

	DT	FDT	FGP	FLGBM
DT		6.46×10^{-9}	9.34×10^{-1}	1.47×10^{-1}
FDT	$\oplus$		1.34×10^{-5}	1.70×10^{-6}
FGP	$=$	$\ominus$		4.70×10^{-2}
FLGBM	$+$	$\ominus$	$\ominus$	

- **DT algorithm**: In Figures 11.36, 11.39, 11.42, 11.45, 11.48, 11.51, 11.54, 11.57, 11.60 and 11.63 information about the distribution of hyperparameters in the Pareto-optimal set found for each respective dataset is displayed using violinplots. In this case, decision trees are used in the optimization process. Additionally, a correlation plot using heatmaps is included for the hyperparameter values, aiming to evaluate potential configurations for each problem.
- **FDT algorithm**: Figures 11.37, 11.40, 11.43, 11.46, 11.49, 11.52, 11.55, 11.58, 11.61 and 11.64 include analogous information to the previous point but for the Fair Decision Tree algorithm.
- **FLGBM algorithm**: Figures 11.38, 11.41, 11.44, 11.47, 11.50, 11.53, 11.56, 11.59, 11.62 and 11.65 include analogous information to the previous point but for the FairLGBM algorithm.
- **Graphs using every individual**: Finally, in Figures 11.66, 11.67, 11.68, analogous information is shown, but considering all Pareto-optimal individuals found for all datasets. This way, it will be possible to see jointly if there is any type of distribution, or not, in general for the hyperparameters and their relationships independent of the dataset.

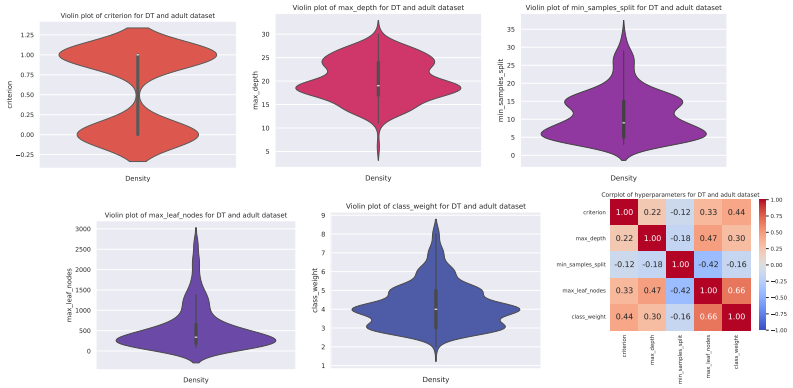


Figure 11.36.: Hyperparameter distribution for Pareto front found using DT for adult dataset.

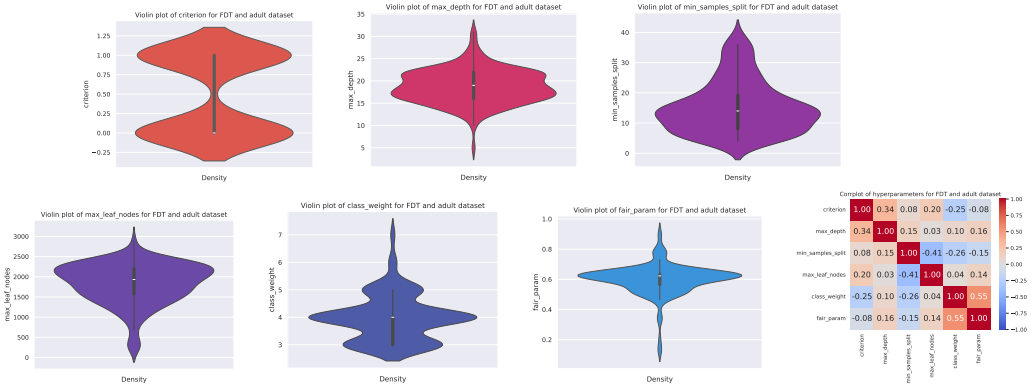


Figure 11.37.: Hyperparameter distribution for Pareto front found using FDT for adult dataset.

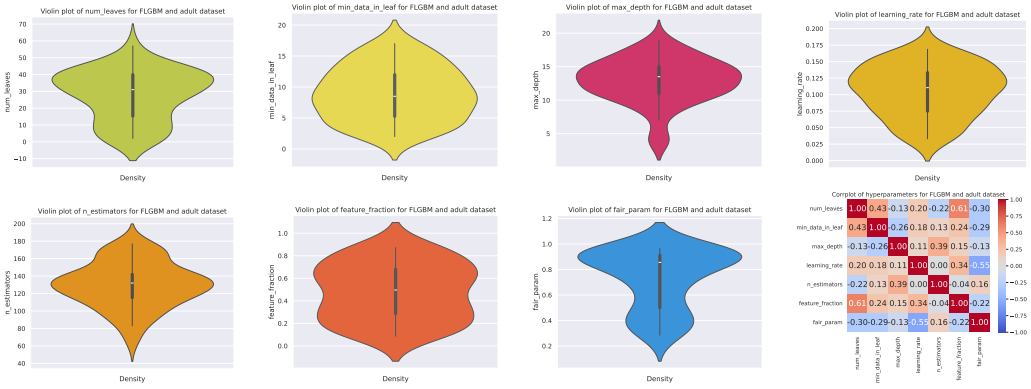


Figure 11.38.: Hyperparameter distribution for Pareto front found using FLGBM for adult dataset.

Figure 11.39.: Hyperparameter distribution for Pareto front found using DT for compas dataset.

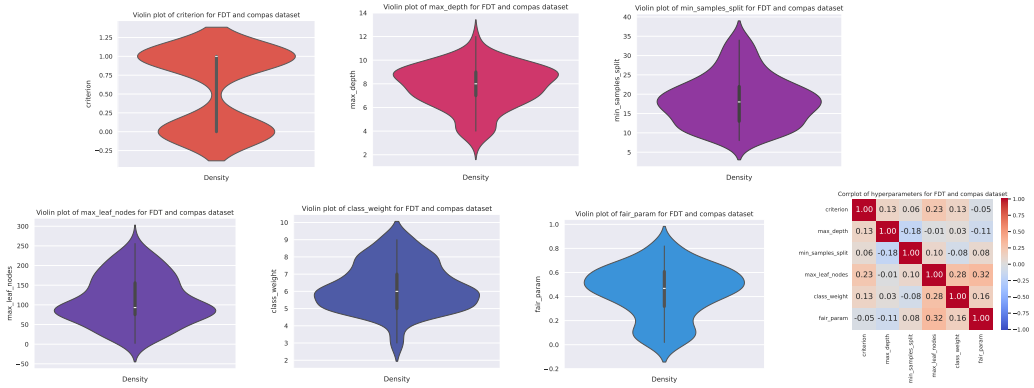


Figure 11.40.: Hyperparameter distribution for Pareto front found using FDT for compas dataset.

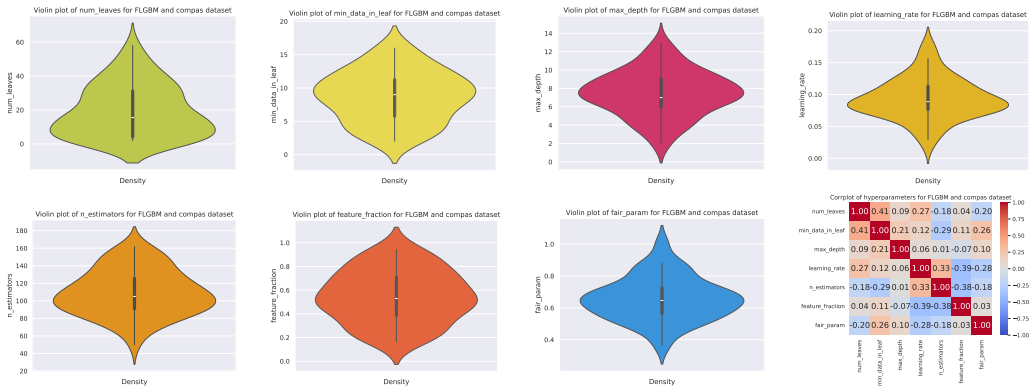


Figure 11.41.: Hyperparameter distribution for Pareto front found using FLGBM for compas dataset.

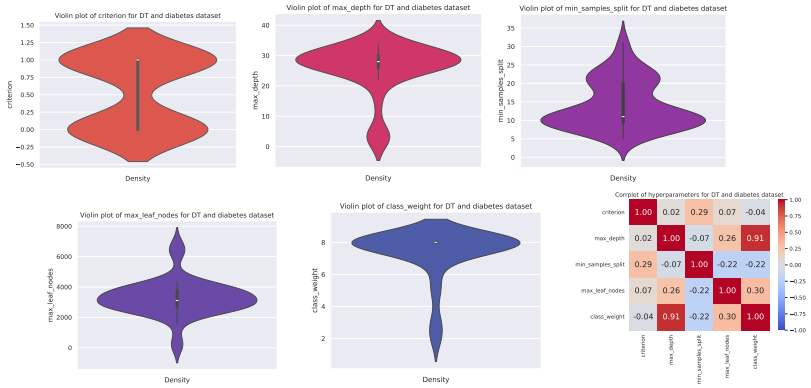


Figure 11.42.: Hyperparameter distribution for Pareto front found using DT for diabetes dataset.

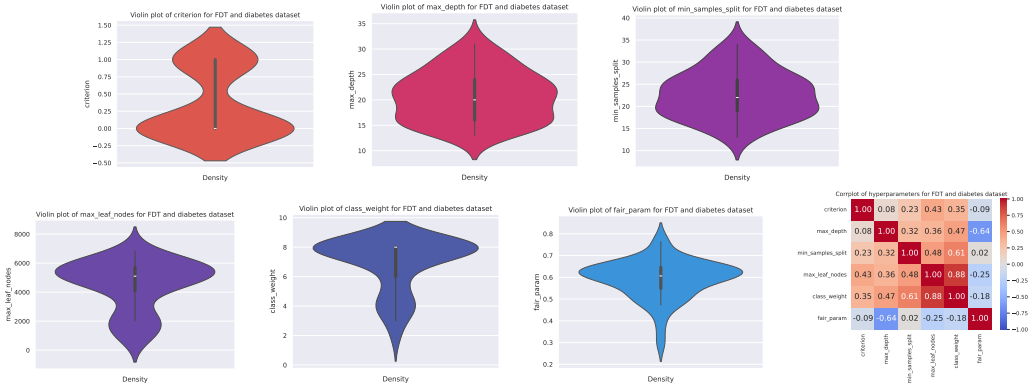


Figure 11.43.: Hyperparameter distribution for Pareto front found using FDT for diabetes dataset.

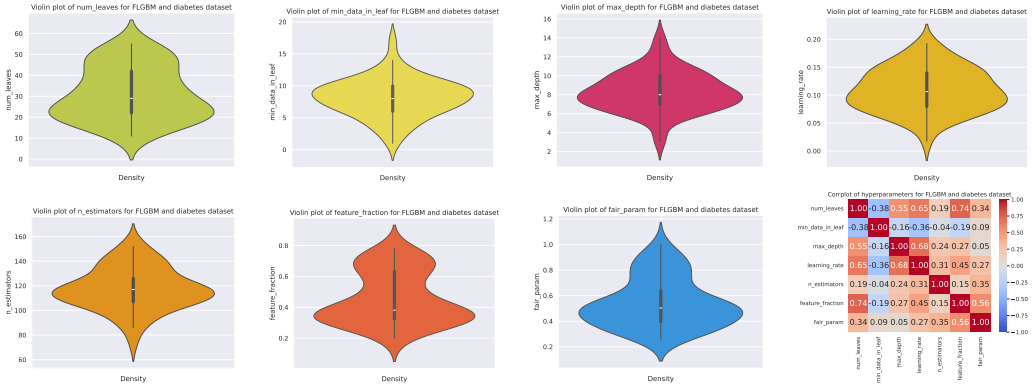


Figure 11.44.: Hyperparameter distribution for Pareto front found using FLGBM for diabetes dataset.

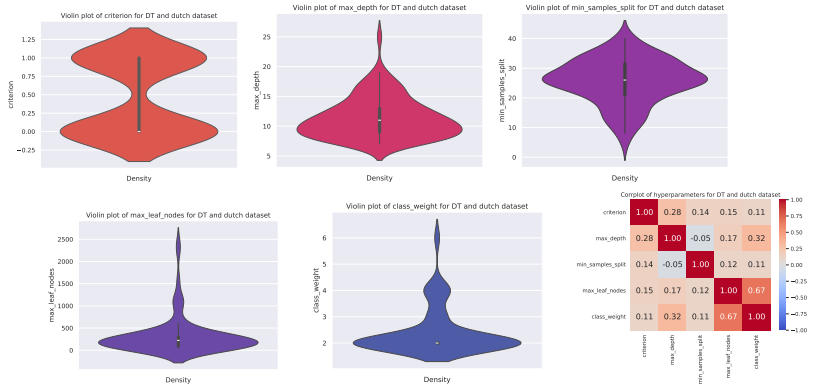


Figure 11.45.: Hyperparameter distribution for Pareto front found using DT for dutch dataset.

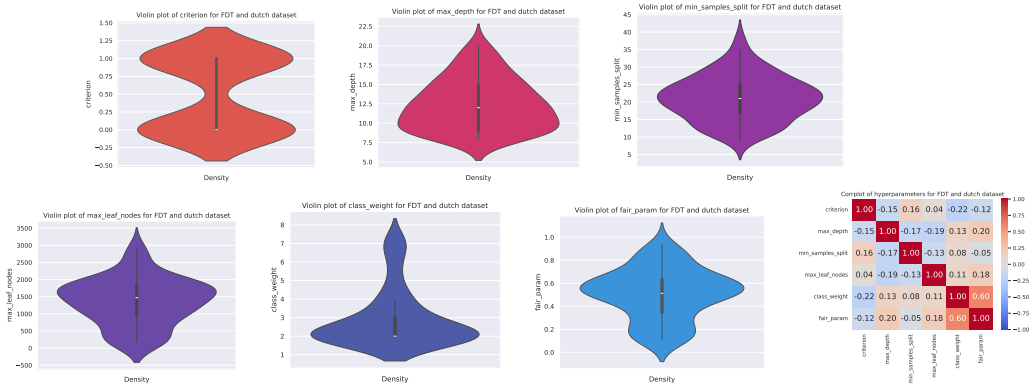


Figure 11.46.: Hyperparameter distribution for Pareto front found using FDT for dutch dataset.

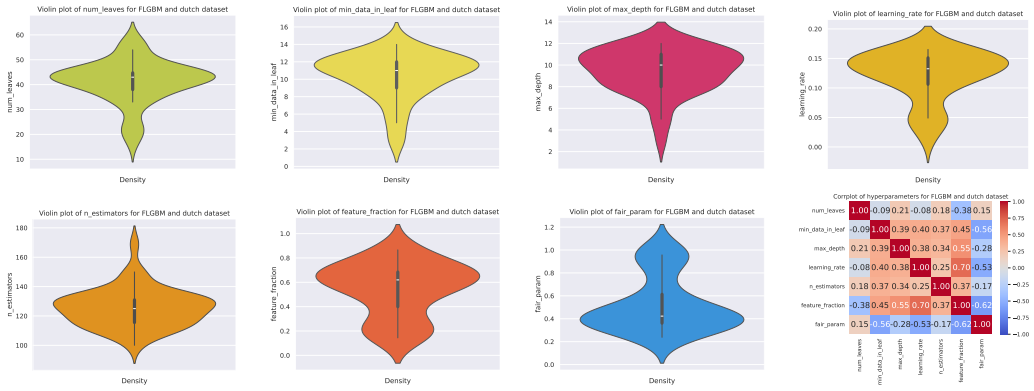


Figure 11.47.: Hyperparameter distribution for Pareto front found using FLGBM for dutch dataset.

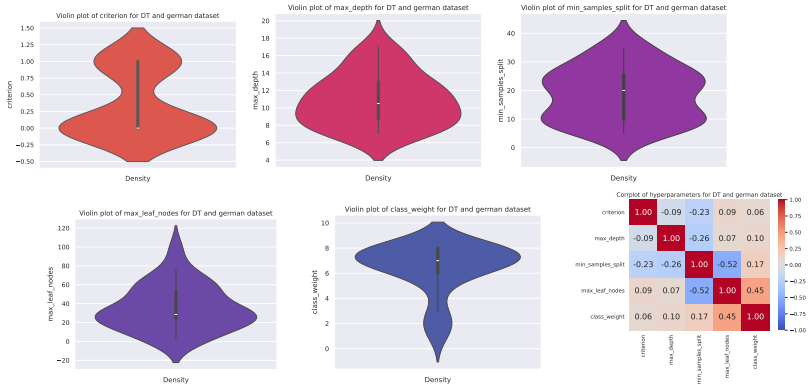


Figure 11.48.: Hyperparameter distribution for Pareto front found using DT for german dataset.

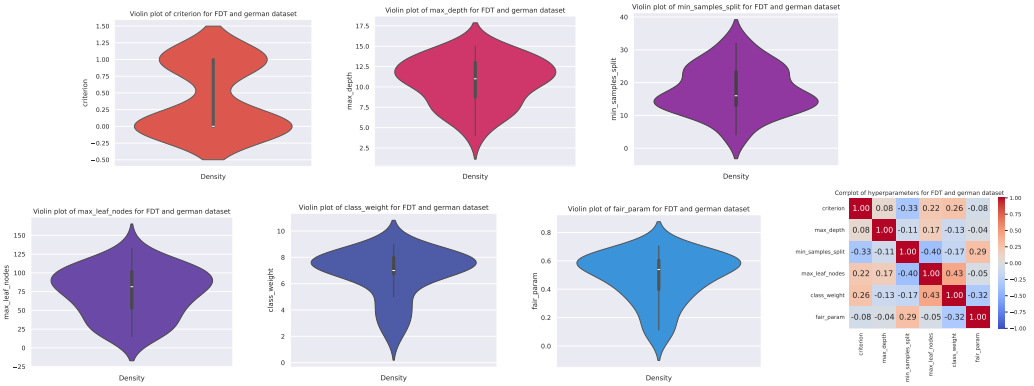


Figure 11.49.: Hyperparameter distribution for Pareto front found using FDT for german dataset.

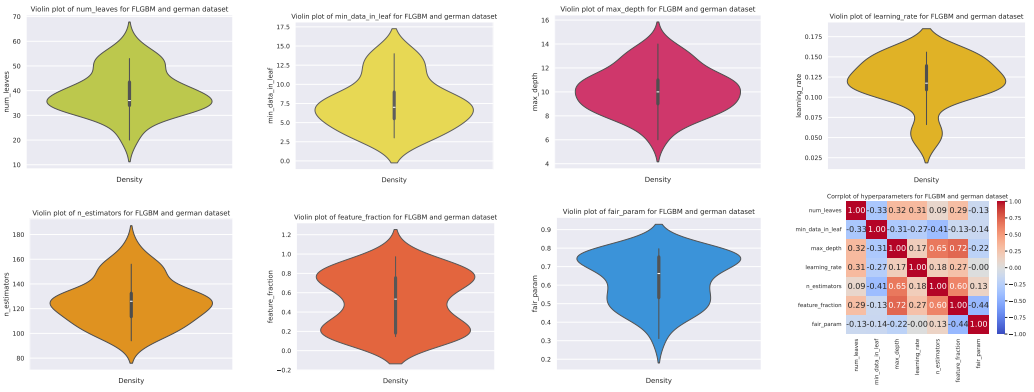


Figure 11.50.: Hyperparameter distribution for Pareto front found using FLGBM for german dataset.

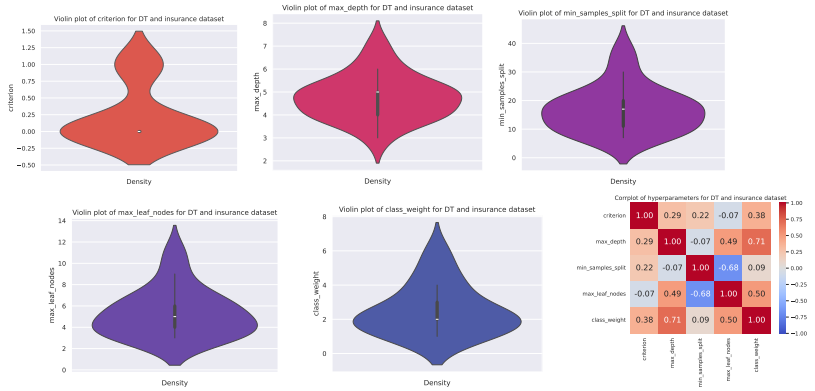


Figure 11.51.: Hyperparameter distribution for Pareto front found using DT for insurance dataset.

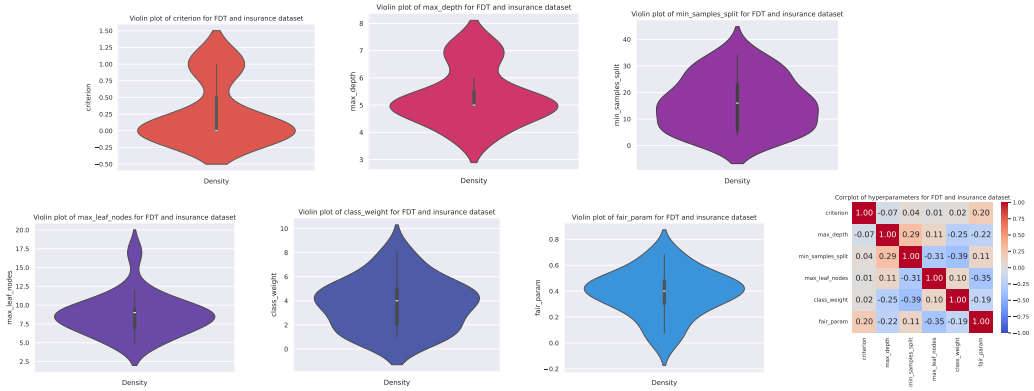


Figure 11.52.: Hyperparameter distribution for Pareto front found using FDT for insurance dataset.

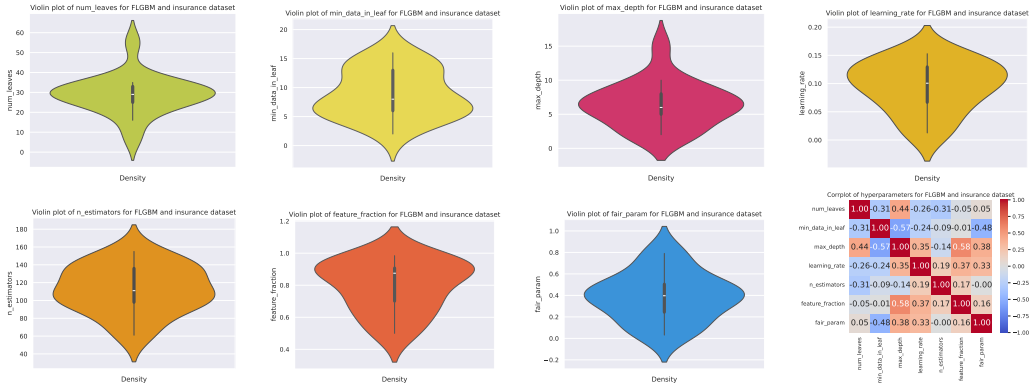


Figure 11.53.: Hyperparameter distribution for Pareto front found using FLGBM for insurance dataset.

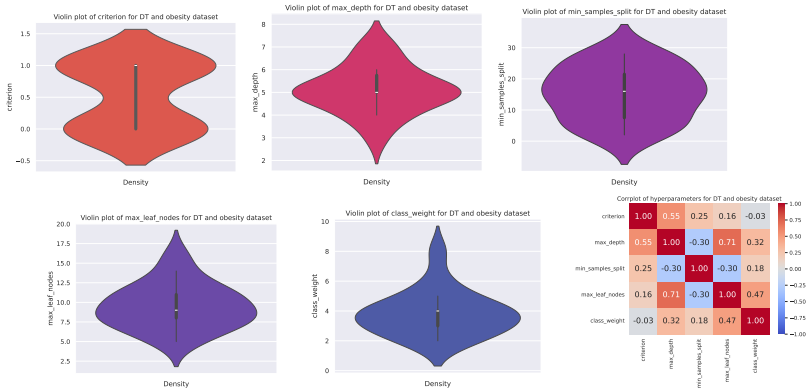


Figure 11.54.: Hyperparameter distribution for Pareto front found using DT for obesity dataset.

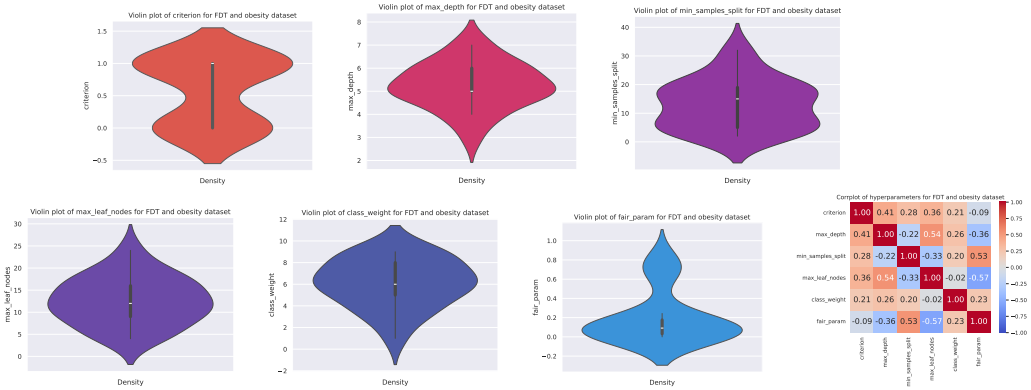


Figure 11.55.: Hyperparameter distribution for Pareto front found using FDT for insurance dataset.

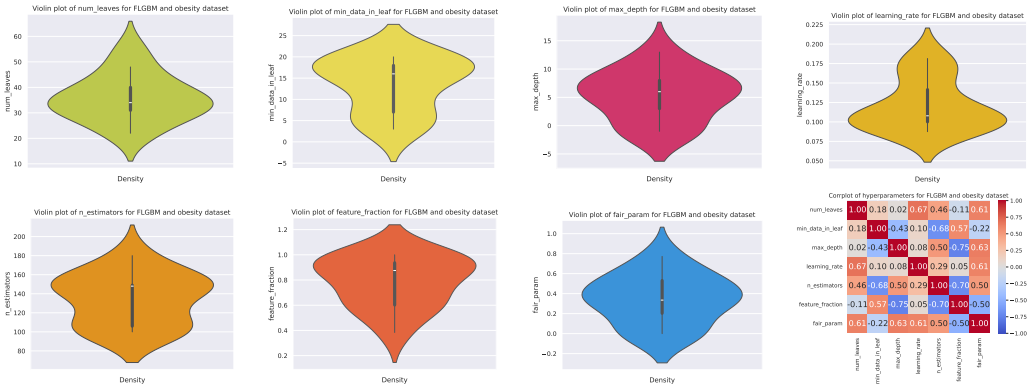


Figure 11.56.: Hyperparameter distribution for Pareto front found using FLGBM for insurance dataset.

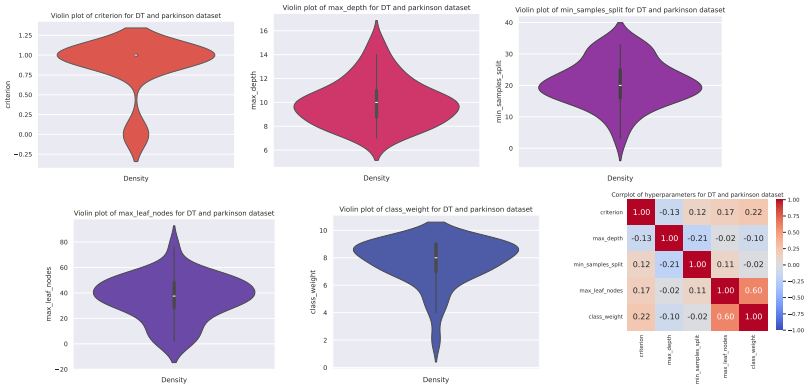


Figure 11.57.: Hyperparameter distribution for Pareto front found using DT for parkinson dataset.

Figure 11.58.: Hyperparameter distribution for Pareto front found using FDT for parkinson dataset.

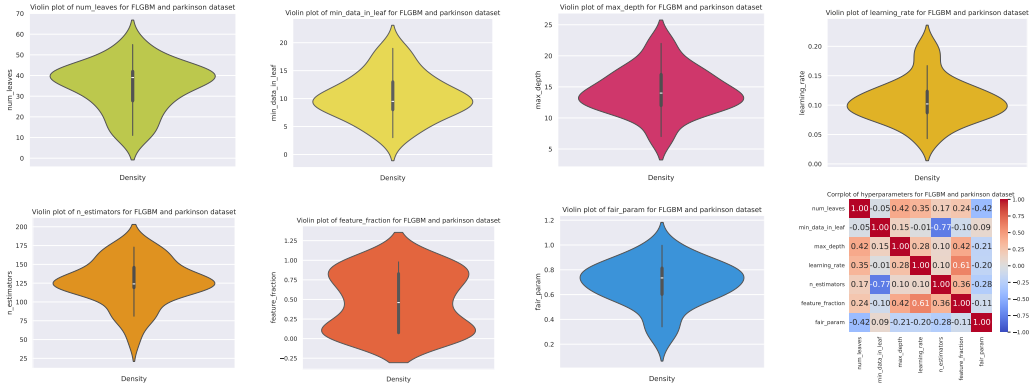


Figure 11.59.: Hyperparameter distribution for Pareto front found using FLGBM for parkinson dataset.

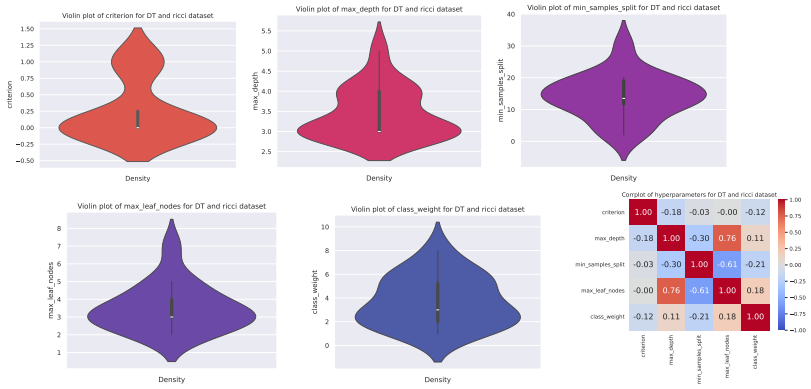


Figure 11.60.: Hyperparameter distribution for Pareto front found using DT for ricci dataset.

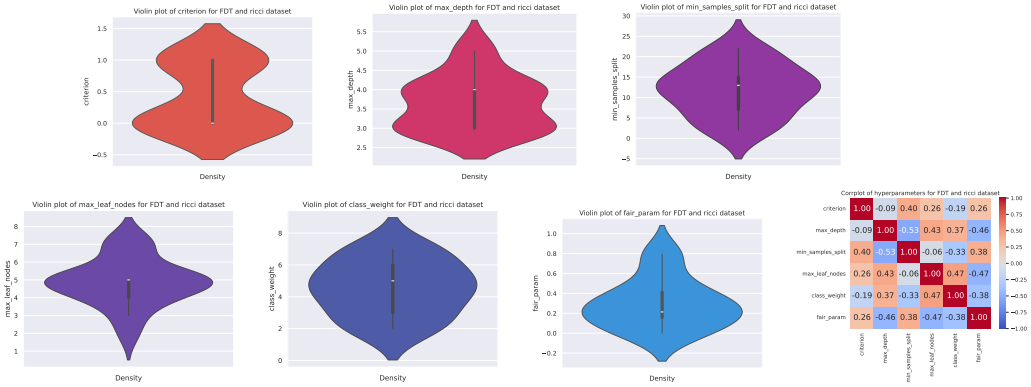


Figure 11.61.: Hyperparameter distribution for Pareto front found using FDT for ricci dataset.

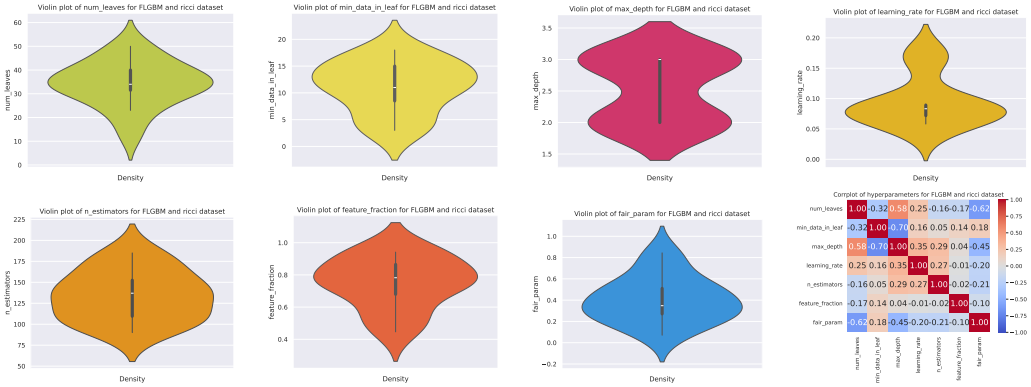


Figure 11.62.: Hyperparameter distribution for Pareto front found using FLGBM for ricci dataset.

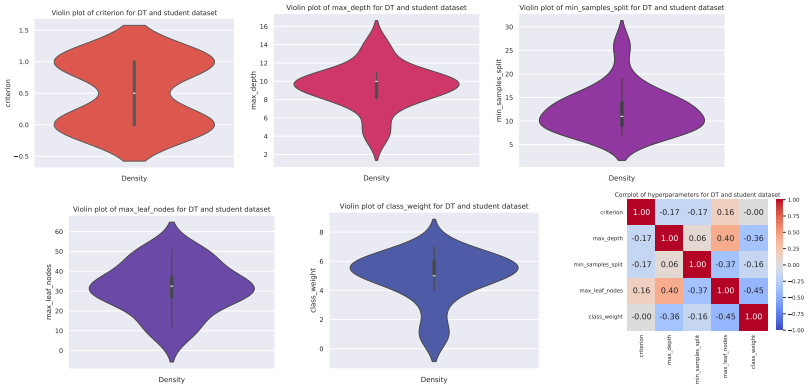


Figure 11.63.: Hyperparameter distribution for Pareto front found using DT for student dataset.

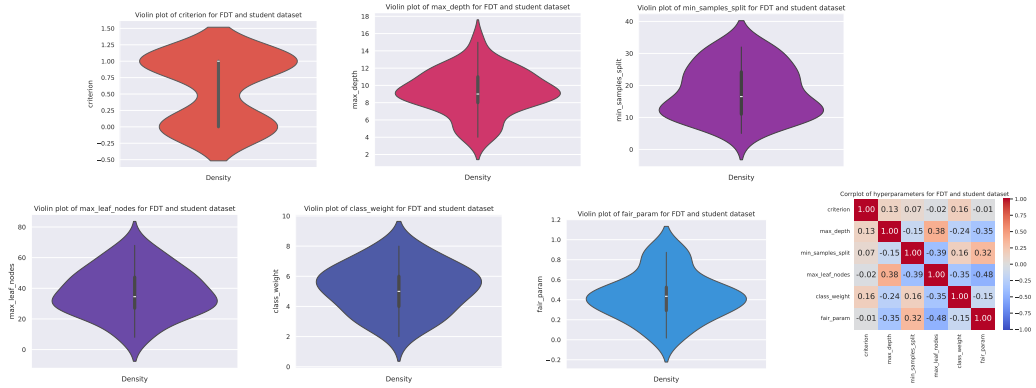


Figure 11.64.: Hyperparameter distribution for Pareto front found using FDT for student dataset.

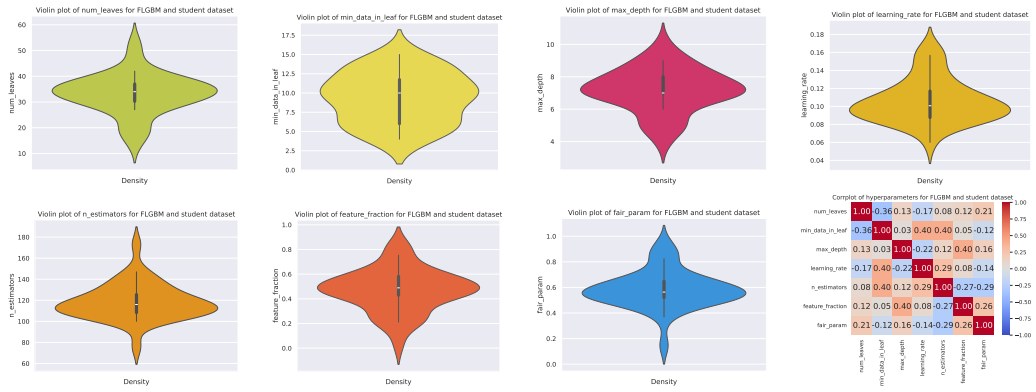


Figure 11.65.: Hyperparameter distribution for Pareto front found using FLGBM for student dataset.

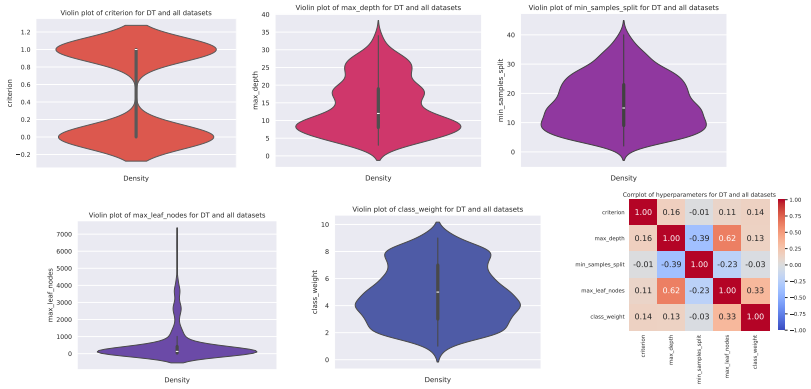


Figure 11.66.: Hyperparameter distribution for Pareto front found using DT for all datasets.

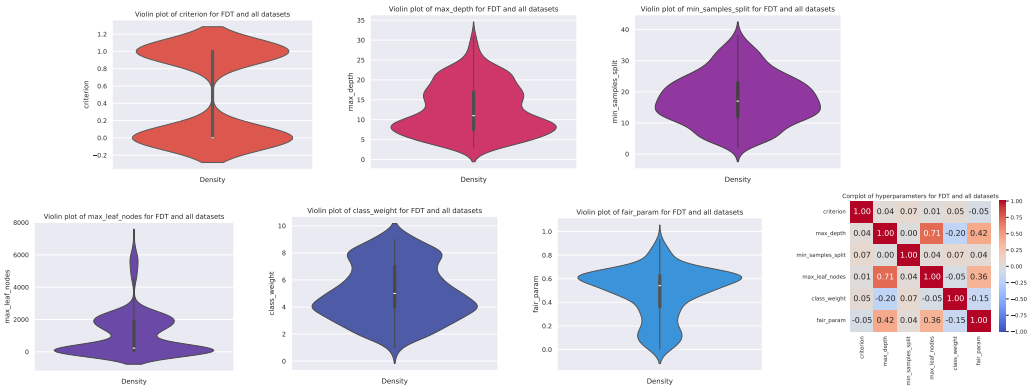


Figure 11.67.: Hyperparameter distribution for Pareto front found using FDT for all datasets.

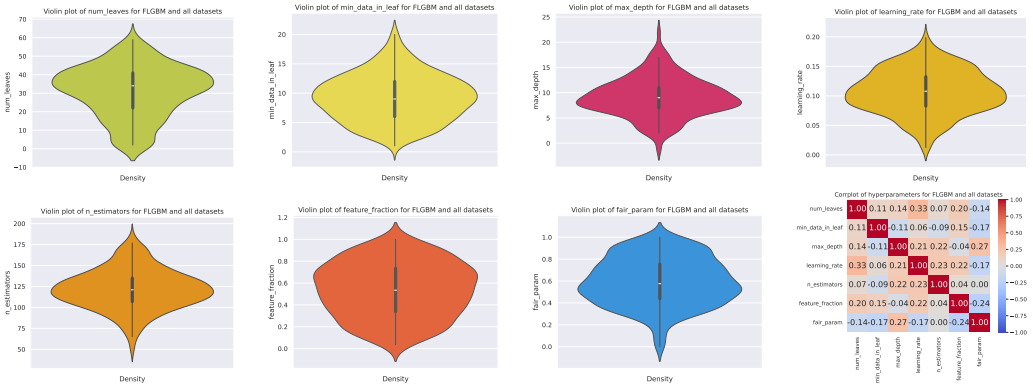


Figure 11.68.: Hyperparameter distribution for Pareto front found using FLGBM for all datasets.

Let us discuss these results based on the hyperparameter, and then we will discuss the correlations:

- **criterion**: This parameter does not exhibit any special structure; its value depends on the dataset and algorithm. It may tend more towards 0, towards 1, or not tend to either one.
- **max_depth**: In the case of LightGBM, it is always 1 or 2, which is really unusual. For DT and FairDT, it usually takes low values within their scale.
- **min_samples_split**: This hyperparameter is generally quite varied, tending towards medium values but being widely distributed.
- **max_leaf_nodes**: This hyperparameter also takes varied values, although it is not common to find values in the high part of the assigned search range.
- **class_weight**: This parameter depends heavily on the problem's own structure, so it doesn't make as much sense to analyze it. Nevertheless, some significant differences can be observed in Figure 11.67, as it generally shows higher values than in Figure 11.66.
- **fair_param**: This parameter takes very varied values and also generally depends on the specific problem being addressed. Fortunately, we can see how in certain occasions high values are selected, which confirms the usefulness of the parameter itself.
- **num_leaves, min_data_in_leaf, n_estimators, learning_rate, feature_fraction**: These tend to consistently have medium values, not taking too many extreme values.
- **Correlation plots**: Generally, it can be observed that the hyperparameters of these best models are not excessively correlated except for some pairs, such as max_depth and min_samples_split, or between n_estimators and learning_rate, showing positive correlations. Furthermore, depending on the problem, certain hyperparameters have been more correlated with others in some cases. It is important to be cautious when viewing these plots individually, as in some cases, a low number of solutions may result in very high absolute correlation values.

12. Conclusions

In this part, conclusions about the project as a whole will be drawn.

The conclusions we can draw from this work are as follows: FALTA

12.1. Future work

As future work, we can highlight the following:

- **Improvements of each method**: Implement the proposed improvements described in the sections for each of the algorithms.
- **Many-objectives approach**: Test them using a Many-objectives environment, where at least one accuracy measure and two fairness measures are considered. This way, we can verify how the algorithms behave in other, more complex execution environments.

Mendeley
Test MANOVA Para comparar las medidas.

Bibliography

References are listed in alphabetical order. References with more than one author are sorted according to the first author.

[pro, 2009] (2009). Ricci v. destefano. Technical Report 557 U.S. 557, S. C. of the United States. [Citado en págs. 7 and 103]

[obe, 2019] (2019). Estimation of Obesity Levels Based On Eating Habits and Physical Condition . UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5H31Z. [Citado en pág. 102]

[Abu-Mostafa et al., 2012] Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. (2012). *Learning from data*, volume 4. AMLBook New York, NY, USA:. [Citado en pág. 53]

[Agarwal et al., 2018] Agarwal, A., Beygelzimer, A., Dudík, M., Langford, J., and Wallach, H. (2018). A reductions approach to fair classification. *arXiv preprint arXiv:1803.02453*. [Citado en pág. 8]

[Angwin et al., 2016] Angwin, J., Larson, J., Mattu, S., and Kirchner, L. (2016). Machine bias. *ProPublica, May*, 23:2016. [Citado en pág. 7]

[Audet et al., 2020] Audet, C., Bigeon, J., Cartier, D., Le Digabel, S., and Salomon, L. (2020). Performance indicators in multiobjective optimization. *European journal of operational research*. [Citado en pág. 40]

[Balashankar et al., 2019] Balashankar, A., Lees, A., Welty, C., and Subramanian, L. (2019). Pareto-efficient fairness for skewed subgroup data. In *International Conference on Machine Learning AI for Social Good Workshop. Long Beach, United States*, volume 8. [Citado en pág. 8]

[Becker and Kohavi, 1996] Becker, B. and Kohavi, R. (1996). Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20. [Citado en pág. 100]

[Biddle, 2006] Biddle, D. (2006). *Adverse impact and test validation: A practitioner's guide to valid and defensible employment testing*. Gower Publishing, Ltd. [Citado en pág. 17]

[Binns, 2020] Binns, R. (2020). On the apparent conflict between individual and group fairness. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pages 514–524. [Citado en pág. 16]

[Binns et al., 2018] Binns, R., Van Kleek, M., Veale, M., Lyngs, U., Zhao, J., and Shadbolt, N. (2018). 'it's reducing a human being to a percentage' perceptions of justice in algorithmic decisions. In *Proceedings of the 2018 Chi conference on human factors in computing systems*, pages 1–14. [Citado en pág. 7]

[Bolukbasi et al., 2016] Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., and Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in neural information processing systems*, pages 4349–4357. [Citado en págs. 7 and 30]

[Castelnovo, 2022] Castelnovo, A. (2022). Extending decision tree to handle multiple fairness criteria. In Raedt, L. D., editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5839–5840. International Joint Conferences on Artificial Intelligence Organization. Doctoral Consortium. [Citado en pág. 69]

[Centraal Bureau voor de Statistiek (CBS) (Statistics Netherlands), 2018] Centraal Bureau voor de Statistiek (CBS) (Statistics Netherlands), M. P. C. (2018). The dutch virtual census of 2001 - ipums subset. The World Bank Microdata repository. [Citado en pág. 101]

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. [Citado en pág. 88]

[Chouldechova, 2017] Chouldechova, A. (2017). Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big data*, 5(2):153–163. [Citado en pág. 10]

[Clore and Strack, 2014] Clore, John, C. K. D. J. and Strack, B. (2014). Diabetes 130-us hospitals for years 1999-2008. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5230J. [Citado en pág. 101]

[Copas, 1983] Copas, J. (1983). Regression, prediction and shrinkage. *Journal of the royal statistical society series b-methodological*, 45:311–335. [Citado en pág. 87]

[Corbett-Davies and Goel, 2018] Corbett-Davies, S. and Goel, S. (2018). The measure and mismeasure of fairness: A critical review of fair machine learning. [Citado en pág. 21]

[Corne and Knowles, 2007] Corne, D. W. and Knowles, J. D. (2007). Techniques for highly multiobjective optimisation: some nondominated points are better than others. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 773–780. [Citado en pág. 47]

[Cortez, 2014] Cortez, P. (2014). Student Performance. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5TG7T. [Citado en pág. 103]

[Cruz et al., 2023] Cruz, A., Belém, C. G., Bravo, J., Saleiro, P., and Bizarro, P. (2023). FairGBM: Gradient boosting with fairness constraints. In *The Eleventh International Conference on Learning Representations*. [Citado en pág. 95]

[Danner et al., 2016] Danner, M. J., VanNostrand, M., and Spruance, L. M. (2016). Race and gender neutral pretrial risk assessment, release recommendations, and supervision: Vprai and praxis revised. *Luminosity, Inc*. [Citado en pág. 21]

[Deb et al., 2002] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197. [Citado en pág. 51]

[del Barrio et al., 2020] del Barrio, E., Gordaliza, P., and Loubes, J.-M. (2020). Review of mathematical frameworks for fairness in machine learning. [Citado en pág. 23]

[Deza and Deza, 2009] Deza, M. M. and Deza, E. (2009). Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer. [Citado en pág. 29]

[Domingos, 2000] Domingos, P. (2000). A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning*, pages 231–238. [Citado en pág. 56]

[Dwork et al., 2011] Dwork, C., Hardt, M., Pitassi, T., Reingold, O., and Zemel, R. (2011). Fairness through awareness. [Citado en págs. 26 and 27]

[Eubanks, 2018] Eubanks, V. (2018). *Automating inequality: How high-tech tools profile, police, and punish the poor*. St. Martin's Press. [Citado en pág. 7]

[Fabris et al., 2022] Fabris, A., Messina, S., Silvello, G., and Susto, G. A. (2022). Algorithmic fairness datasets: the story so far. *Data Min. Knowl. Discov.*, 36(6):2074–2152. [Citado en pág. 100]

[Fabris et al., 2021] Fabris, A., Mishler, A., Gottardi, S., Carletti, M., Daicampi, M., Susto, G. A., and Silvello, G. (2021). Algorithmic audit of italian car insurance: Evidence of unfairness in access and pricing. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '21, page 458–468, New York, NY, USA. Association for Computing Machinery. [Citado en pág. 102]

[Fonseca and Fleming, 1998] Fonseca, C. M. and Fleming, P. J. (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms. i. a unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 28(1):26–37. [Citado en pág. 47]

[Friedler et al., 2019] Friedler, S. A., Scheidegger, C., Venkatasubramanian, S., Choudhary, S., Hamilton, E. P., and Roth, D. (2019). A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 329–338. [Citado en págs. 7 and 100]

[Friedman, 2001] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232. [Citado en págs. 83 and 88]

[Galles and Pearl, 1998] Galles, D. and Pearl, J. (1998). An axiomatic characterization of causal counterfactuals. *Foundations of Science*, 3(1):151–182. [Citado en pág. 31]

[Garg et al., 2020] Garg, P., Villasenor, J., and Foggo, V. (2020). Fairness metrics: A comparative analysis. [Citado en pág. 21]

[Hofmann, 1994] Hofmann, H. (1994). Statlog (German Credit Data). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5NC77. [Citado en pág. 101]

[Hu and Chen, 2018] Hu, L. and Chen, Y. (2018). A short-term intervention for long-term fairness in the labor market. *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*. [Citado en pág. 17]

[Hu and Chen, 2020] Hu, L. and Chen, Y. (2020). Fair classification and social welfare. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 535–545. [Citado en pág. 9]

[Ilvento, 2020] Ilvento, C. (2020). Metric learning for individual fairness. [Citado en pág. 29]

[Johnson and Zhang, 2014] Johnson, R. and Zhang, T. (2014). Learning nonlinear functions using regularized greedy forest. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(5):942–954. [Citado en pág. 88]

[Julia et al., 2016] Julia, Angwin, J., and Larson (2016). Machine bias. [Citado en págs. 10, 11, and 101]

[Kamiran et al., 2010] Kamiran, F., Calders, T., and Pechenizkiy, M. (2010). Discrimination aware decision tree learning. In *2010 IEEE International Conference on Data Mining*, pages 869–874. IEEE. [Citado en pág. 8]

[Ke et al., 2017] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA. Curran Associates Inc. [Citado en pág. 88]

[Kleinberg et al., 2016] Kleinberg, J., Mullainathan, S., and Raghavan, M. (2016). Inherent trade-offs in the fair determination of risk scores. [Citado en pág. 22]

[Knowles and Corne, 2007] Knowles, J. and Corne, D. (2007). Quantifying the effects of objective space dimension in evolutionary multiobjective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 757–771. Springer. [Citado en pág. 51]

[Kusner et al., 2018] Kusner, M. J., Loftus, J. R., Russell, C., and Silva, R. (2018). Counterfactual fairness. [Citado en pág. 31]

[Li et al., 2015] Li, B., Li, J., Tang, K., and Yao, X. (2015). Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 48(1):1–35. [Citado en págs. 9, 47, and 48]

[Li et al., 2014] Li, M., Yang, S., and Liu, X. (2014). Diversity comparison of pareto front approximations in many-objective optimization. *IEEE Transactions on Cybernetics*, 44(12):2568–2584. [Citado en pág. 40]

[Martos Venturini, 2015] Martos Venturini, G. A. (2015). Statistical distances and probability metrics for multivariate data, ensembles and probability distributions. [Citado en pág. 27]

[Menon and Williamson, 2018] Menon, A. K. and Williamson, R. C. (2018). The cost of fairness in binary classification. In *Conference on Fairness, Accountability and Transparency*, pages 107–118. [Citado en pág. 7]

[Mukherjee et al., 2020] Mukherjee, D., Yurochkin, M., Banerjee, M., and Sun, Y. (2020). Two simple ways to learn individual fairness metrics from data. [Citado en pág. 29]

[O'neil, 2016] O'neil, C. (2016). *Weapons of math destruction: How big data increases inequality and threatens democracy*. Broadway Books. [Citado en pág. 7]

[Pearl, 1999] Pearl, J. (1999). Probabilities of causation: three counterfactual interpretations and their identification. *Synthese*, 121(1):93–149. [Citado en pág. 34]

[Purshouse and Fleming, 2007] Purshouse, R. C. and Fleming, P. J. (2007). On the evolutionary optimization of many conflicting objectives. *IEEE transactions on evolutionary computation*, 11(6):770–784. [Citado en pág. 47]

[Ranzato et al., 2021] Ranzato, F., Urban, C., and Zanella, M. (2021). Fairness-aware training of decision trees by abstract interpretation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 1508–1517, New York, NY, USA. Association for Computing Machinery. [Citado en pág. 81]

[Sani et al., 2018] Sani, H. M., Lei, C., and Neagu, D. (2018). Computational complexity analysis of decision tree algorithms. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 191–197. Springer. [Citado en pág. 60]

[Selbst et al., 2019] Selbst, A. D., Boyd, D., Friedler, S. A., Venkatasubramanian, S., and Vertesi, J. (2019). Fairness and abstraction in sociotechnical systems. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 59–68. [Citado en pág. 7]

[Strack et al., 2014] Strack, B., DeShazo, J. P., Gennings, C., Olmo, J. L., Ventura, S., Cios, K. J., and Clore, J. N. (2014). Impact of hba1c measurement on hospital readmission rates: Analysis of 70, 000 clinical database patient records. *BioMed Research International*, 2014:1–11. [Citado en pág. 101]

[Tang and Zhang, 2020] Tang, Z. and Zhang, K. (2020). Attainability and optimality: The equalized-odds fairness revisited. [Citado en págs. 18 and 19]

[Truong et al., 2024] Truong, V.-H., Tangaramvong, S., and Papazafeiropoulos, G. (2024). An efficient lightgbm-based differential evolution method for nonlinear inelastic truss optimization. *Expert Systems with Applications*, 237:121530. [Citado en pág. 89]

[Tsanas and Little, 2009] Tsanas, A. and Little, M. (2009). Parkinsons Telemonitoring. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5ZS3N. [Citado en pág. 102]

[Valdivia et al., 2020] Valdivia, A., Sánchez-Monedero, J., and Casillas, J. (2020). How fair can we go in machine learning? assessing the boundaries of fairness in decision trees. *arXiv preprint arXiv:2006.12399*. [Citado en págs. 7, 9, and 104]

[van der Linden et al., 2022] van der Linden, J., de Weerdt, M., and Demirović, E. (2022). Fair and optimal decision trees: A dynamic programming approach. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 38899–38911. Curran Associates, Inc. [Citado en pág. 81]

[Van Veldhuizen, 1999] Van Veldhuizen, D. A. (1999). Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING. [Citado en pág. 40]

[Vapnik, 1971] Vapnik, V. (1971). Chervonenkis: On the uniform convergence of relative frequencies of events to their probabilities. [Citado en pág. 55]

[Verma and Rubin, 2018] Verma, S. and Rubin, J. (2018). Fairness definitions explained. In *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*, pages 1–7. IEEE. [Citado en págs. 7 and 17]

[Xing et al., 2024] Xing, C., Liu, M., Peng, J., Wang, Y., Liu, Y., Gao, S., Zheng, Z., and Liao, J. (2024). Disturbance frequency trajectory prediction in power systems based on lightgbm spearman. *Electronics*, 13(3):597. [Citado en pág. 89]

[Zafar et al., 2017a] Zafar, M. B., Valera, I., Gomez Rodriguez, M., and Gummadi, K. P. (2017a). Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th international conference on world wide web*, pages 1171–1180. [Citado en págs. 7 and 19]

[Zafar et al., 2019] Zafar, M. B., Valera, I., Gomez-Rodriguez, M., and Gummadi, K. P. (2019). Fairness constraints: A flexible approach for fair classification. *J. Mach. Learn. Res.*, 20(75):1–42. [Citado en pág. 9]

[Zafar et al., 2017b] Zafar, M. B., Valera, I., Rogriguez, M. G., and Gummadi, K. P. (2017b). Fairness constraints: Mechanisms for fair classification. In *Artificial Intelligence and Statistics*, pages 962–970. PMLR. [Citado en pág. 8]

[Zehlike et al., 2017] Zehlike, M., Bonchi, F., Castillo, C., Hajian, S., Megahed, M., and Baeza-Yates, R. (2017). Fa* ir: A fair top-k ranking algorithm. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1569–1578. [Citado en pág. 7]

[Zehlike and Castillo, 2020] Zehlike, M. and Castillo, C. (2020). Reducing disparate exposure in ranking: A learning to rank approach. In *Proceedings of The Web Conference 2020*, pages 2849–2855. [Citado en pág. 7]

[Zhang and Zhao, 2020] Zhang, W. and Zhao, L. (2020). Online decision trees with fairness. *ArXiv*, abs/2010.08146. [Citado en pág. 69]

[Zhuang et al., 2024] Zhuang, H., Lehner, F., and DeGaetano, A. T. (2024). Improved diagnosis of precipitation type with lightgbm machine learning. *Journal of Applied Meteorology and Climatology*, 63(3):437–453. [Citado en pág. 89]