Faculty of Economics and Business Administration

Bachelor Thesis

Fair Machine Learning by means of Multi-Objective Bayesian Optimization with Constraints

Author
Jorge Calvar Seco

Supervised by
Dr. Eduardo César Garrido Merchán

Madrid, June 2023

Firstly, I would like to thank my family for always supporting me. I would also like to thank my director, Eduardo, for his invaluable guidance throughout the development of this project. Lastly, I would like to thank my classmates for being great friends and motivating me to do my best.

Resumen

En este trabajo exploramos el uso de la Optimización Bayesiana Multiobjetivo para maximizar simultáneamente la precisión (*accuracy*) y la equidad (*fairness*) de un modelo de aprendizaje automático. Exploramos diferentes métricas de equidad, y decidimos utilizar la diferencia en la tasa de verdaderos positivos. Para implementar la Optimización Bayesiana, utilizamos la librería `botorch`, y utilizamos la función de adquisición *Expected Hypervolume Improvement*, entre otras. Realizamos experimentos para ajustar varios hiperparámetros: la tasa de aprendizaje (*learning rate*), el abandono (*dropout*) y el tamaño de las dos capas ocultas. Probamos este modelo en dos conjuntos de datos tomados del UC Irvine ML Repository: *Adult Census* y *German Credit*. Demostramos que la Optimización Bayesiana obtiene mejores resultados en menos iteraciones que la búsqueda aleatoria.

Palabras clave

Optimización Bayesiana, Apredizaje Automático, Equidad

Abstract

In this dissertation, we explore the use of Multi-Objective Bayesian Optimization to simultaneously maximize the accuracy and fairness of a machine learning model. We explore different fairness metrics, and we decide to use the difference in true positive rate. To run the Bayesian Optimization, we use the `botorch` library, and we use the Expected Hypervolume Improvement acquisition function, among others. We run experiments to tune several hyperparameters: the learning rate, the dropout, and the size of two hidden layers. We test this model on two datasets taken from the UC Irvine ML Repository: *Adult Census* and *German Credit*. We prove that Bayesian Optimization obtains better results in less iterations than random guessing.

Keywords

Bayesian Optimization, Machine Learning, Fairness

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The rapid growth of artificial intelligence has transformed the way we interact with technology and the world around us. These computational models have penetrated most industries: from healthcare and finance to entertainment and transportation. AI systems continue to grow in complexity and influence. For instance, GPT-3 has 175 billion parameters [1], while its successor, GPT-4, has over 1 trillion. This represents a more than five-fold increase in the number of parameters with less than three years of difference between both models. The quality of the models has also improved. Table 1.1 shows how well GPT-4 and GPT-3.5 (ChatGPT) did in different tests. The results are surprising: GPT-4 is in the 89th percentile in the SAT Math exam. However, it is still far from replacing software engineers, as the performance in Leetcode medium and hard questions is very low. Because of the considerable impact that these models are having on our society, it is important to ensure that they are fair and equitable.

Exam	GPT-4	GPT-3.5
SAT Math	700/800 (89th)	590/800 (70th)
Leetcode (easy)	31/41	12/41
Leetcode (medium)	21/80	8/80
Leetcode (hard)	3/45	0/45

Table 1.1: Results of GPT models on specific exams. Source: OpenAI [2]

Group unfairness in machine learning models arises when the algorithms inadvertently discriminate against certain groups of individuals based on their protected characteristics, such as race, gender, or age. There are several factors that can cause this unfairness:

- **Biased data:** If the data used to train the model is already biased against certain groups, the model may also discriminate against that group. For example, this can be a problem in industries where there are discriminatory practices, so historical data is biased because of this.
- **Feature selection:** Even if a model excludes features of protected characteristics (e.g., race), if there is a feature that is correlated with a protected characteristic, then the model may still discriminate. For example, a model may have as an input variable if a person likes football. If there is a positive correlation between liking football and being a male, the model will have access to gender data even if it is not an input variable.
- **Algorithmic bias:** This is caused if the algorithm design is based on certain assumptions that may cause discrimination. For example, if a facial recognition system expects skin to have a certain color range, it may fail to work properly with other groups.

The presence of group unfairness can have severe consequences, as it not only continues existing disparities but also undermines the trust and acceptance of artificial intelligence in our society. Therefore, developing techniques to reduce group unfairness is a critical task.

The impact of group unfairness is widespread and can be observed across numerous sectors:

- **Healthcare industry:** unfairness may lead to incorrect diagnoses or treatment recommendations for specific patient populations.
- **Criminal justice system:** disproportionate sentencing or recidivism predictions for minority groups. If, in the past, certain groups have had a higher probability of committing a crime, if this data is used, unfairness will be introduced.
- **Financial sector:** discriminatory lending practices or biased credit risk assessments. For instance, it is unacceptable if a bank has higher probability of lending to a male than a female, if all other factors are the same.

The research community has been investigating various ways to improve the explainability of AI models. Explainability refers to the ability of an AI system to provide human-understandable justifications for its decisions. However, this is not always possible. As GPT-4 has over 1 trillion parameters, it is impossible to understand why the model returns a certain output. In this case, the model is like a black box, as we do not understand how it works, nor can we predict its output. The efforts to address group unfairness in machine learning have largely focused on three main approaches:

- **Pre-processing:** These techniques involve modifying the training data to remove or reduce biases before training a model. This can include re-sampling, re-weighting, or transforming the data to ensure fair representation of different groups.
- **In-processing:** It aims to incorporate fairness constraints directly into the learning algorithm, such that the model learns to make fair predictions during training. These techniques often involve optimizing objective functions that balance predictive accuracy with fairness considerations.
- **Post-processing:** They adjust the model's predictions after training to satisfy fairness constraints. This can include thresholding, or other algorithmic modifications that ensure equitable outcomes for different groups.

In this dissertation, we will explore an in-processing technique that aims to maximize both accuracy and a fairness metric.

Despite the progress made in developing fair machine-learning models, several challenges remain. One key challenge is defining and quantifying fairness itself. There is no universally agreed definition of fairness, and the choice of fairness metric can significantly impact the resulting model. Additionally, ensuring fairness in complex models, such as deep learning architectures, can be particularly difficult due to their non-linear and opaque nature.

Another obstacle is the fact that intervening to achieve group fairness may be at the cost of individual fairness. This refers to the unfair treatment of specific individuals within protected groups. It often happens, and we will prove so in this dissertation that increasing the group fairness metric will cause a drop in the overall model performance metric (e.g., accuracy). This may have a negative consequence for the whole population. It is important to balance the trade-offs between fairness, accuracy, and other objectives is an ongoing area of research. In this dissertation, we will use a tool that can maximize multiple objectives, creating a Pareto front. With that, a human can then decide what the best course of action is.

Finally, the issue of group unfairness has many other implications, such as ethical or legal, that cannot be addressed from a technical point of view. For example, the General Data Privacy Regulation (GDPR) of the European Union restricts how sensitive data can be collected and stored. This may limit how we may use protected attributes in our machine learning models, even if we are doing so to increase fairness. Ensuring that artificial intelligence is fair demands an interdisciplinary approach encompassing not only technical innovations but also the work of policymakers considering legal frameworks and societal values.

1.2 Objectives and scope

In light of these challenges, the present dissertation aims to contribute to the research on mitigating group unfairness in machine learning models. To attack the problem, we will apply an optimization technique that has gained significant attention in recent years: Bayesian Optimization.

Bayesian Optimization is an approach to solving optimization problems (i.e., when we want to maximize or minimize a function), which is particularly suitable when the target function, the one we want to optimize, is a black box and is expensive to evaluate, either from an economic or temporal point of view. The meaning of "black box" is that we do not make any assumptions about the function (other than it is continuous), and we do not have information about it, such as gradients. Later, in chapter 2, we will dive slightly into the mathematical details of BO.

The objective of this work is to apply Multi-objective Bayesian Optimization to maximize fairness and accuracy successfully. We want to show that this approach achieves better results in fewer iterations than random searching.

We assume that there is an inverse relationship between the performance of a model and its fairness. Therefore, an improvement in accuracy will come at the cost of fairness. These two variables will be the objectives that we will try to maximize using BO.

There is an important restriction to our problem: there must be a relationship between the input and output of the target function we want to use. What this means is that we must be able to control both the accuracy and fairness of our models through the hyperparameters that we will optimize.

1.3 Structure of the dissertation

The present dissertation is organized into six chapters, which aim to fulfill the objectives outlined in the previous section:

- The present chapter (1) tries to set up a context for the project, explaining the motivation and objectives of it.
- Chapter 2 emphasizes some theoretical concepts necessary for a good understanding of the project.

- Chapter 3 reviews the existing literature on the main topics treated in the dissertation including the concept of justice when applying Bayesian Optimization.
- The next chapter, chapter 4, talks about the algorithm. We will explain the methodology followed and also its implementation.
- Chapter 5 will deal with the results obtained once the algorithm has been implemented in different known datasets.
- Finally, chapter 6 will sum up the conclusions extracted from the results and will guide next possible steps to continue working on this theme.

Chapter 2

Theoretical Background

2.1 Multi Objective Optimization

In many applications, such as the one we will work on in this dissertation, there is more than one objective that we want to optimize. In this case, the solution is not a point, but a set of points, which form a Pareto optimal set, such as the one shown in Figure 2.1. This task is called multi-objective optimization.

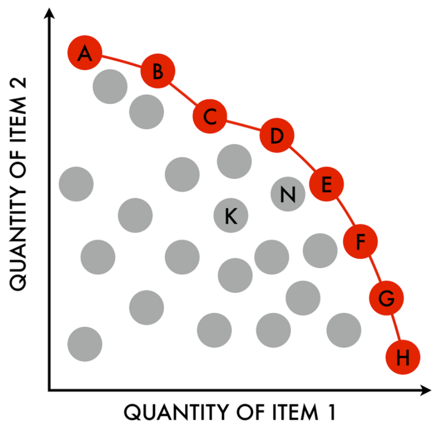

Figure 2.1: Example of a Pareto optimal set. Source: Wikipedia

The multi objective optimization problem is significantly more complex than if there is just one objective. There is not a trivial way to approach it. Some of the most common methods are:

- **Scalarization:** It consists in transforming the different objectives into a single one. To do this, we use a scalarization function. For example, we can apply weights to each objective and add them up, obtaining a single objective which we can optimize with normal techniques.

- **Constrained:** This method involves optimizing one of the objectives and using the others to set upper and lower bounds.

- **Evolutionary algorithms:** These algorithms are inspired by how evolution, and natural selection works. There are different kinds, such as Genetic Algorithms, Particle Swarm Optimization, and Differential Evolution.

There are several well-known algorithms that are commonly used to solve MOO problems, such as NSGA-II [3], which is a genetic algorithm. The algorithm starts with an initial population of potential solutions, which undergo several operations and are then ranked. Additionally, this algorithm uses a crowding distance metric, which favors solutions that are in less densely populated areas. The algorithm combines the rank and crowding distance to ensure a diverse Pareto set of optimal solutions.

Another famous algorithm is MOEA/D [4], a differential evolution algorithm that decomposes a multi-objective optimization problem into a set of scalar optimization subproblems. It uses a set of weight vectors and a scalarization function (e.g., the weighted sum) and it evolves based on the solution of the subproblem solutions.

2.2 Bayesian Optimization

In this dissertation, we will use an optimization technique called Bayesian Optimization [5]. Bayesian optimization is an optimization technique (i.e., it tries to solve a problem of the form $\max f(x)$) that is especially suitable for cases where:

- The objective function ($f(x)$) is a black box and does not have a specific structure. We do not know any information about it, including the gradient. However, we assume the function is continuous. This implies that for two vectors x_1 and x_2 in the domain space, if $x_1 \approx x_2$ then $f(x_1) \approx f(x_2)$.

- The objective function is costly to evaluate. There is a limit to the number of function images $y = f(x)$ we can compute. The practical reason behind this is that the function may take a long time to compute or that it uses a lot of computational resources, i.e., it is economically expensive.

- The function may introduce noise. This means that two different evaluations of the same point may produce different results, i.e. $f(x_1) \neq f(x_2)$ even if $x_1 = x_2$.

The aim of Bayesian Optimization is to find a point whose image is the closest possible to the global maximum (or minimum) given these constraints. There are two main components in a BO problem:

- **Prior:** it models the objective function $f(x)$. Because we do not know with certainty the values of $f(x)$, we use a Gaussian process represented as $f(x) \sim \mathcal{GP}(\mu, \Sigma)$, where μ is the mean function and Σ is the covariance function.
- **Acquisition function:** it is used to decide which point will be sampled next based on the current estimates and uncertainties about the objective function. It can be denoted as $\alpha(x|D_n)$, where D_n is the current data.

In this dissertation, we will use Multi-Objective Bayesian Optimization (MOBO). In this type of BO, the objective function $f(x)$ has more than one output. This means that we are trying to find a point x^* that maximizes a vector-valued function:

$$x^* = \arg\max f(x) = \arg\max \left[f_1(x), f_2(x), ..., f_k(x)\right] \tag{2.1}$$

where $f_i(x)$ is the i^{th} output of the objective function, and we are interested in maximizing all of the outputs.

2.2.1 Acquisition functions

BO tries to find the trade-off between exploration and exploitation. Exploration refers to trying new areas far from those points we have already evaluated with the hope of finding a better point in an unexplored area. On the other hand, exploitation evaluates points next to those where we have obtained the best results so far. Basically, it assumes that, because the target function is continuous, if a point evaluates to a high value, the area next to that point will also take high values.

To achieve a balance between exploration and exploitation, we use an acquisition function. This function receives the Gaussian process as input. The optimal point of the acquisition function is the one that will be evaluated next. Some typical acquisition functions are:

- **Expected Improvement (EI):** It is one of the most common acquisition functions. It favors points that have a good combination of expected value and uncertainty, effectively balancing exploration and exploitation. It is calculated as follows:

$$EI(x) = \mathbb{E}\left[\max(f(x) - f(x^+), 0)\right] \tag{2.2}$$

where $f(x^+)$ is the value of the best sample so far, and the expectation is computed with respect to the Gaussian process posterior.

- **Probability of Improvement (PI):** It calculates the probability that a point will make an improvement over the current best. One problem of this acquisition function is that it does not consider the magnitude of the improvement, so it can sometimes lead to excessive exploitation. It can be defined as:

$$PI(x) = \mathbb{P}\left[f(x) - f(x^+) > 0\right] \tag{2.3}$$

  where again $f(x^+)$ is the value of the best sample so far, and the probability is computed over the Gaussian process posterior.

- **Upper Confidence Bound (UCB):** It calculates a confidence interval of the target function and takes the upper interval value. This acquisition function may incur in excessive exploration as it favors points with higher uncertainty. It is defined as follows:

$$UCB(x) = \mu(x) + \kappa\sigma(x) \tag{2.4}$$

  where $\mu(x)$ and $\sigma(x)$ are the mean and the standard deviation of the Gaussian process at x, and κ is a parameter that controls the trade-off between exploitation and exploration.

In this dissertation, we will do multi-objective optimization. Therefore, we will use acquisition functions that are specific to this use case. However, they are very similar to the previous ones but with slight modifications.

A common acquisition function for MOBO, which we will use in this dissertation, is the Expected Hypervolume Improvement (EHVI). It is a generalization of the Expected Improvement. When the objective function has multiple outputs, it is not straightforward to measure the improvement. To do it, we calculate the hypervolume defined by the multiple objectives. It can also be seen as the expected increase in the Pareto area/volume after adding the new candidate. As a result, EHVI considers aims to find points that increase all objectives or that are a better trade-off between the objectives. It can be defined as:

$$EHVI(x) = \mathbb{E}\left[V(Y \cup f(x)) - V(Y)\right] \tag{2.5}$$

where Y is the set of points of the Pareto front, $V(Y)$ is the hypervolume defined by Y, and the expectation is computed with respect to the Gaussian process posterior. This measure tries to find points that increase all objectives or that provide a better trade-off between the objectives.

The qEHVI is a variation of the EHVI acquisition function, which accounts for proposing more than one candidate point. The is very similar:

$$qEHVI(x_1, ..., x_q) = \mathbb{E}\left[\sum_{i=1}^{q} V_i(Y \cup f_i(x_i)) - V_i(Y)\right] \tag{2.6}$$

where x_i are the points being evaluated, Y is the set of points of the Pareto front, $V_i(Y)$ is the hypervolume defined by Y for the i^{th} objective, and the expectation is computed with respect to the Gaussian process posterior. This measure tries to find points that increase all objectives or that provide a better trade-off between the objectives.

Other acquisition functions we will try are the qNEHVI, and the qNParEGO. This last method aims to scalarize the multi-objective optimization problem into a single-objective one by assigning weights to each of the objectives, and then optimizing the resulting scalarized function. The weights are usually updated in each iteration of the optimization. This can be represented as follows:

$$nqParEGO(x) = \arg\min_{x}\left[\sum_{i=1}^{Q} w_i \cdot f_i(x)\right] \tag{2.7}$$

where w_i are the weights assigned to each objective function $f_i(x)$ and the goal is to minimize the weighted sum.

2.2.2 Gaussian Processes

Gaussian processes are at the core of Bayesian Optimization. Mathematically, a random process is a collection of random variables. We say the process is Gaussian if any finite number of the random variables form, jointly, a Gaussian distribution. An extensive explanation of Gaussian processes can be found in [6]. These processes allow us to define a prior over an unknown function, which can be updated when images of the function have been observed.

A Gaussian process is defined by a mean function $m(x)$ and a covariance or kernel function $k(x, x')$. Often, the mean function is zero, and we focus on the covariance function, which defines the relationship between any two random variables of the process.

2.3 Fairness measures

We are concerned with analyzing the fairness of machine learning models. However, there are several categories of fairness in the field. We will be working specifically in the group fairness category, which focuses on making sure that different groups are treated similarly by the model.

We will consider that a feature is a protected class when it is unfair to use it in a model. Some examples of protected classes are age or race. There are several measures that are widely used to measure group fairness. Some of them are:

- **Equal opportunity:** it measures the percentage of true positives that are predicted correctly. We use it to measure fairness by calculating the difference in TPR for different members of a protected class. For example, let's consider a model used by a bank to decide whether a client is credit-worthy and approve them for a loan. If the TPR of males is 90%, but that of females is only 80%, there is a clear situation of unfairness. A woman who is credit-worthy has a lower probability of being approved for the loan than a male.
- **Equalized odds:** it goes further beyond than the previous metric and establishes that not only the TPR must be the same for the model to be fair, but also the false positive rate (FPR). What motivates this metric is that it can also be considered unfair if members of a class have a higher chance of being a false negative than members of another class. Returning to the previous example, if the FPR is higher for males than females, then males who are not credit-worthy would have a higher chance of being approved for the loan than females.
- **Statistical parity**: this metric states that, for a model to be fair, the probability the model outputs for a class must be the same independently of the protected class variable.

Throughout this dissertation, we will measure fairness by calculating the difference in TPR, i.e., we will use the equal opportunity metric.

To better understand this metric, we will provide an example. Let's denote TPR_m as the true positive rate for males and TPR_f as the true positive rate for females. Then, the difference in TPR can be calculated as:

$$\Delta TPR = |TPR_m - TPR_f| \tag{2.8}$$

This difference, ΔTPR, is what we aim to minimize for fairness under the equal opportunity criterion. In an ideal fair model, ΔTPR would be zero, which means that the model has the same true positive rate for both males and females.

It's important to note that TPR, also known as recall or sensitivity, is defined as $TPR = \frac{TP}{TP+FN}$, where TP represents True Positives and FN represents False Negatives. So in this context, TPR_m and TPR_f would be calculated by considering true positive and false negative outcomes for males and females, respectively.

Chapter 3

State of the art

In this section, we outline some recent and interesting uses existing in the literature of Bayesian Optimization.

3.1 Random Scalarizations

Paria et al [7] proposed a framework for MOBO. The acquisition function that they use is a scalarization function that has a parameter λ, which will be sampled from a probability distribution. By using different scalarization functions (by setting λ), this method becomes highly flexible and is able to adapt to many kinds of problems. This method also returns a set of points that form a Pareto front. This is achieved by minimizing a Bayes regret function, which penalizes points that are not Pareto optimal or that are clustered in a small region. In Figure 3.1, we observe the evolution of the regret function as the number of iterations increases. We see that this method outperforms other common MOO algorithms.

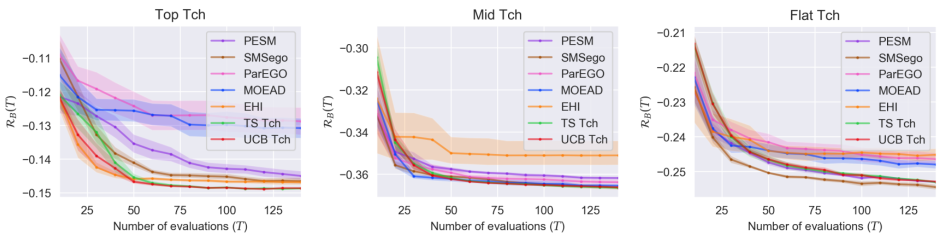

Figure 3.1: Performance of the framework using Tchebyshev random scalarizations for different sampling regions of the Pareto front (top, mid, flat), compared with other well-known MOO algorithms. Source: [7]

This method is specially suited to situations when there are more than two objectives, as it scales linearly with the number of objectives. In comparison, computing the EHVI acquisition function becomes really expensive when there are more than two objectives. However, this is not a problem that we will have to face in this dissertation, as we will only be concerned with the performance and fairness objectives.

3.2 The concept of fairness when applying Bayesian Optimization

Machine learning's growing prominence and the essential need for fairness in algorithms have prompted the development of various techniques to assess and rectify biases in these models. These biases are related with variables like genre, sex or age that can favor or harm people based on the algorithms' outputs. However, most of these strategies are specific to one model type and a certain fairness concept, which limits their practical utility. In this section, it will be discussed some implementations and ideas to solve this problems of fairness.

Before diving into these solutions, it is important to have an idea of what fairness is. [8] deals with this problems of defining this concept. In the article, three different fairness conditions are defined that can be summarized in: calibration within groups and balance for both negative and positive classes symmetrically. After presenting them, it is proven that unless very special and highly constrained cases, there is no way that all those three conditions can be satisfied simultaneously. With this idea, is is introduced the concept of the inherent trade-off when trying to determine fairness.

FairBO [9] is an algorithm that applies BO to ensure a fairness constraint. The authors argue that we often have a predefined fairness requirement when we are creating a fair model, which could from a law, a company policy, or other ethical source. Therefore, they claim that using MOO to create a Pareto front is not necessary, and that their algorithm is much more efficient in this case.

The contribution of FairBO is the proposal of a modification of the EI acquisition function to account for the fairness constraint. This is shown in Equation 3.1, where $c(\boldsymbol{x})$ is the fairness level and ϵ is the required level of fairness.

$$cEI(\boldsymbol{x}) = EI(\boldsymbol{x})P(c(\boldsymbol{x}) \leq \epsilon) \tag{3.1}$$

To prove their hypothesis, they have run experiments on the Adult Census, German Credit, and COMPAS datasets. They have optimized the hyperparameters of four ML models: XGBoost, Random Forest, a Neural Network, and a Linear model. The results are shown in Figure 3.2, where we observe that Fair BO outperforms random search.

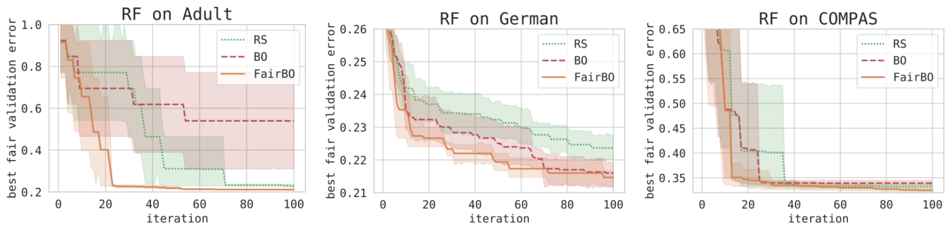

Figure 3.2: Results achieved using the FairBO algorithm. Source: [9]

The difference between FairBO and our work is that we will apply Multi-Objective Bayesian Optimization instead of using a single accuracy objective and applying a fairness constraint. This is because we think there are benefits to exploring the trade-off between fairness and performance, and not setting an a priori fairness constraint.

Another important work in the field of fairness is [10]. It deals with the fact of how can decision making be fair when there is uncertainty in the underlying probabilistic model of the world. The main conclusion is that it is crucial to directly include uncertainty about parameters in machine learning models and it is introduced the concept of Bayesian fairness. By applying this concept in some datasets, the study demonstrates that adopting a Bayesian viewpoint can result in effective and equitable decision-making rules, even amidst substantial uncertainty.

Chapter 4

Methodology

In this chapter, we will talk about the algorithm, its methodology, and implementation.

The first part is to explain the procedure of the algorithm design. It is divided into three parts, which will be explained in the following subsections. These are:

- **Bayesian Optimization (BO) model:** it uses the outputs of the target function to determine what would be the best candidate to try out next.
- **Trainer:** it implements the training logic of the previous model. It acts as the target function of the Bayesian optimization model.
- **Deep learning model:** it is a set of layers that define a neuronal network.

4.1 Bayesian Optimization

This is the most important part of the algorithm. It has been coded using `BoTorch`[1], which is a Python module for BO built on top of `PyTorch`. In Figure 4.1, we show a diagram of the process, which has the following parts:

1. Firstly, we sample a set of random points and evaluate them. The purpose of this step is to get an initial training sample to fit the Gaussian process.
2. The next step is to get the next candidate. To do this, we fit a Gaussian process with all the points we have evaluated so far, and we maximize the acquisition function of this process.

$$x_{next} = \arg\max_{x} AF(x; GP) \tag{4.1}$$

[1]See the official documentation

3. Then, we evaluate the candidate point with the target function. And we repeat the previous step. This is the training loop.

4. Finally, we return the set of points that form the best Pareto set.

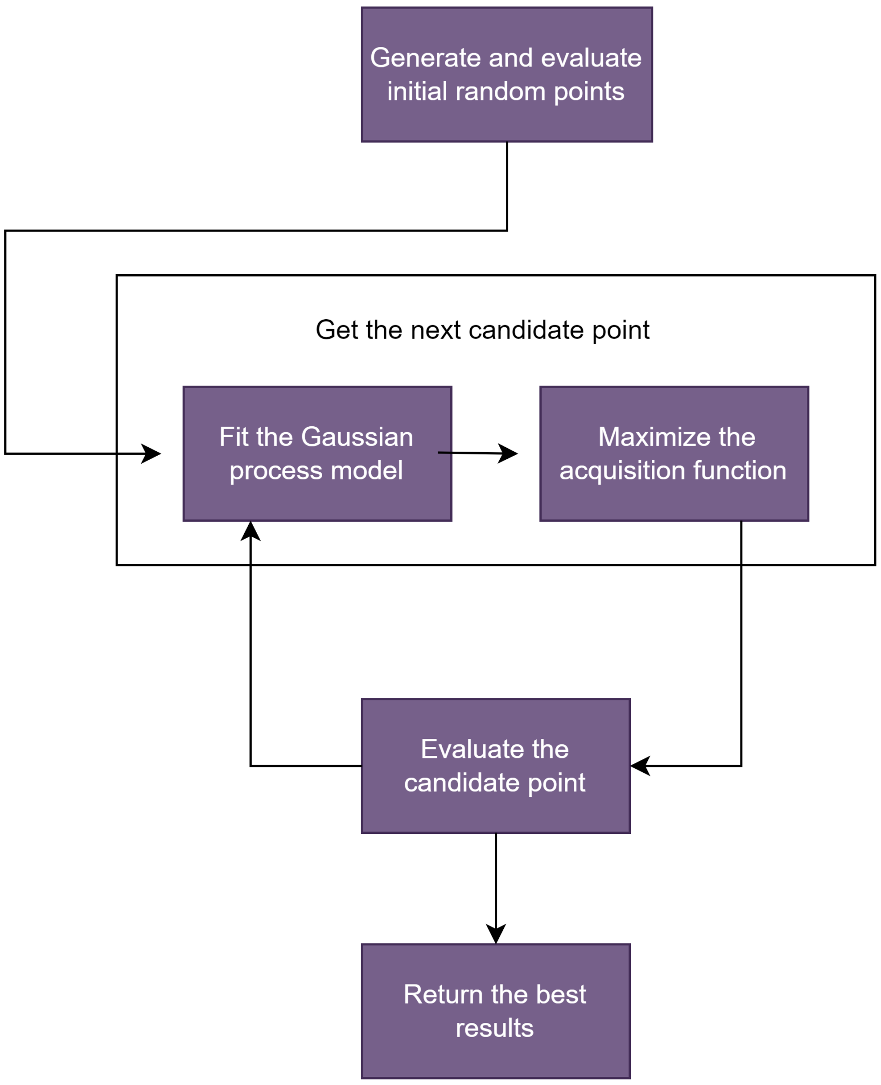
Generate and evaluate initial random points
Get the next candidate point
Fit the Gaussian process model
Maximize the acquisition function
Evaluate the candidate point
Return the best results

Figure 4.1: Diagram of the BO algorithm

4.1.1 `MOBO_Experiment` class

The `MOBO_Experiment` class is the entry point for running an experiment with our algorithm. It defines all the logic of Bayesian Optimization. When creating an instance, we must pass it a multi-objective function, which we will try to maximize. All the code can be found in the Github repository (see Appendix B). Here, we will explain the most important component: the `run(...)` method, which runs a single experiment.

The first part is initializing the `x` and `y` variables. It does not make much sense to start applying BO without any initial point because we will not be able to fit a Gaussian process that has some meaning, i.e., the process would have the same mean and variance at every point. Therefore, before applying the algorithm, we generate a set of initial random points (we have set this hyperparameter to five, but it can be changed).

Additionally, during the algorithm we will also compute the hypervolume of the Pareto front we have achieved so far. This is a good metric because it takes into account both objectives and all iterations in the experiment and combines this into a single number, which allows us to compare performance of different experiments. The hypervolume can be easily computed with help of other defined class called `DominatedPartitioning`.

We have allowed for the possibility of simultaneously running a random experiment with the BO algorithm. The purpose of this is to be able to compare the performance of BO. As opposed to BO, which uses an intelligent algorithm to decide which is the best next point to try, the random method just samples from a uniform distribution $U[0, 1)$.

We start the loop which we will run for the number of iterations we have set in the experiment setup. In this loop, we will call the `get_candidates` function (explained later in section 4.1.2), which returns the next point to try out. Next, we call the target function passing this point as input.

The vectors `x` and `y` contain all candidates that have been tried out. Therefore, when obtaining a new candidate we append it to these vectors. Additionally, we will compute the hypervolume on each iteration to find out how it changes during the experiment.

If we are running the random experiment as well, we will do the same to the random vectors. In this case, the new candidate to try out is obtained with the `torch.rand` function in the same way that we used to generate the initial points.

Lastly, we create a dictionary with all the variables that we want to save for later analysis.

To save this dictionary, we can just use the `torch.save(...)` function, which offers support for tensor objects.

4.1.2 `get_candidates` function

The `get_candidates(...)` function contains the core of the BO logic. It receives the vectors `x` and `y` with all the points that have been tried out so far and returns a new candidate point obtained with BO.

The first step of this function is two create a Gaussian Process model, which consists of several single models, one for each objective. Then, we fit this model, which calculates the Gaussian process that best explains the input points.

Now, we create a sampler object. Later, to optimize the acquisition function, it will be necessary to integrate over it. This is approximated using Monte Carlo integration and for that, we will have to sample sample points from the posterior distribution. This is the task of the sampler.

The partitioning aims to simplify the process of optimizing the acquisition function by dividing the Pareto front into a set of non-overlapping parts, calculating the improvement on each of them separately, and then summing them up.

We create an acquisition function using the model, the reference point, the partitioning and sampler. This function will estimate the improvement of each new point.

Finally, we optimize the acquisition function to find out which is the best candidate.

4.2 Target function

In the diagram of Figure 4.1 there is a step where we evaluate the candidate point. This is done by calling the target function, which is the black box of BO. Inside our target function, we will instantiate a new model with initial random weights, we will train it, and then evaluate it. Finally, we will compute and return a performance metric (e.g., accuracy) and a fairness metric (e.g., the difference in TPR). Figure 4.2 shows a diagram of how our target function works from the outside. We observe that we are using multi-objective optimization, as there are two values that we are trying to maximize (performance and fairness).

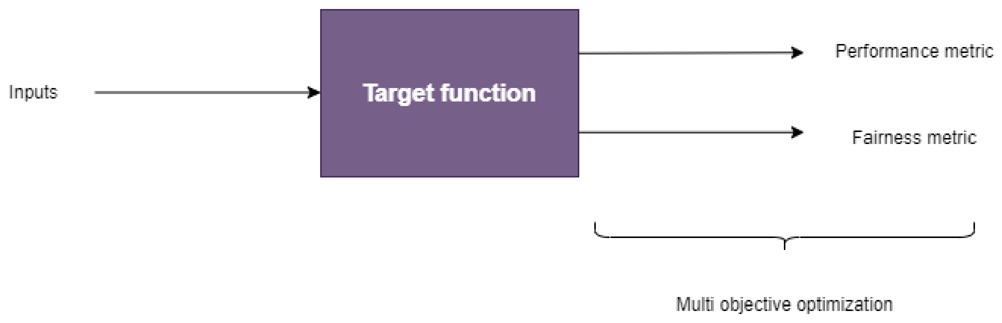

Figure 4.2: Diagram of the target function

Before training the model, we will separate the dataset into two subsets: one for training and the other for validation. This is necessary because if we evaluate the model on the same data we used to train it, we could obtain results that are not real because of overfitting.

However, when doing BO, there is another way in which overfitting can take place. After each iteration, we use the output of the target function to decide which are the best points to try next. As we explained before, the output of the target function includes the validation accuracy. Therefore, we are actually optimizing the validation accuracy, and this means that we can cause overfitting. To solve this, we will use K-fold cross-validation, so that we are not influenced by a specific train-validation split.

4.3 Model

The target function explained in the previous section creates and trains a model. This model is a neural network that we have created using `PyTorch`. Additionally, we have used the `PyTorch Lightning` module, which simplifies the process of training deep learning models.

The architecture of the neural network is shown in Table 4.1. It has two hidden layers with n_1 and n_2 neurons, respectively. The default value of these variables is 200 and 100, but we will also try to set them using BO. When doing that, they will be the input shown in Figure 4.2. After each linear layer, we add a ReLU activation function and a dropout layer. The value of the dropout can also be set using BO, and the default is 0. To train the model, we have used the Adam optimizer [11]. The default learning rate is 0.001, but this can also be optimized using BO.

Layer	Output shape
Input	(# features,)
Linear	$(n_1,)$
ReLU	$(n_1,)$
Dropout	$(n_1,)$
Linear	$(n_2,)$
ReLU	$(n_2,)$
Dropout	$(n_2,)$
Linear	(1,)
Sigmoid	(1,)

Table 4.1: Layers of our simple classification model

In Table 4.2, we show the four input variables of the model. The candidates proposed by BO are normalized, so they take values between 0 and 1. Therefore, we have to apply a transformation for it to take an acceptable value. This transformation depends on the variable. In the Mapping function column of Table 4.2, we show the transformations we have applied. In the case of the learning rate, we have applied an exponential function between the minimum and maximum values. For all others, we have applied a linear transformation, so the input is proportional to the output.

Input	Min value	Max value	Mapping function
learning rate	10^{-5}	10^{-1}	$10^{-(1+4\cdot x)}$
dropout	0	0.4	$x \cdot 0.4$
n_1	50	500	$50 + (500 - 50) \cdot x$
n_2	$0.3 \cdot n_1$	$0.6 \cdot n_1$	$0.3 \cdot n_1 + 0.3 \cdot n_1 \cdot x$

Table 4.2: Input variables to the classification model and range of values

4.4 Implementation

In this section, we will present the implementation. We will show a diagram of the process carried out when executing the algorithm and the parameters needed to run it.

The diagram from figure Figure 4.3 represents the different classes that have been coded in Python (explained in the methodology part, and the relationship between them.

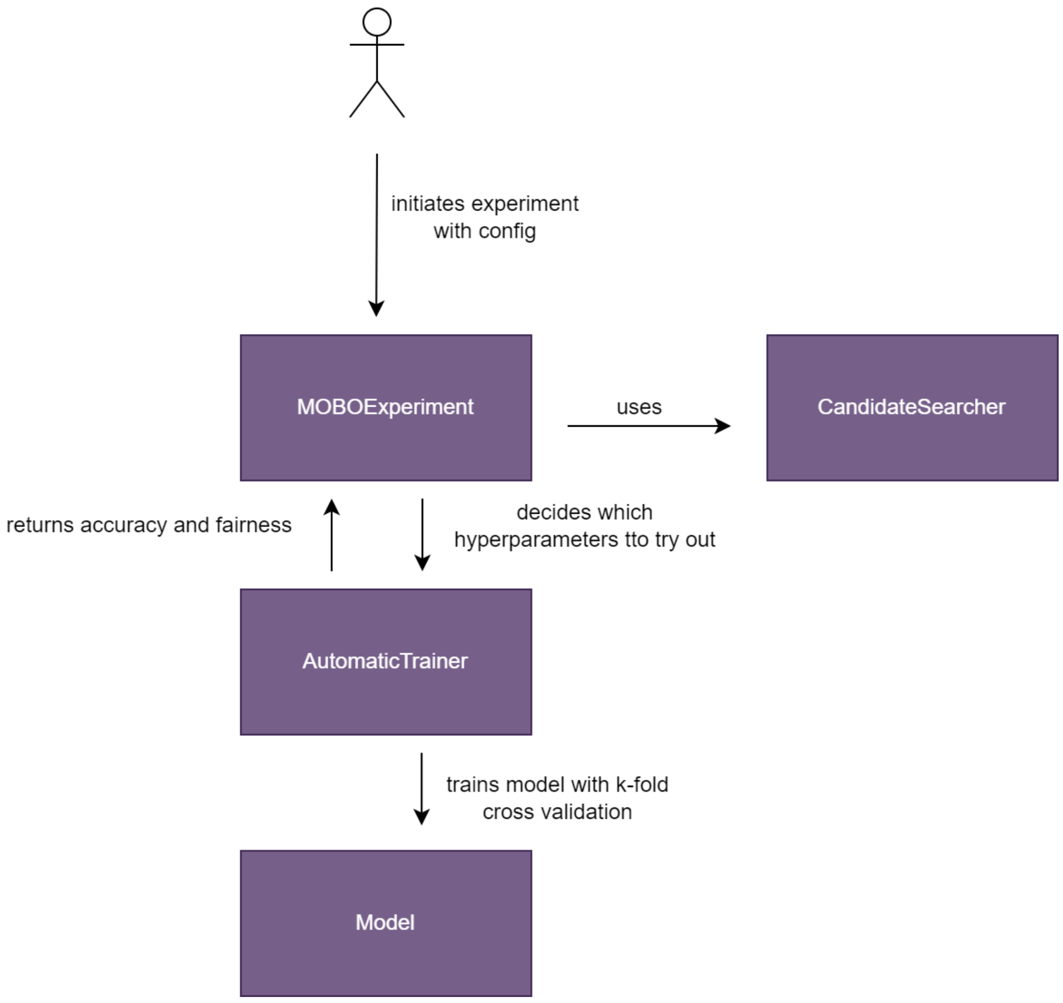

Figure 4.3: Diagram of the algorithm implementation

The configuration of an experiment is defined in a yaml file, requiring the following parameters to be specified:

- Acquisition: the possible values are random, qehvi, nqehvi, qnparego. It defines which candidate searcher we will use to decide which is the next point to try out inside a MOBO experiment. The `RandomCandidateSearcher` will choose the next point randomly, and the `BOCandidateSeacher` will use Bayesian Optimization. There are three types of BO candidate searchers, depending on the acquisition function used.

- Dataset: this is the dataset we will use to train the model. Possible values are adult_census, german_credit, and communities_crime.

- Device: whether the model will be trained in cpu or gpu.

- init_points: the number of points that will be sampled randomly before using the candidate searcher to decide which point to try out next.

- input_vars: which model hyperparameters will be optimized.

- n_experiments: the number of times the `MOBOExperiment` will be repeated with the same configuration. The purpose is to be able to prove that the results obtained are statistically significant.

- n_iterations: the number of candidates that we will try.

- n_points: the number of candidates that we will choose in each iteration of the BO optimization, i.e., this is the number of points the `CandidateSearcher` returns each time it is called.

An example of this initial configuration described in a yaml file and necessary to run an experiment is shown in Appendix A.

4.5 Tools

This project is mostly empirical, although it requires some theoretic concepts. Moreover, the project is software-based, an important part of this dissertation was writing in Python in order to program the algorithm, run different experiments that will be explained in section 5, and analyze the results obtained.
That is why the main tool used was a laptop with an IDE (Integrated Development Environment) that allows Python code to be written and executed. In particular, I used the PyCharm IDE. To keep track of the changes and make sure all the code worked properly as the size of the project increased, I used git for version control.

4.6 Obstacles encountered

One of the problems encountered during the experiments is that it took too much time to run an iteration. The cause was the sum of several factors, including the fact that we have used k-fold cross-validation and that each experiment is run 100 times to make sure the results are statistically significant.

To alleviate this problem, we decided to include support for the use of a GPU. This was relatively straightforward because we have used the PyTorch module to create the experiments. Even the BO library, BoTorch, is based on PyTorch. PyTorch includes, by default, support for NVIDIA GPUs, among others. After including support for CUDA (the software which controls NVIDIA GPUs), the experiments were trained on a laptop with an NVIDIA GeForce GTX 1060, which has 6Gb RAM. The time taken to run an experiment decreased considerably.

Chapter 5

Experiments

This chapter deals with the results of implementing the algorithm explained in the previous section. But firstly, in section 5.1, we will briefly discuss the datasets where the algorithm is going to be tried. Secondly, in section 5.2, it will be proven that there exists an inverse relation between the performance and fairness metrics, which is a necessary assumption for this dissertation. In the subsequent sections, we will explain and comment on different experiments carried out.

5.1 Datasets

In this first section, we will introduce two different and well-known datasets (sections 5.1.1 and 5.1.2), discuss their baseline models (section 5.1.3) and explain briefly some important steps carried out when preprocessing this data (section 5.1.4).

5.1.1 Adult census dataset

This first dataset is very famous and can be found in the UC Irvine Machine Learning repository [12]. After reading the data, we will transform the column names by applying the `strip` function. This is important because the names have trailing and leading spaces, which could cause us problems later.

The dataset has a total of 32 561 rows and 15 columns. The columns can be seen Table 5.1, as well as their type. The input columns are those that we will use in our models. The target column is what we are trying to predict. In this case, it is the whether the income of a person is greater or lower than 50k. Therefore, it is a binary variable (the only possible values are 0 and 1). Finally, the protected columns are those that we consider would be unfair to use. We will not use them in the models, but we will keep them so that we can analyze later whether our

Feature	Type
age	protected
workclass	input
fnlwgt	input
education	input
education-num	input
marital-status	input
occupation	input
relationship	input
race	protected
sex	protected
capital-gain	input
capital-loss	input
hours-per-week	input
native-country	protected
income	target

Table 5.1: Features of the Adult Census dataset

model is fair or not, by comparing results for different groups of a protected column.

Performing a data exploration analysis of the dataset, we have decided to plot pie charts showing different members of a protected class (in this case sex) for both positive and negative values of the target variable (income). These charts are shown in Figure 5.1. It is remarkable how there are huge differences between the sexes. The percentage of women is way smaller in the high-income class with respect to the low-income one. This can be a source of unfairness in itself. The model will have a tendency to predict with more probability that a male has a high income than a woman, even if all the other variables are the same.

Even if we remove the sex variable from the model, there could be other features that have a relationship with sex. We have plotted in Figure 5.2 the most correlated variables with the sex variable. We observe that some of them have significant values. However, the most correlated variable is *Relationship*, which is also a protected characteristic. Therefore, this variable will not be included in the model.

To find out whether in this dataset the model will be able to infer the sex of a person with the input variables (not including the protected classes) we have trained several simple models: a logistic regression, an SVM, a decision tree, and a

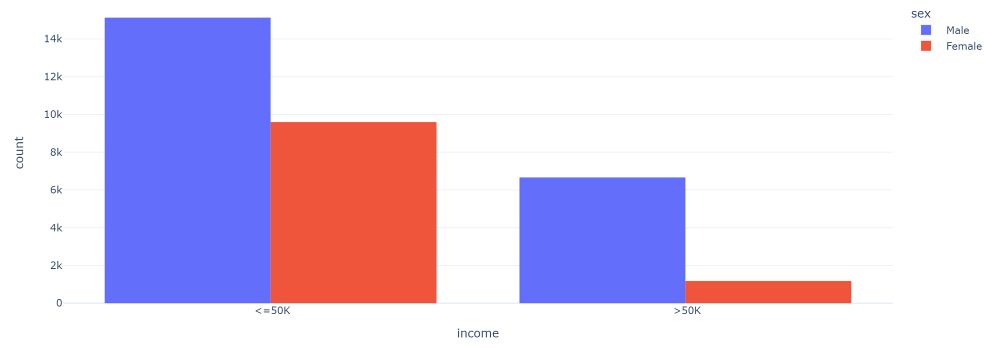

Figure 5.1: Distribution of target variable by sex of the Adult Census dataset

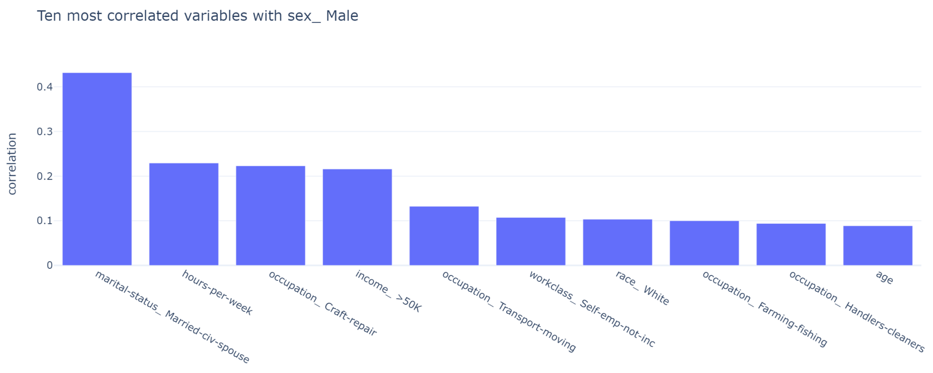

Figure 5.2: Correlation with the sex variable

neural network with two hidden layers of size 100 each. We have used the `sklearn` library to the models. After training them, we have calculated the accuracy of the test set, which is shown in Figure 5.3. If we use a random model (i.e., we always predict the most frequent sex), we obtain an accuracy of 66.80%. However, using the trained models, we have obtained accuracies above 80%. The maximum has been achieved with the SVC, obtaining 83.92%. As a result, we have proven that the non-protected variables of this dataset contain information about the sex of the person. It is not enough to exclude protected classes from machine learning models to guarantee that they are fair.

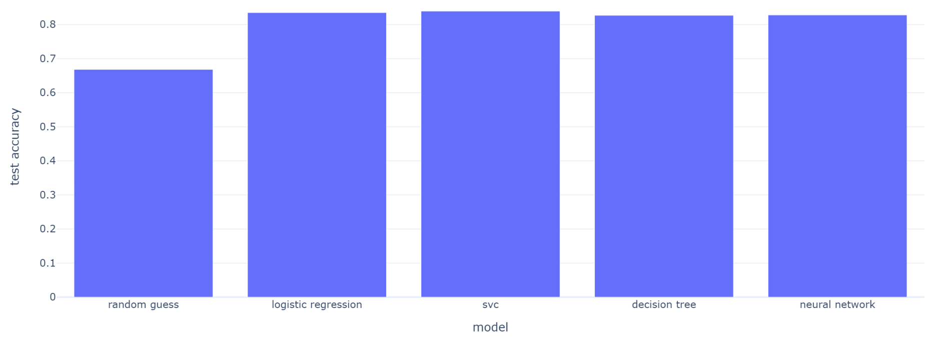

Figure 5.3: Accuracy of models that predict sex using the input variables of the Adult Census dataset

5.1.2 German credit dataset

Another dataset we have used in this dissertation is the German Credit dataset, also taken from the UC Irvine Machine Learning Repository [12]. This dataset has 1000 rows and 22 columns, shown in Table 5.2.

In this dataset, we also have several protected variables: sex, marital status, age, and foreign worker. We doubted whether to include the last variable as a protected variable. We finally decided to do so, as it was similar to the native country of the Adult Census dataset.

In Figure 5.4, we have shown the count of the dataset by the target variable and sex.

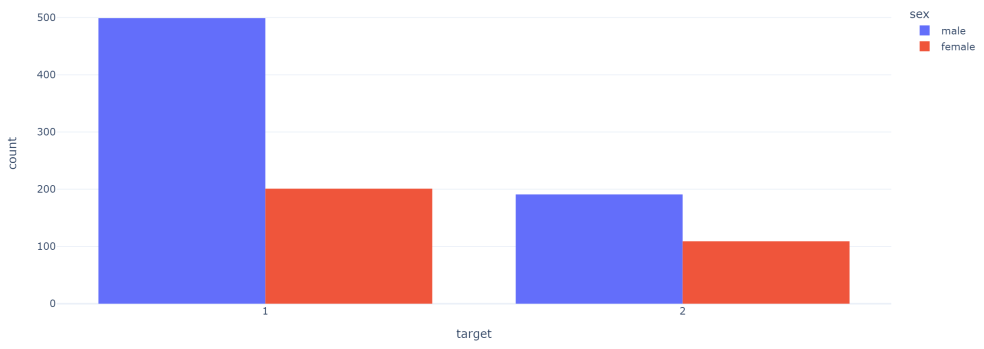

Figure 5.4: Count by target variable and sex of the German Credit dataset

Feature	Type
checking-balance	input
duration	input
credit-history	input
purpose	input
credit-amount	input
savings-balance	input
employed-since	input
installment-rate	input
personal-info	deleted
debtors	input
residence-since	input
property	input
age	protected
other-installment-plans	input
housing	input
n-credit-cards	input
job	input
n-people-liable	input
telephone	input
foreign-worker	protected
sex	protected
marital-status	protected
target	target

Table 5.2: Features of the German Credit dataset

We get the same finding as we previously obtained with the Adult Census dataset: there is a significant difference in the percentage of males when the target variable takes different values.

Similarly, we have created several models to predict the sex variable using the input variables of the German Credit dataset. The results are shown in Figure 5.5. We observe that there is not a significant difference between the random guess model and the other models. In fact, two models (logistic regression and svc) achieve a slightly higher accuracy than the random guess, while the two others (decision tree and the neural network) obtain a lower score. As opposed to the case of the Adult Census dataset, in this case we cannot claim that we can predict the sex of an observation using the other variables. It is a good thing that we have obtained different results with each dataset, as we will later be able to evaluate the impact of this in the success of our algorithm.

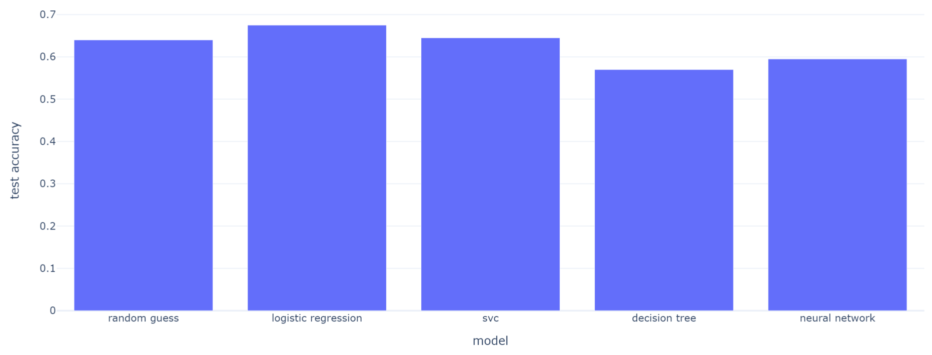

Figure 5.5: Accuracy of models that predict the sex variable using the input variables of the German Credit dataset

5.1.3 Baseline models

Before designing the algorithm and running experiments, we will create some baseline models to find out what accuracy can be achieved with our datasets. This is similar to the previous analysis to find out if there was a relationship between the input variables and the sex. In this case, we want to predict the target variables.

In Table 5.3, we show the results from these experiments. We observe that the Adult dataset obtains an accuracy of around 85%, while the german dataset models obtain an accuracy of around 71%. Both can improve the accuracy obtained when guessing the most common class in the training dataset. When implementing any kind of fairness algorithm, it is important to take these baseline values into account to be aware of the accuracy we are losing because of the fairness constraints.

Model	Adult Census	German Credit
Random guess	76.09 %	64%
Logistic regression	85.57%	71.7%
Random forest	83.77%	71.5%
SVM	85.75%	70.5%

Table 5.3: Validation accuracy achieved in the baseline models

5.1.4 Data processing

Each dataset that is added to our python module must have its own class which inherits from `datasets.base_dataset.BaseDataset`. The `BaseDataset` is an abstract class that implements `torch.utils.data.Dataset`. When adding your own dataset, you must implement the `_load(self)` method, which is called at the end of the constructor and is responsible for populating the class variables.

In the current section, we will explain the `_load(self)` method we have created for the Adult Census dataset. This method is very similar to that of the German Credit dataset. The first step in the function is to load the csv file that contains the data. The path to the file is a class variable that was saved in the constructor. Next, we find out which columns are categorical and which are numerical. To do this, we use the `df.select_dtypes(...)` method. Finding out this information is necessary because we will apply different transformations to each type.

These columns will later be the input that we will use in the models. Therefore, we must remove the protected columns and the target columns because we do not want to use them as input.

To process the numerical columns, we will standardize them. This is done by applying the following equation (5.1) to each column:

$$\boldsymbol{x_{scaled}} = \frac{\boldsymbol{x} - mean(\boldsymbol{x})}{std(\boldsymbol{x})} \tag{5.1}$$

However, we do not need to program this manually. We can simply use the `StandardScaler` class from `sklearn`.

To process the categorical columns, we can use the `pd.get_dummies(...)` function. This function converts each column to a set of binary columns. The number of columns in this set equals the number of categorical classes of the original column minus one. For example, if a categorical column has three possible values, which are A, B, and C, two columns will be created. A, B, and C will be converted to (0, 0), (1, 0), and (0, 1), respectively. If we had not used the `drop_first=True` attribute, A, B, and C would have been converted to (1, 0, 0), (0, 1, 0), and (0, 0, 1) instead.

One important consideration to take into account in this phase is that the dataset used to train the model must contain all possible values that the categorical column can take. For example, if the training dataset has a country column that only has values corresponding to Spain and Italy, and we later want to use the model with another country as input, it will not work because it will not be possible to encode that country.

The next step is to process the target. We will use a `LabelEncoder` from `sklearn`. This class will convert the possible values of the target to a set of integers. In our case, we are doing binary classification so the output will be 0 or 1.

Now, we will create the X variable, which contains the dataset as we will train it. This is done by concatenating the processed numerical and categorical data from above. Lastly, we will convert it to a `pytorch` tensor, as we will use this module to train our models. We must also do the same to the target column.

Although we will not use the protected columns (with sex, race, etc) to train the models, we do need to process them, as we will use them to analyze the fairness of our model.

5.2 Proving the hypothesis

The algorithm we are proposing tries to find a trade-off between a performance metric and a fairness metric. Therefore, the first thing we must do is prove that, often, there is an inverse relationship between these metrics, i.e., when a model works better, it is less fair.

To prove this hypothesis, we have run the following experiments:

1. We have created and trained a logistic regression model to predict the target variable as a function of the input variables, not including the protected ones.

2. We created 100 thresholds linearly between 0 and 1, and evaluated the two metrics (accuracy and difference in TPR) for both of them.

3. We have created a scatter plot showing the each variable in an axis. The plot is shown in Figure 5.6. Additionally, we have drawn the trendline.

In the figure, we can clearly see that there is a positive relation between accuracy and difference of TPR. Accuracy is a measures performance, while the difference in TPR is a measure of unfairness. Therefore, when the model has a higher accuracy, it is also less fair.

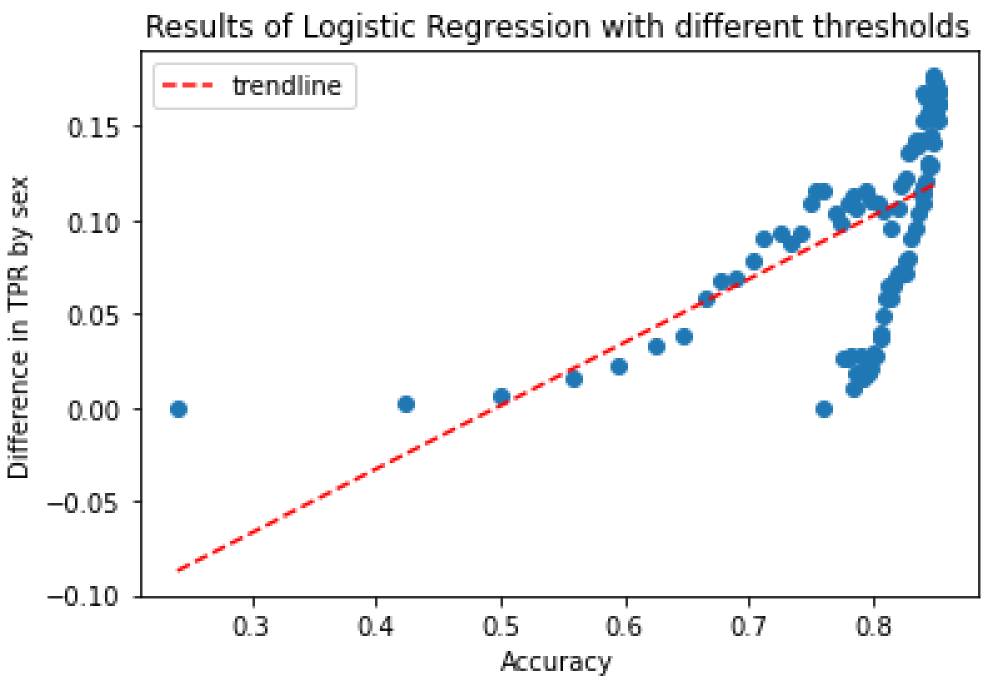

Figure 5.6: Trade-off between accuracy and difference in TPR

We will also observe in this trade-off in the experiments we will run in the later section. However, we decided to run this small experiment to make sure we were working in the right direction and avoid spending time creating Bayesian Optimization experiments if there wasn't anything to optimize.

5.3 Experiments on the Adult dataset

The first experiments were conducted on the explained adult dataset (section 5.1.1). They are divided into two blocks: the first one, tuning the learning rate (lr) and the dropout variables (section 5.3.1) simultaneously, and the second one, tuning the hidden layers' size (section 5.3.2).

5.3.1 Experiment 1: Tuning learning rate and dropout

In this first experiment, we have run the algorithm with the Adult dataset. We have tried to optimize two parameters: learning, and dropout. All the configuration of this experiment is shown on Table 5.4, where it can be seen that the input variables to tune are the learning rate and the dropout.

Hyperparameter	Value
Dataset	Adult
Initial points	5
Iterations	15
Number of experiments	100
Epochs	2
Input variables	lr, dropout

Table 5.4: Configuration of the first experiment

Let's jump into the results obtained. In Figure 5.7, we can see the percentage of times that Bayesian Optimization performed better than random guessing for different acquisition functions (at a 95% confidence interval).

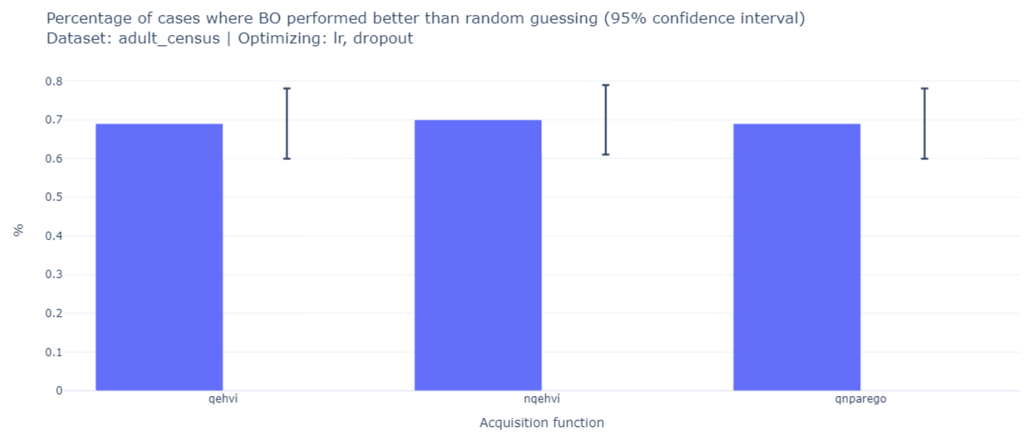

Figure 5.7: Success of experiment one

It can be concluded that for all the three different studied acquisitions functions (i.e., qEHVI, qNEHVI and qNParEGO), applying Bayesian Optimization gave better results than applying random guessing approximately 70% of the time. These results are somewhat expected since the BO model tries to optimize the choice of the two hyperparameters (learning rate and dropout) in an intelligent way, but chance and randomness can sometimes give better and even optimal results.

The results of the experiment can also be seen in a Pareto front (Figure 5.8), where the three different acquisitions functions and the random guess are compared in terms of accuracy (x-axis) and fairness (y-axis).

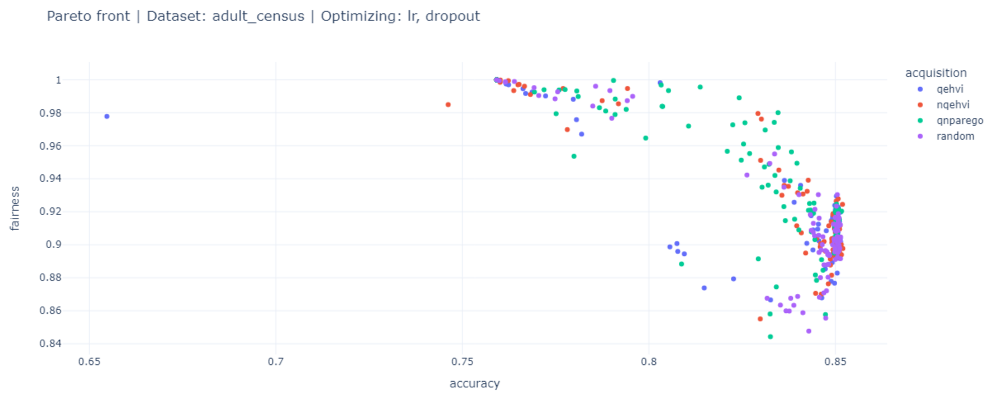

Figure 5.8: Pareto front of the experiment one

Two main conclusions can be extracted from the Pareto representation shown above. The first one is that random guessing (purple dots) has the worst results in terms of fairness, as many of its points are on the bottom right side of the graphic. The second one is that, in our search for a good trade-off between the two studied variables, fairness and accuracy, the acquisition function qNParEGO (green dots) seems to be the best one, drawing almost a continuous curve line among the different experiments.
The other two acquisition functions (red and blue dots representing qEHVI and qNEHVI, respectively) have a good performance, better than random guessing in terms of fairness, but they are not capable of finding a good trade-off among both variables as qNParEGO, which has many observations in the right top part of the diagram, meaning that both variables, fairness and accuracy, are high.

Figure 5.9 shows the hypervolume evolution in 6 random iterations among the 100 performed in each experiment. The x-axis represents the initial random points (5), and the number of iterations (15), and the evolution of each acquisition function and random guess is represented in different colors shown in the legend. In general terms, it can be seen that when applying BO, the volume improves constantly. However, when random guessing the parameters, the volume is more irregular, staying for long periods of iterations with similar values as it does not learn anything on each iteration (e.g., experiment 65).

We must not forget that this is the least meaningful of the three figures shown in this section, as it only shows 6 experiments of the 100 carried out. We cannot extract significant conclusions from Figure 5.9, and we only draw it to have a more detailed insight of what is happening inside the algorithm.

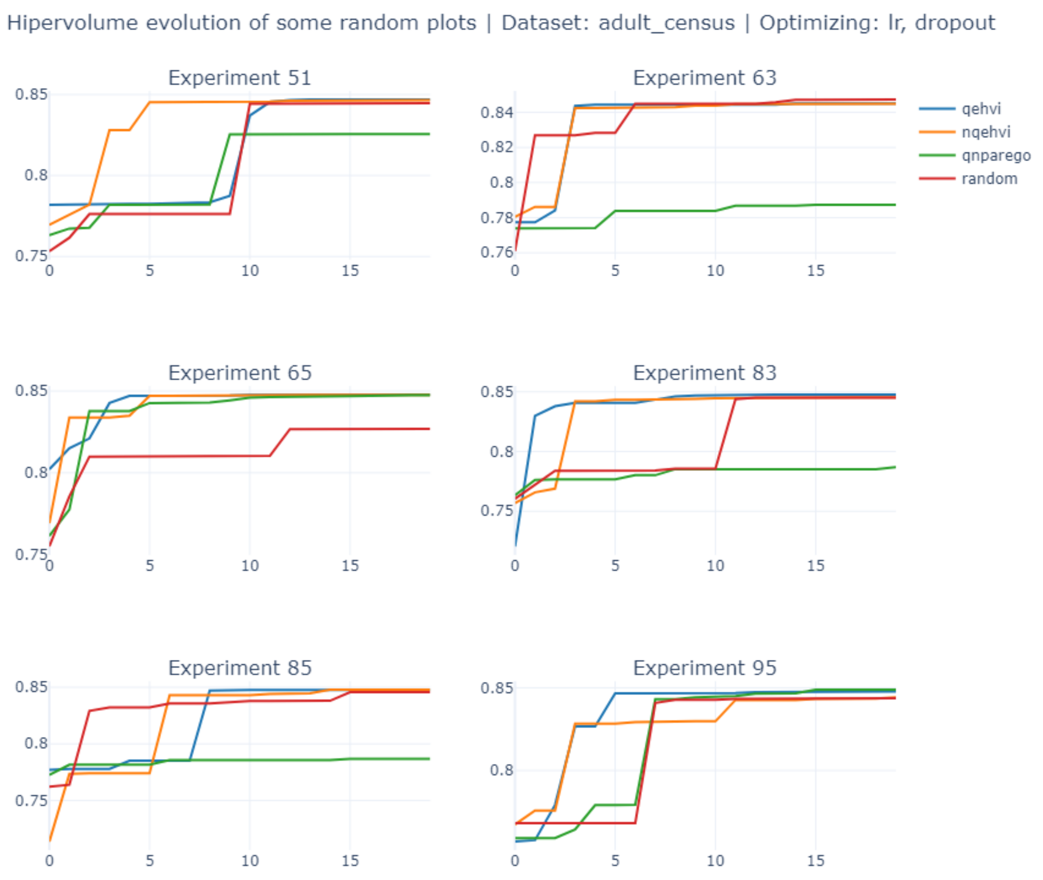

Figure 5.9: Hypervolume evolution of some random experiments in experiment 1

5.3.2 Experiment 2: Tuning hidden layers' size

In this second experiment, we had the same configuration as in the previous section but tried to tune the hidden layers' size instead of the learning rate and the dropout. This is controlled with the n1 and n2 parameters.

We will show the same three graphs as in experiment number one so that they can be compared in equal terms: the percentage of cases where applying BO outperforms random guessing (Figure 5.10), the Pareto representation for the different acquisition functions (Figure 5.11) and, finally, some random experiments to see the hypervolume evolution (Figure 5.12).

The first one is the number of times that applying BO works better than random guessing. The values show that, in this case, when tuning the size of the hidden layers, the percentage of BO outperforming random guessing is even higher than when tuning other parameters like in experiment 1 (learning rate and dropout). Furthermore, it can be seen that when applying qNEHVI, that percentage is around 93%, which is much higher than the 70% of experiment 1.

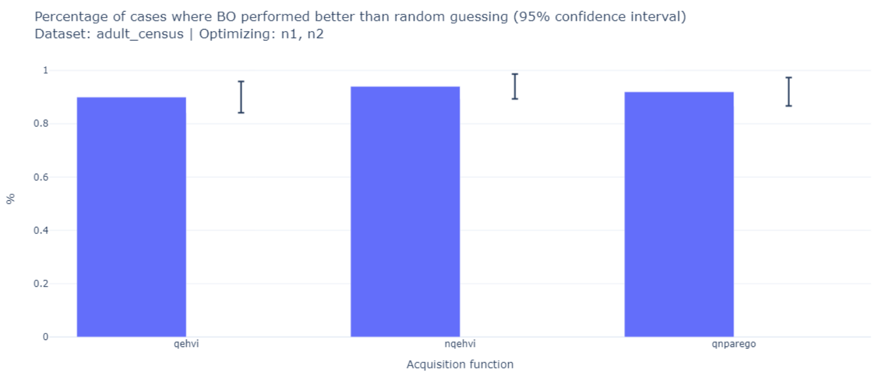

Figure 5.10: Success of experiment two

The second representation is the Pareto front, shown in Figure 5.11. This is not such a clear representation as the one obtained in experiment one, but it can also be concluded that the acquisition function qNParEGO, represented by the green dots, is the one with the best performance as it has many observations on the top right part of the graph (high fairness and high accuracy) and not that many in the bottom left part, especially when compared with the other acquisition functions. Again, most of the purple dots (random guessing) are in the bottom left part of the representation, obtaining the worst possible results in both variables. In this experiment, it is remarkable how hard it is to obtain a good result in terms of

Finally, 6 random experiments among the 100 carried out are shown in Figure 5.12. In this case, it can be seen clearly that random guessing is always the option that has the lowest hypervolume when achieving the last iteration (20) for five out of the six chosen experiments. In the other one, experiment number 33, random guessing is the second worst option.

5.4 Experiments on the German Credit dataset

The same two experiments carried out in the Adult dataset are replicated in the German Credit dataset. Thus, results can be compared, and the conclusions extracted will be stronger. Section 5.4.1 will show the results when tuning the

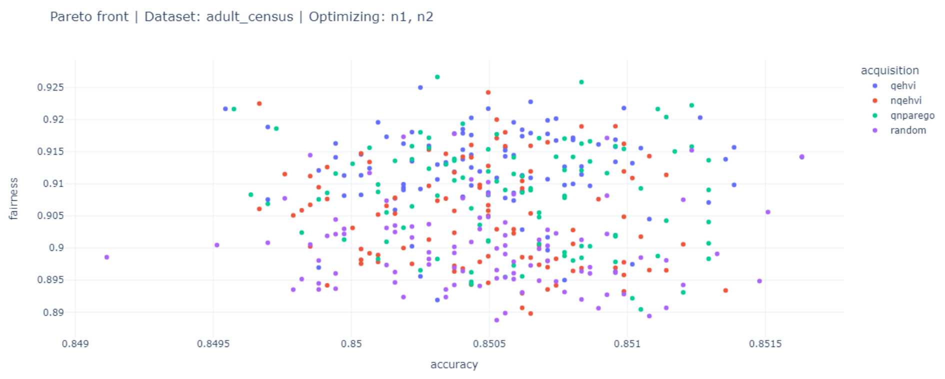

Figure 5.11: Pareto front of experiment 2

learning rate and dropout and section 5.4.2 will do the same when tuning the hidden layers' size.

5.4.1 Experiment 3: Tuning learning rate and dropout

Experiment 3 is equivalent to experiment 1 but with this new dataset (German Credit). Therefore, the three figures and results shown are equivalent to those seen earlier.

Figure 5.13 shows the percentage of cases where applying BO brought better results than applying random guessing. Although this time, the acquisition function qEHVI had lower results than in the previous dataset (Figure 5.7), the other two functions had similar results, always applying BO being better than random guessing.

The qEHVI acquisition function in this experiment is the only case where we have not proved with 95% confidence that it works better than random guessing. This is because we can see that the lower bound of the confidence bracket is below the 50% line. If we wanted to make the range of the confidence interval smaller, we would need to run more experiments.

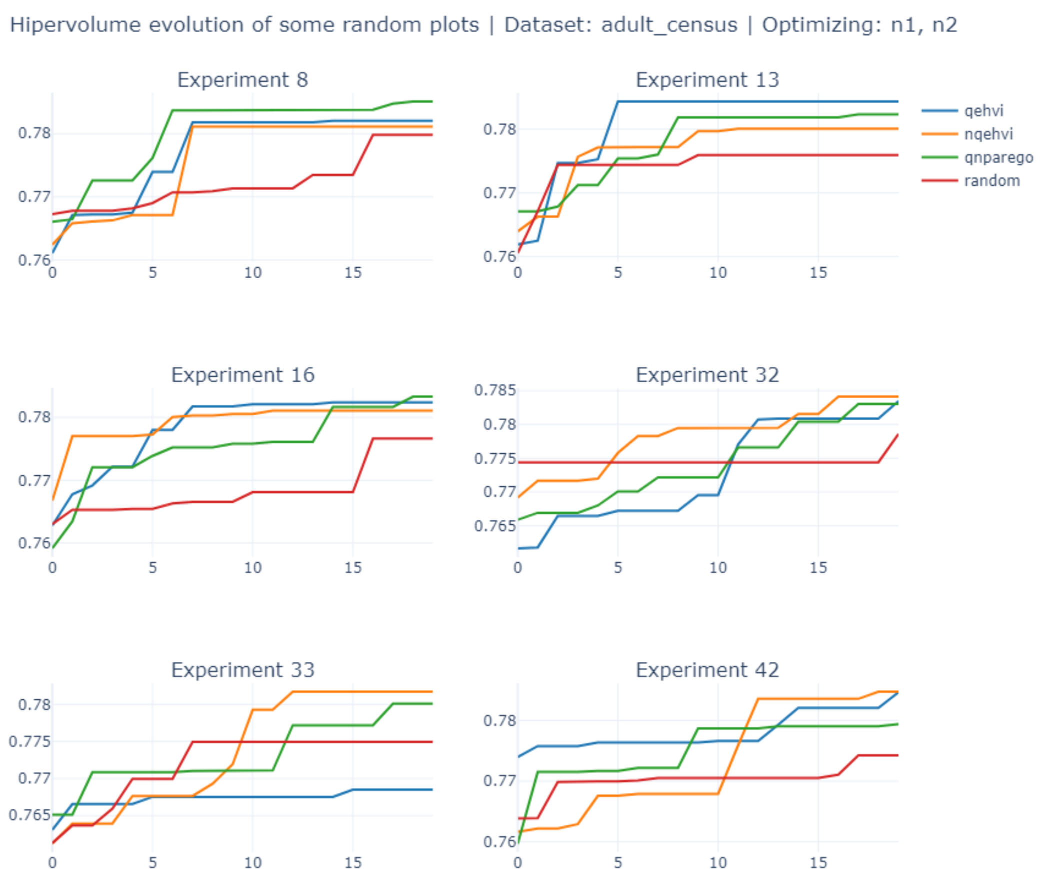

Figure 5.12: Hypervolume evolution of some random experiments in experiment 2

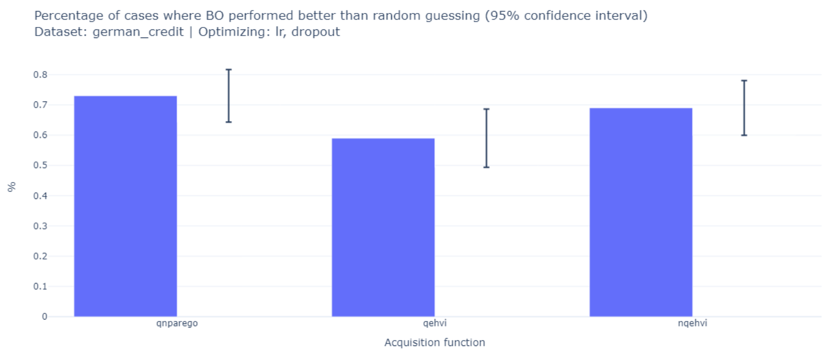

Figure 5.13: Success of experiment 3

The second representation that can be analyzed is the Pareto front (Figure 5.14). In this case, most of the observations are on the top right of the graph, meaning good results. However, it can be seen that most red dots (random guessing) are outside that bubble of good results, having many purple dots (NQEVHI) representing good experiments.

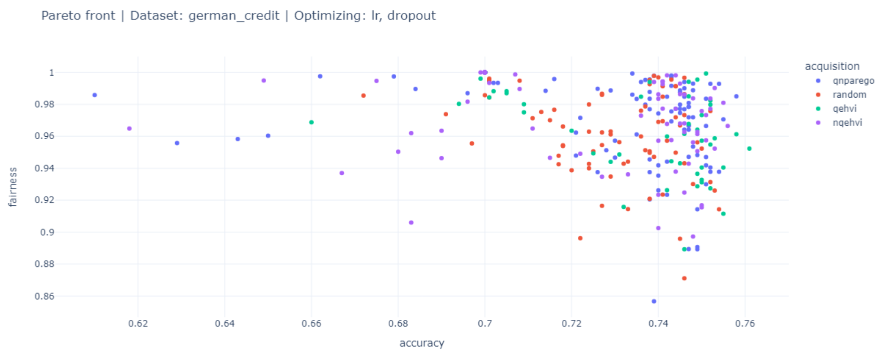

Figure 5.14: Pareto front of experiment 3

In general terms, it can be stated that this dataset had worse accuracy results than the adult dataset (around 85%, (Figure 5.8), whereas fairness results are in both cases similar and good.

Finally, in Figure 5.15, the hypervolume evolution of six random experiments for all the different acquisition functions and random guessing is shown. As expected, in most cases, random guessing gets worst or similar results as when using BO.

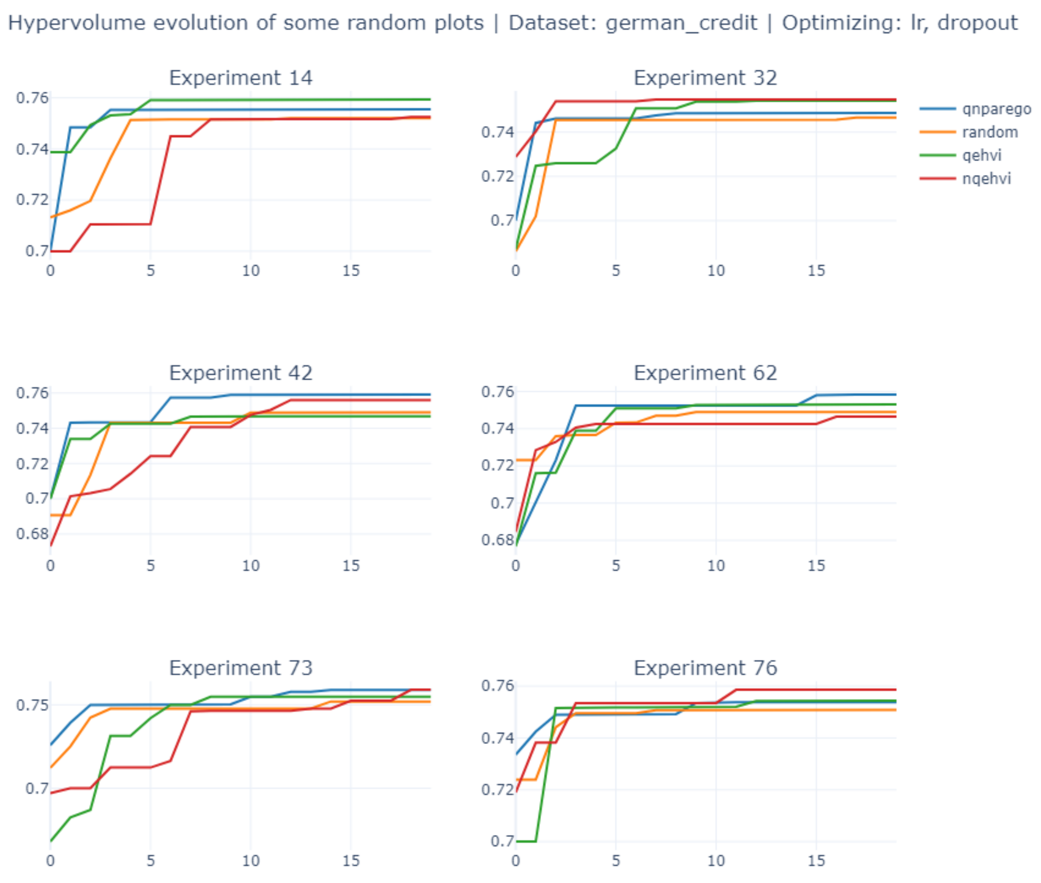

Figure 5.15: Hypervolume evolution of some random experiments in experiment 3

5.4.2 Experiment 4: Tuning hidden layers' size

Finally, experiment 4 tunes the hidden layers' size in this German credit dataset. The same idea as in experiment 2 in the adult datasset (subsection 5.3.2).

Figure 5.18 shows the percentage of cases where BO performed better than random guessing for the three different acquisition functions. Although in the three cases, BO performs better, it is true that when comparing the percentage of success now and in Figure 5.12 in the Adult dataset, they are now lower.

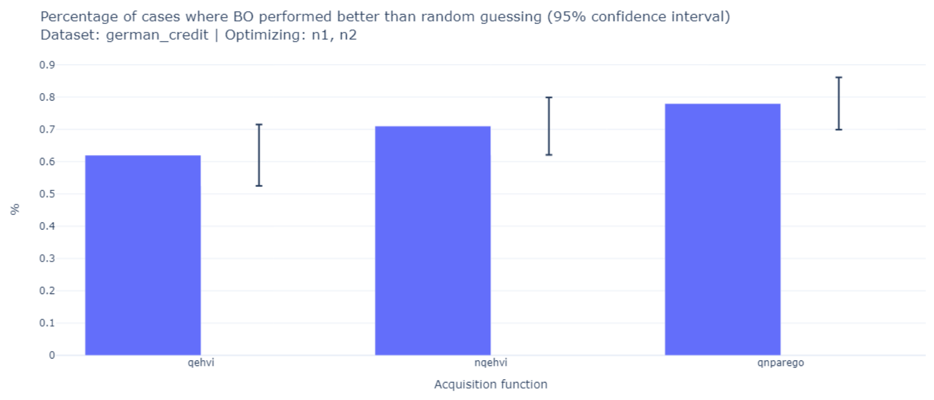

Figure 5.16: Success of experiment 4

When observing the Pareto front representation (Figure 5.17), it can be seen that green dots (qNParEGO) outperform all others. Random guessing is represented by purple dots, the worst scenario.

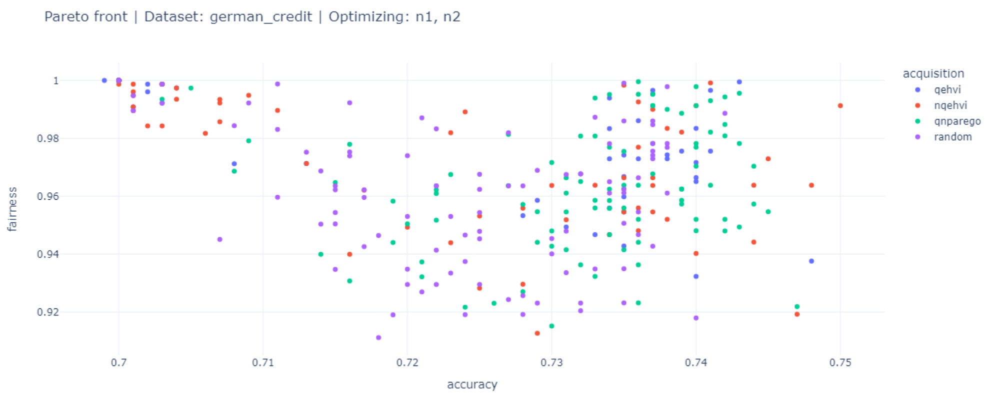

Figure 5.17: Pareto front of experiment 4

Finally, in Figure 5.18, the hypervolume evolution of some random experiments is shown. It can be seen that red lines representing random guessing usually have the worst metrics.

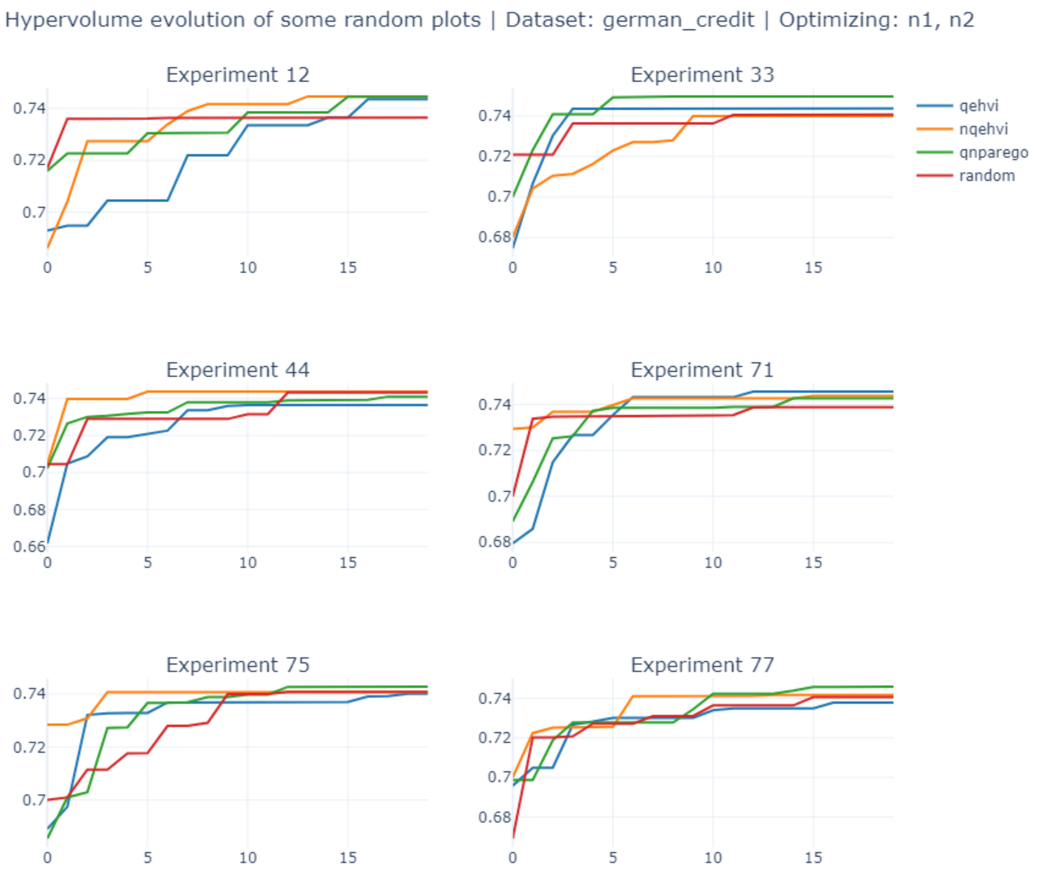

Figure 5.18: Hypervolume evolution of some random experiments in experiment 4

Chapter 6

Conclusions and next steps

This dissertation has helped extract some important conclusions about the use of Bayesian Optimization (BO) when referring to the selection of hyperparameters for Deep Learning models with the objective of maximizing both accuracy and fairness. They can be summarized in:

- First and most importantly, we have demonstrated that, in most cases, Bayesian Optimization is better than random guessing to find optimal hyperparameters for our use case (accuracy and fairness). BO achieves better results in fewer iterations. Proving this fact was the ideal outcome of the dissertation. In the experiments we have run, BO performed better than ramdom guessing in between approximately 60% and 95% of experiments. This number depends on the dataset, the acquisition function, and the hyperparameters being optimized.

- Secondly, we have studied three multi-objective acquisition functions. We conclude that, in most cases, qNParEGO is the one that achieves the best results (i.e., the Pareto set is the most spread out, and dots are not concentrated in a single area). The qNEHVI acquisition function also achieves very good results, almost as good as qNParEGO in most cases. Lastly, we obtain the worse results with the qEHVI acquisition function.

- The only experiment where we could not prove (with 95% confidence) that BO is better than random guessing is when optimizing the learning rate and dropout with the German Credit dataset, and using the EHVI acquisition function. In this case, the lower bound of the confidence interval (measuring the percentage of cases where we obtain better results than random guessing) was slightly below 50%. Although the mean was almost 60%, we cannot claim confidently that, in this case, BO is better than random guessing.

- We have proven our hypothesis with two datasets to guarantee that it is a general solution that is applicable to many cases. Additionally, on the public Github repository, users may also find a third dataset implemented, called Communities and Crime.

Therefore, the overall feeling about the project is very positive because it has been able to demonstrate in an effective and robust way what we had hypothesized. However, this project can be seen as just the beginning of more advanced work and applications of Bayesian Optimization. Some of the experiments that could be carried out in the future are outlined below:

- Increasing the range of hyperparameters that can be optimized. We could even optimize a full model architecture, including, not only the number of neurons in a layer but also the depth of the model or the number of filters if it is a convolutional neural network.

- We could try out other fairness measures. In this project, we have used the difference in true positive rate, but there are others. This should especially be considered if we do not want to achieve equal opportunity, but instead aim towards equalized odds or statistical parity.

- Finally, we could also allow the BO algorithm to suggest more than one point in each experiment. We should analyze whether this brings a decrease in the time taken to achieve same-quality results.

Bibliography

[1] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: (May 2020).

[2] OpenAI. "GPT-4 Technical Report". In: (Mar. 2023).

[3] K. Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6 (2 Apr. 2002), pp. 182–197. ISSN: 1089778X. DOI: `10.1109/4235.996017`.

[4] Qingfu Zhang and Hui Li. "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition". In: *IEEE Transactions on Evolutionary Computation* 11 (6 Dec. 2007), pp. 712–731. ISSN: 1941-0026. DOI: `10.1109/TEVC.2007.892759`.

[5] Peter I Frazier. "A Tutorial on Bayesian Optimization BayesOpt consists of two main components : a Bayesian statistical model for modeling the objective". In: *arXiv* (Section 5 2018), pp. 1–22.

[6] Carl Edward. Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006, p. 248. ISBN: 026218253X.

[7] Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. "A Flexible Framework for Multi-Objective Bayesian Optimization using Random Scalarizations". In: (May 2018).

[8] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. "Inherent Trade-Offs in the Fair Determination of Risk Scores". In: (Sept. 2016).

[9] Valerio Perrone et al. "Fair Bayesian Optimization". In: (June 2020).

[10] Christos Dimitrakakis et al. *Bayesian fairness*. 2018. arXiv: `1706.00119 [cs.LG]`.

[11] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (Dec. 2014).

[12] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[13] Rich Zemel et al. “Learning Fair Representations”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 325–333. URL: `https://proceedings.mlr.press/v28/zemel13.html`.

[14] Alexander Keller, Stefan Heinrich, and Harald Niederreiter. *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Ed. by Alexander Keller, Stefan Heinrich, and Harald Niederreiter. Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-74495-5. DOI: `10.1007/978-3-540-74496-2`.

[15] Till Speicher et al. “A Unified Approach to Quantifying Algorithmic Unfairness: Measuring Individual & Group Unfairness via Inequality Indices”. In: (July 2018). DOI: `10.1145/3219819.3220046`.

[16] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. In: (June 2012).

[17] Bobak Shahriari et al. “Taking the Human Out of the Loop: A Review of Bayesian Optimization”. In: *Proceedings of the IEEE* 104 (1 Jan. 2016), pp. 148–175. ISSN: 0018-9219. DOI: `10.1109/JPROC.2015.2494218`.

[18] Antonio Candelieri, Andrea Ponti, and Francesco Archetti. “Fair and Green Hyperparameter Optimization via Multi-objective and Multiple Information Source Bayesian Optimization”. In: (May 2022).

[19] Robin Schmucker et al. *Multi-Objective Multi-Fidelity Hyperparameter Optimization with Application to Fairness*.

[20] J. Knowles. “ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems”. In: *IEEE Transactions on Evolutionary Computation* 10 (1 Feb. 2006), pp. 50–66. ISSN: 1089-778X. DOI: `10.1109/TEVC.2005.851274`.

[21] Rich Zemel et al. “Learning Fair Representations”. In: ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. PMLR, Apr. 2013, pp. 325–333. URL: `https://proceedings.mlr.press/v28/zemel13.html`.

[22] Rachael Hwee Ling Sim et al. “Collaborative Bayesian Optimization with Fair Regret”. In: ed. by Marina Meila and Tong Zhang. Vol. 139. PMLR, Apr. 2021, pp. 9691–9701. URL: `https://proceedings.mlr.press/v139/sim21b.html`.

[23] Xilu Wang et al. “Recent Advances in Bayesian Optimization”. In: (June 2022).

Appendix A

Example of an experiment configuration

Example of yaml file with the necessary parameters to run an experiment:

```yaml
acquisition: qehvi
dataset: adult_census
device: auto
init_points: 5
input_vars:
- lr
- dropout
n_experiments: 100
n_iterations: 15
n_points: 1
```

Appendix B

Github Repository

The software developed in this dissertation is publicly available on Github:
`https://github.com/jorgecalvar/fair-bo-dissertation`