# Theory and Applications of Machine Learning Fairness: Review from the Perspective of Fairness Tester *Aequitas*

Yemi Shin, Yunping Wang, Michael Worrell, and Juanito Zhang Yang
*Department of Computer Science, Carleton College, 300 North College Street, Northfield, MN, 55057*

(Dated: March 16, 2022)

In recent years, an increase in application for machine learning gave rise to the discipline of machine learning fairness. Machine learning fairness is the study that aims to reduce the inherent discrimination present in all models. This survey paper aims to explore the theory and application of machine learning fairness through the lens of the algorithm *Aequitas*. *Aquitas* is a preprocessing bias-correcting algorithm that implements individual and counterfactual fairness. To understand the algorithm We first address the theory of machine learning and give a overview of the landscape of fairness. After that, the algorithm *Aquitas* is studied in more detail, especially its mathematical background and code implementation.

## CONTENTS

# I. INTRODUCTION

In recent years, machine learning has gradually become an integral tool in our everyday lives. Machine learning is used to determine whether a bank should issue a loan to someone[1], a company should hire someone[2], and in certain extreme cases, how long someone's criminal sentence should be[3,4]. However, this up-ticking in range of application for machine learning is not necessarily a net positive for the society. In particular, since machine learning refers to the process of using real-world data to improve upon a computer program, pre-existing social biases in those data may carry through the resulting model. These biases may then be propagated further when biased models are used in real world applications, creating potential discrimination[5].

Thus, a new field of study is emerging that aims to counteract these biases, namely machine learning fairness, a discipline where we examine the intrinsic biases in machine learning models and devise algorithms to adjust our model to be resilient to these biases. One of these algorithms is call *Aequitas*, developed by Udeshi et al and presented in the paper *Automated Directed Fairness Testing*[6].

In this paper, we will give an overview of the field of machine learning fairness and explore *Aequitas* in more detail. In the first half of the paper, we will provide the necessary background information to understand the algorithm and its significance in the overall machine learning fairness landscape. In the second half, we will provide an overview and explanation of the *Aequitas* algorithm.

We will start our discussion by developing the theoretical framework of machine learning that can aid us in understanding *Aequitas*. In Section II, we will discuss the basic theory of machine learning. We will then develop the theory of training machine learning models in Section III. Building on that, in Section IV, we will discuss three machine learning models that are relevant to *Aequitas*: decision tree, random forest and support vector machines (SVM). This will conclude our excursion into the theory of machine learning.

Starting from Section V, we shift our focus to machine learning fairness. In this section, we aim to give an overview of the field of machine learning fairness by discussing different definitions of fairness and how they can be applied to various circumstances. We will also discuss common strategies that are used to enforce these types of fairness in a machine learning model. This section will provide background information for us to place *Aequitas* within the larger field of fairness and pinpoint the exact problem that it tries to address.

The last necessary piece of information that is needed will be the mathematical principle behind *Aequitas*'s operation. *Aequitas* leverages two main mathematical concepts in its implementation: robustness in machine learning model and the law of large numbers. We will discuss the rigorous details of machine learning robustness in Section VI and law of large numbers in Section VII.

After presenting all the relevant information needed to understand *Aequitas*, we will explore the algorithm itself in Section IX. The original author also provides a proof of concept implementation for the algorithm in Python, which we will discuss in the Appendix.

# II. FUNDAMENTALS OF MACHINE LEARNING

To start the discussion of the theory behind machine learning, we will need to define what learning is. Learning occurs when an agent improves its performance after making an observation about the world. An agent can make observations through a data set. A data set is a set of input-output pairs. In this paper, the input in the data set will be in factored representation. A factored representation refers to a vector of attribute values $(x_1, x_2, \ldots, x_n)$. As for the output, we will focus on two main types of output: classification and regression. A classification is when an output is one of a finite set of values, e.g., true/false. A regression is when the output is a continuous number[7].

We characterize the idea of learning through the concept of feedback. Specifically, we divide machine learning algorithms into three sub-categories: 1), Supervised Learning, 2), Unsupervised Learning, and 3), Reinforcement Learning. We call a learning process supervised if the agent observes input-output pairs and learns a function that best maps input to output. We call a learning process unsupervised if the agent learns patterns without explicit feedback from the programmer. Finally, we call a learning process reinforced if the agent learns from a series of reinforcement taking the form of rewards and punishment. In this paper, we will focus our attention on supervised machine learning.

## A. Supervised Learning

Here we provide a formal definition of supervised learning:

**Definition II.1** (Supervised Learning)**.** [7] Given a training set of $N$ example input-output pairs

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

where each pair is generated by an unknown function $y = f(x)$, discover a function $h$ that approximates the true function $y$.

We call $h$ the hypothesis function about the world (or model of the world), and we define the function space of all possible $h$ as the hypothesis space/model class $\mathscr{H}$. We also define the output $y_i$ as the ground truth.

With Definition II.1, we are equipped to define a consistent hypothesis:

**Definition II.2** (Consistent Hypothesis)**.** A hypothesis $h$ is consistent on a training set $N$ if $\forall (x_i, y_i) \in N$, $h(x_i) = y_i$.

To evaluate a hypothesis, we will test it on a test set $T$ that is different from the training set $N$. A hypothesis generalizes well if $h(x_j) = y_j \forall (x_j, y_j) \in T$.

To analysis how well a hypothesis $h$ generalizes, we will introduce two ideas: bias and variance, which in turn defines the concept of underfitting and overfitting.

Bias refers to the tendency of a predictive hypothesis to deviate from the expected value when averaged over different training set. In other words, a hypothesis that is high on bias has tendencies to ignore patterns in data sets. We say that the hypothesis is underfitting the data. Bias often results from strong restrictions placed over the hypothesis space $\mathscr{H}$. For example, a hypothesis space of linear function induces a strong bias as any hypothesis generated will only captures the overall slope of the data over time and lose any other patterns.

Variance refers to the tendency for the hypothesis to exhibit massive change from small fluctuations in the training set. Alternatively, a hypothesis with high variance pays too much attention to details that are specific to a particular training set and fails to generalize over unseen data. We characterize such hypothesis as overfitting. A typical example of a hypothesis space that tends to overfit is a high-degree polynomial.[7].

## III. DEVELOPING MACHINE LEARNING MODELS

The goal of machine learning is to select a hypothesis that will optimally fit unseen future examples. However, in order to parse this statement, we would first need to specify the meaning of optimal fit and the restrictions on future examples[7].

We first need to place a restriction onto the dataset. Here we are assuming that all data a randomly drawn from a population with a probability distribution $P$. We claim that all future examples must satisfies the stationary assumption. The definition of stationary assumption is stated below:

**Definition III.1** (Stationary Assumption)**.** We assume that for an arbitrary data set $E$, the following are true $\forall E_j \in E$:

i), All $E_j$ has the same prior probability distribution, or

$$P(E_j) = P(E_{j+1}) = P(E_{j+2}) = \dots;$$

ii), All examples are independent of the previous examples, or

$$P(E_j) = P(E_j | E_{j-1}, E_{j-2}, \cdots).$$

In other words, all examples need to be independent and identically distributed[7].

To define an optimal fit, we first introduce the concept of error rate, which is the proportion of time that $h(x) \neq y$ for an $(x, y)$ example. We define an optimal fit as the hypothesis that minimizes error rate. Since a model is based on a training set, we need to evaluate the model on an alternative data set to make sure that our model is not overfitting. Thus, we will need a test set comprised of different examples from the training set.

Before we proceed, we also need to define the idea of a hyperparameter. A hyperparameter is a parameter that determines how the model is generated. In other words, it is a parameter for the model training itself. Therefore, when generating a model, the programmer has control over the hyperparameters.

To summarize, we can write down the following simple process for supervised learning:

1. Use a training set to train the data
2. Adjust the hyperparameters for the model
3. Test the data on a validation set to see whether the adjustment improved the model or not
4. Repeat steps 1 - 3
5. Evaluate the final model on a test set

Note that the validation set here refers to a data set that is different from both the training set and the test set. The reason for needing a validation set is that we need to evaluate each model independently of the data set.

We can break down the task of finding a hypothesis into two sub-tasks: 1), Model selection, which refers to the process of choosing a hypotheses space, and 2), Optimization/Training, which finds the best hypothesis in the space. The study of model selection is beyond the scope of this paper is best addressed in textbooks such as Stuart Russell and Peter Norvig's *Artificial Intelligence: A Modern Approach*[7].

## IV. COMMON MACHINE LEARNING MODELS

In this section we will be discussing some common machine learning models and their training processes.

### A. Decision Tree

A decision tree is a Boolean classifier that maps a vector of attributes to true/false. In other words, a decision tree hypothesis takes the form

$$h : A^n \to \{true, false\}, \tag{1}$$

where elements of $A^n$ are lists with $n$ elements.

We can modify a Boolean classification into a decision tree using the following structure: a node in the decision tree is a test of a single input attribute. A branch is labeled with possible values of the attribute, and a leaf is a specific classification that the tree will return. A decision tree thus reaches its decision by passing through a series of tests, starting from the root until it reaches one of its leaves. In the remaining sections, we will consider any example with $true$ output as a positive example, and $false$ output as a negative example.

Note that we can rewrite a Boolean decision tree to the following equivalent logical statement:

$$Output \Leftrightarrow (Path_1 \vee Path_2 \vee Path_3 \vee \dots), \tag{2}$$

where each $Path_i$ is a conjunction of the form $(A_m = v_x \wedge A_n = v_y \wedge \dots)$ of attribute-value tests that correspond to a path from the root to a $true$ leaf. The whole expression is then in disjunctive normal form. Thus, any function of propositional logic can be turned into a decision tree.

To generate a decision tree from a training set, we will use a greedy divide-and-conquer algorithm. The main idea behind the algorithm is that we want to test the attribute with the largest information gain first, and then recursively solve the smaller sub problems. The full algorithm is given below:

1. If the examples are all positive or negative, we are done.

2. If there are both positive and negative examples, then choose the attribute with the highest *IMPORTANCE* to split them. Then recursively continue testing the subsets.

3. If the are no examples, it means that no examples are in the train set with this combination of attributes values, and we return the most common output value from the training set that were used in constructing the node's parent.

4. If there are no attributes, but still both positive and negative examples, it means that these examples have the same exact description, but different Boolean classifications. In other words, the model gave two identical example different classifications. This can happen due to error or noise in the data. The best we can do is return the most common output value from the remaining sets.

Now we need to define the *IMPORTANCE* function. To do that, we need to introduce the concept of information entropy. Entropy measures the uncertainty of a random variable. Thus, more information correlates to less entropy. For example, a random variable with one possible outcome has zero entropy. Formally, we define entropy of a random variable $V$ as

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k). \quad (3)$$

For a Boolean random variable that is true with probability $q$, we define its entropy $B(q)$ as

$$B(q) \equiv -(q \log_2 q + (1-q) \log_2 (1-q)). \quad (4)$$

Therefore, for a training set with $p$ positive examples and $n$ negative examples, the total entropy would be

$$H(Output) = B(\frac{p}{p+n}). \quad (5)$$

Moreover, an attribute $A$ with $d$ distinct values divides the training set $E$ into subsets $E_1, \ldots, E_d$. Each subset $E_k$ has $p_k$ positive examples and $n_k$ negative examples. Formally, we define $E_k$ as

$$E_k = \{e \in E : A(e) = A_k\}, \quad (6)$$

where $A(e)$ is the value of attribute $A$ for example $e$. Thus, if we know $A = A_k$, we still need additional

$$H(Output|A = A_k) = B(\frac{p_k}{p_k + n_k}) \quad (7)$$

bits of information to determine whether a given example is positive or negative.

A randomly chosen example from the training set would have the $k$th value for an attribute. In other words, it will be in set $E_k$ with probability

$$P(e \in E_k) = \frac{p_k + n_k}{p+n} = P(A = A_k). \quad (8)$$

Combining these two results, we can determine the entropy of the tree after we test for attribute $A$, or

$$Remainder(A) = H(Output|A). \quad (9)$$

We know that

$$H(Y|X) = \sum_x P(X = x) H(Y|X = x). \quad (10)$$

Using our previous results, we then have

$$H(Output|A) = \sum_{k=1}^{d} P(A = A_k) H(Output|A = A_k) \quad (11)$$

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B(\frac{p_k}{p_k + n_k}) \quad (12)$$

Now we can define the information gain from the attribute test on $A$ as the expected reduction in total entropy, or

$$Gain(A) = H(Output) - Remainder(A) \quad (13)$$

$$\Rightarrow \quad Gain(A) = B(\frac{p}{p+n}) - Remainder(A) \quad (14)$$

Finally, we can formally define a way to quantify the *IMPORTANCE* of attributes $A$ as $Gain(A)$.

However, a decision tree algorithm only finds a hypothesis that best fits the training data, when what we really want is to generalize for unseen data. With higher attribute count, we are much more likely to overfit. Therefore, we would need to introduce the concept of decision tree pruning. Pruning works by eliminating nodes that are clearly not relevant. We start with the full tree as last time, but this time we look at test nodes with only leaf nodes as their descendants. If a test is irrelevant, we replace the test node with a leaf node. Pruning continues until all test nodes with only leaf nodes as their descendants are either pruned or accepted as they are.

We now need to decide how to determine whether a test node is relevant. To do this, we use a statistical significance test. We start by assuming that there are no underlying patterns (Null hypothesis). We then compute to what extent the actual data deviates from a total lack of pattern. If the degree of deviation is statistically unlikely($> 5\%$), it signifies that there is a significant pattern in the data.

In our case, if we have $v = p + n$ examples, we can compute the expected positive $\hat{p_k}$ and negative $\hat{n_k}$ examples in each subset $E_k$ using the overall ratios of $p$ and $n$, and compare them to the actual $p_k$ and $n_k$. Thus, we define $\hat{p_k}$ and $\hat{n_k}$ as such

$$\hat{p_k} = p \times \frac{v_k}{v} = p \times \frac{p_k + n_k}{p+n}, \quad (15)$$

$$\hat{n_k} = n \times \frac{v_k}{v} = n \times \frac{p_k + n_k}{p+n}. \quad (16)$$

We can then compute the deviation using a $\chi^2$ test:

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k^{\,2}} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k^{\,2}} \tag{17}$$

This equation follows the $\chi^2$ distribution with $d-1$ degrees of freedom. Using this distribution, we see that $\Delta = 7.82$ would reject null hypothesis at 5% level, and value below are accepted. This technique is called $\chi^2$ pruning. With this, our construction of a decision tree is complete.

## B. Support Vector Machines

A support vector machines(SVM) is a non-parametric model that attempts to generate a maximum margin separator. A maximum margin separator is a decision boundary where every example on one side of the boundary will have the classification 1, and every example on the other side will have the classification $-1$. We need to develop an algorithm that generates such decision boundary. Note that since we do not constrain the dimension of the attribute space, our decision boundary will be a hyperplane defined by

$$\{\vec{x} : \vec{w} \cdot \vec{x} = b\}, \tag{18}$$

where $\vec{w}$ and $b$ are the coefficients that we need to find. We will also define the distance between the decision boundary and the nearest example as $d$. See Fig. 1 for a two-dimensional illustration of the problem.

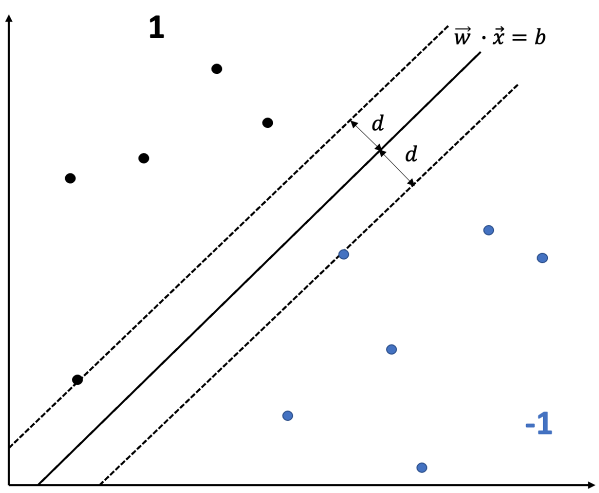


Figure 1. A two-dimensional SVM problem. The top half are the examples with classification 1, and the bottom half are the example with classification $-1$. The line in the center is the decision boundary with a separation of $d$ to the nearest examples.

Observe that

$$\forall \vec{x} \in \mathbf{1},\ \vec{w} \cdot \vec{x} > b + d, \tag{19}$$

and

$$\forall \vec{x} \in -\mathbf{1},\ \vec{w} \cdot \vec{x} < b - d, \tag{20}$$

where $\mathbf{1}$ is the set of examples with classification 1, and $-\mathbf{1}$ is the set of examples with classification $-1$.

We can now define $y_i$ such that for every $x_i$:

$$y_i = \begin{cases} 1, & x_i \in \mathbf{1}, \\ -1, & x_i \in -\mathbf{1}. \end{cases} \tag{21}$$

Then, we can combine Eq. 19, 20 and Eq. 21 to get

$$y_i(\vec{w} \cdot \vec{x} - b) < d. \tag{22}$$

We can also show that

$$d = \frac{2}{||\vec{w}||}. \tag{23}$$

Then, we can rearrange this inequality to get

$$y_i(\vec{w} \cdot \vec{x}_i - b) < 1 + e_i, \tag{24}$$

where $e_i$ is the error of the decision boundary, or the number of examples that are placed on the wrong side. Therefore, this problem has been reduced to a problem of finding

$$\min(\sum_{i=0} e_i + ||\vec{w}||^2). \tag{25}$$

This is a quadratic programming optimization problem, and the detail of such algorithm is beyond the scope of this paper. After solving this for $\vec{w}$, we will have our decision boundary.

## C. Random Forest

To discuss random forest classifier, we first need to introduce the concept of ensemble learning. Ensemble learning is the process of selecting a collection of hypotheses(base model) and combining them to create a joint-decision model. The main reason for using ensemble learning is that it reduces bias and variance in our hypothesis.

Random forest is a form of ensemble learning that is based on a technique called bagging. In bagging, $K$ distinct training sets are generated by randomly sampling $N$ examples with replacement from the original training set. A machine learning algorithm is then run on the sub-training set to get $K$ hypotheses. These hypotheses are aggregated to make the final prediction. For classification problems, we use plurality vote; for regression problems, we compute the average:

$$h(x) = \frac{1}{K} \sum_{i=1}^{K} h_i(x). \tag{26}$$

Bagging tends to reduce variance and is the standard approach when the base model overfits.

Random forest is a form of decision tree bagging in which we take extra steps to make the ensemble of $K$ trees more diverse. The key idea is to randomly vary the attribute choice rather than the training examples. At each point in the construction of the tree, we select a random sampling of $n$ attributes, and compute the one that yields the most information gain. Consequently, random forests are not pruned and are resistant to overfitting. As more trees are added to the forest, the error converges.

## V. FAIRNESS IN MACHINE LEARNING

It can be challenging to know what exactly machine learning fairness is. The field of fairness in machine learning has been growing in size, coverage, and importance. It has been growing into its own conferences, and many new papers on machine learning fairness are released each year.

When relying on the use of software to calculate outcomes for complicated decision making tasks, it is easy for unintentional bias and discrimination to make their way into more and more elements of everyday life. This could make it so that decision making processes thought of to be fair and unbiased could produce inaccurate and unfair decisions.

The consequences of machine learning making mistakes when it comes to fairness are important to consider when dealing with machine learning. One notable recent example is the COMPAS computer program, which was used to assist in sentencing defendants, which was eventually determined to be making unfair decisions. After filtering criminal history, COMPAS was 77% percent more likely to incorrectly flag black defendants as high risk than it was likely to falsely flag white defendants, and was also more likely to falsely identify white individuals as low-risk.[8]

These kinds of problems require technical and legal definitions of fairness, as well as definitive ways to improve the fairness of machine learning models. However, defining, measuring, and improving fairness is a long-standing problem that has been debated and discussed for decades, long before machine learning reached the prevalence it has today. One indicator of the current state of machine learning fairness is how fairness is determined legally. Some of the modern ways that laws evaluate the fairness of decision making processes are via the presence of *disparate treatment* and *disparate impact* present in or resulting from a particular decision making process.[9]

Disparate treatment is the direct use of sensitive attribute data in the decision making process, while disparate impact is the presence of negative impacts resulting from the decision making process. Attempting to circumvent either disparate treatment or disparate impact on their own, by methods such as not considering the sensitive attribute to avoid disparate treatment, or by using sensitive attributes to avoid disparate impact, can lead to other forms of discrimination (Indiscrete Discrimination and Reverse Discrimination being a few notable examples).[9]

### A. Defining Fairness

One thing we took away from our research is that one cannot guarantee the ability to maximize multiple conflicting definitions of fairness simultaneously. Because of this, those that measure machine learning fairness need to make a choice of how one will define the fairness of a given model prior to analyzing its fairness.[10]

Definitions of fairness attempt to help mediate a problem specific to their definition. There have been multiple definitions of fairness proposed throughout the past several decades. These definitions work to help individuals maximize different categories of fairness, made to provide fairness in a specific context . A few of these categories include individual fairness, non-comparative fairness, subgroup parity, and correlation.[10]

With Aequitas, we are catering to the category of individual fairness. Aequitas attempts to discover discriminatory input (inputs that violate individual fairness). Individual fairness attempts to make sure that, for two entries that are identical with the exception of some sensitive attribute (or identical within a specific input space), the fairness model will output identical results for both entries[6]. For Aequitas, the precise definition of (individual) fairness is as follows:[6]

**Definition V.1.** Let $f$ be a classifier under test, $\gamma$ be the predetermined discrimination threshold (e.g. chosen by the user), $\mathbb{I}$ be the input domain of the model, $P_i$ be the $i$-th input parameter of the model, $P$ be the set of all input parameters, $P_{disc}$ be the non-empty set of all potentially discriminatory parameters, $I \in \mathbb{I}$, and $I_p$ be the value of input parameter $p$ in input $I$. Assume $I' \in \mathbb{I}$ such that there exists a non-empty set $Q \subseteq P_{disc}$ and for all $q \in Q$, $I_q \neq I'_q$ and for all $p \in P \backslash Q$, $I_p = I'_p$. If $|f(I) - f(I')| > \gamma$, then $I$ is called a discriminatory input of the classifier $f$ and is an instance that manifests the violation of (individual) fairness in $f$.

This provides Aequitas with a method to check whether a given model produces what it considers unfair classifications.

### B. Measuring Accuracy

There are multiple metrics that can be used to measure the performance of a machine learning fairness model. This includes fairness (exact definition subject to the model), but another way that machine learning fairness can be measured is by looking at the accuracy of the model based on classifier label assignments. The accuracy of a given model can be measured in terms of correct assignments, false positives, and / or false negatives relative to the total number of results outputted by the model. When working with models whose classifiers assign a binary decision $-1$ or $+1$ (for example, whether or not an individual is likely to succeed at Carleton College, in order to determine whether a student should be accepted), an incorrect assignment can either be a false positive or a false negative (in which an entry is given the binary outcome $+1$ when it should have returned $-1$, and vice versa). These rates can differ for different attributes.

The distributions of false positives and false negatives within a given dataset can sometimes provide more information about the fairness of a model than considering both as a single measurement. While the accuracy of COMPAS was approximately the same for different racial groups, the false positive and false negative rates differed widely between the racial groups.[8]

From a theory perspective, unless you have access to perfect information, you cannot make sure that all three forms of accuracy are equally treated. Because it is so challenging for all three factors to be treated equally, different groups and individuals have proposed different ideas over which elements of scoring the accuracy of fairness should be prioritized.

Aequitas may fail to locate all discriminatory inputs in the input space; however, it does guarantee that no false positives (non-discriminatory inputs that are incorrectly marked as being discriminatory) will be generated. This needs to be taken into account when considering the accuracy of the model.

### C. Fairness / Accuracy Trade-offs

One method to deal with the trade-off between fairness and accuracy is via the use of fairness constraints as well as via the use of accuracy constraints. These constraints allow a minimal degree of leeway in either accuracy or fairness, as long as the other remains at a predetermined level deemed acceptable. One important accuracy constraint is the business necessary clause, and one important fairness constraint is the p% rule.

The business necessity clause maximizes fairness under accuracy constraints. Because corporations, academic institutions, and other groups are often required to meet performance standards when it comes to the accuracy of their decision making processes, they are allowed to exhibit some degree of unfairness in order to meet these accuracy thresholds.[9]

While Aequitas aims to increase accuracy when possible, it instead works more to maximize accuracy under fairness constraints. It does this using a commonly-used fairness constraint present in multiple previous models: the p% rule.[6]

A given decision making process fails to satisfy the p% rule if the proportion of individuals with some attribute that receive some classification (positive or negative) is less than p% of the proportion of individuals without the attribute that receive the other classification. Decision making processes in violation of this p% rule are likely to be making biased decisions. However, if a committee decides to investigate a company in violation of this rule, said company only need to provide a reasonable explanation for this violation as a justification if it occurs. This rule has been occasionally altered to fit different kinds of models, such as working alongside a decision boundary.[9]

While Aequitas states that it can deal with discriminatory input using any p% rule, the authors of the Aequitas paper only provided the results when using a 100% rule, by setting their discriminatory threshold used in testing to 0.[6] While this greatly increases the fairness present in the retrained dataset, having such a high p% rule value could potentially lead to high levels of inaccuracy.[9]

### D. Machine Learning Fairness Strategies Overview

Different machine learning strategies use different methods to deal with issues of fairness. On a high level, these fall into the following categories: preprocessing, processing, and post-processing models.

Aequitas is a pre-processing model that discovers discriminatory inputs and can retrain a model based off of the discriminatory inputs discovered. Therefore, to better understand Aequitas, it is relevant to first dive deeper into the purposes and mechanics behind pre-processing, as well as how Aequitas retraining differs from previous approaches at machine learning fairness.

### E. Preprocessing

Preprocessing assumes that unfairness stems from biased data that fails to reflect reality. Some argue that preprocessing is correcting historical biases. In other words, preprocessing attempts to deal with the truth that not all data reflects the reality of the world, because assumptions have been made that led to an under-representative dataset. Preprocessing is a common method amongst machine learning fairness models, yet different models rely on different assumptions about how the data being provided reflects the real world.

One model around causality is counterfactual fairness. Counterfactual fairness assumes that if some discriminatory attribute should not be causally linked to a given outcome in reality, then changing said attribute will not change the outcome in a fair model. If the data demonstrates an effect between the input and output, then the model is deemed to be unfair. If the assumption being made reflects reality, then counterfactual fairness is likely to identify discriminatory inputs. At a high level, Aequitas runs on the assumption that the discriminatory input being discovered (in their case, gender) should not have a causal effect on the results if all other attributes are shared between two entries.[6]

### F. Retraining

One of the things that sets Aequitas apart from other machine learning fairness models is the fact that Aequitas is capable of using found discriminatory inputs in order to retrain a model, making it possible for the retrained model to exhibit a higher degree of fairness.[6] Aequitas uses a directed test generation strategy to assist in retraining classifiers. More information on Aequitas retraining can be found in the "Understanding Aequitas" section of this survey paper.

Many of the previous strategies in machine learning fairness initially find discriminatory input based on random points in the input space. Aequitas also finds discriminatory input based on random points in the input space. However, unlike previous methods, when Aequitas identifies said input, it attempts to find related points in the vicinity of the said inputs that might also be discriminatory. This deals with a common problem other programs face: random test generation risks not identifying discriminatory input when all the discriminatory input is clumped in a few regions.[6] Aequitas then uses the results of the directed test generation to retrain existing models with an increasing percentage of discriminatory input added to the training set, as long as the fairness of the model increases and the percentage does not exceed 100%. One thing to consider is that the research presented by the primary source assumes the robustness property of machine learning models is true. More information on robustness property can be found in Section VII of this survey paper.

### G. Limitations of Machine Learning Fairness Models

Previous machine learning fairness strategies were not perfect, and were fraught with limitations. Some of them also apply to Aequitas, while for others Aequitas managed to find new strategies to work around them. One limitation faced by a wide breadth of machine learning fairness models is that each only worked with a narrow range of classifiers, and not all previous models were able to eliminate both disparate impact and disparate treatment simultaneously. Another limitation of previous machine learning fairness strategies is that there are often limits in the types of sensitive attributes the methods can handle. These can include only working with a single attribute, or not being able to handle non-binary sensitive attributes.[9] The primary source believes that Aequitas could be expanded to work on different types of sensitive attributes. At the moment, Aequitas only works on one discriminatory input.

### H. Further Resources

This fairness in machine learning section of the survey paper has provided a brief overview of some of the topics of fairness preceding and relevant to Aequitas, but there is more information available for further learning that did not get covered in this limited amount of space. Here are a few:

- For more information on more types of fairness that have been proposed over the past several decades, as well as more information about fairness and unfairness both in and out of the field of machine learning, see B. Hutchinson and M. Mitchell (2019).
- More information on the strengths and limitations of previous models, as well as more information on fairness constraints, can be found from Zafar M, Valera I, Rodriguez M et al. (2015).

## VI. ROBUSTNESS IN MACHINE LEARNING

As we will see when we explain *Aequitas* in detail, the principal underlying assumption of the *Aequitas* algorithm is the robustness of machine learning models. Without this hypothesis, we cannot generate similar discriminatory inputs based on the initial set of discriminatory inputs found on the first part of the algorithm (see Section IX). Even though we will give the precise and technical definition of robustness, we can understand, for now, that robustness states that the output of a machine learning model is not dramatically changed by small changes to its input.

The robustness claim made by our primary source is based on the results of Fawzi et. al (2015)[11], which we describe in this section.

Adversarial perturbations are minimal perturbations made to the inputs of a classifier so that the classifier will switch the estimated label of that input. The opposite of adversarial perturbation is random uniform noise, which refers to randomly changing an input, in contrast to making changes to intentionally change the label of the input. The main result from this paper is that there is an upper bound to classifiers' robustness against adversarial perturbations and random uniform noise.

The paper also claims that the adversarial instability is due to the low flexibility of some classifiers. This refers to the capability of expression of some model classes, for instance, linear classifiers are less flexible than high-degree polynomial classifiers.

### A. Definitions

Now, we begin introducing the technical terms that will allow us to formally express the aforementioned upper bound.

Let $\mu$ be a probability measure defined on $\mathbb{R}^d$, which assigns probabilities on the subset of points that we wish to classify. For each point $x$ in this subset, let $y$ be the unknown function that we want to approximate and such that $y(x) \in \{-1, 1\}$. Then $\mu$ tells you the probability that $y(x) = 1$ or $y(x) = -1$ for any $x$ that we wish to classify. Furthermore we assume that these points $x$ are in a $M$-ball,

$$\mathbf{B} = \{x \in \mathbb{R}^d : \|x\|_2 < M\}.$$

Note that,

$$P_\mu(x \in \mathbf{B}) = 1.$$

Now let $\mu_1$ and $\mu_{-1}$ be the probability distribution of points $x \in \mathbf{B}$ such that $y(x) = 1$ and $y(x) = -1$, respectively. Furthermore, suppose that $f : \mathbb{R}^d \to \mathbb{R}$ is a classification function where the sign of $f(x)$ is the label that we will assign to $x \in \mathbb{R}^d$. Note that in the language that we had previously, $f$ is the hypothesis, while $y$ is the unknown "truth of the world."

One common measure of performance for $f$ is the risk of $f$,

$$\begin{aligned} R(f) &= P_\mu(\text{sign}(f(x)) \neq y(x)) \\ &= p_1 P_{\mu_1}(f(x) < 0) + p_{-1} P_{\mu_{-1}}(f(x) \geq 0), \end{aligned} \tag{27}$$

where $p_{\pm 1} = P_\mu(y(x) = \pm 1)$. The non-technical way to express the risk of $f$ is "what is the probability that we sample from $\mathbf{B}$ by $\mu$ and get a point $x$ that $f$ will missclassify."

Given $x \in \mathbb{R}^d$ sampled with probability as determined by the probability measure $\mu$, we define $\Delta_{\text{adv}}(f : x)$ as the smallest perturbation that changes the sign of $f(x)$. In other words,

$$\Delta_{\text{adv}}(f : x) = \min_{r \in \mathbb{R}^d} \|r\|_2 \text{ with } f(x)f(x+r) \leq 0. \tag{28}$$

The robustness to adversarial perturbations of $f$ is defined as the average $\Delta_{\text{adv}}(f : x)$ over all $x \in \mathbf{B}$,

$$\rho_{\text{adv}}(f) = E_\mu(\Delta_{\text{adv}}(f : x)). \tag{29}$$

Furthermore, note that this value is independent of the actual signs of $f(x)$ or $y(x)$, rather it depends solely on how much we need to perturb the inputs to change the estimated label.

We can similarly define robustness of $f$ to random uniform noise. Given $\varepsilon \in [0,1]$, let,

$$\Delta_{\text{unif},\varepsilon}(x:f) = \max_{\eta \geq 0} \eta \ \text{ with } \ P_{n\sim\eta\mathbb{S}}(f(x)f(x+r) \leq 0) \leq \varepsilon \tag{30}$$

where $\eta\mathbb{S}$ is the $\eta$-Ball around $x$. Intuitively, this value measures the maximum radius around $x$ for which the probability of missclassifying a point within the ball defined is less than $\varepsilon$. Similarly, the robustness is the average $\Delta_{\text{unif},\varepsilon}(x:f)$ over all $x \in \mathbf{B}$,

$$\rho_{\text{unif},\varepsilon}(f) = E_{\mu}(\Delta_{\text{unif},\varepsilon}(f:x)). \tag{31}$$

It is important to note that there are classifiers $f$ such that $R(f) = 0$ for which $\rho_{\text{adv}}(f)$ is small. In other words, the fact that a classifier has, in particular, 0 risk on its training set doesn't mean that it will be robust against adversarial perturbations. This is akin to the problem of over-fitting, where a classifier works very well for its training dataset, but it doesn't capture the concept of the problem correctly.

### B. Assumption A

We say that a classifier $f$ satisfies Assumption A if there are real numbers $\tau > 0$ and $0 < \gamma \leq 1$ such that for every $x \in \mathbf{B}$,

$$\operatorname{dist}(x, S_-) \leq \tau \max(0, f(x))^{\gamma} \tag{32}$$

$$\operatorname{dist}(x, S_+) \leq \tau \max(0, -f(x))^{\gamma} \tag{33}$$

where

$$\operatorname{dist}(x, S) = \min_{y}\{\|x-y\|_2 : y \in S\}$$

and $S_+ = \{x \in \mathbf{B} : f(x) \geq 0\}$ and $S_- = \{x \in \mathbf{B} : f(x) < 0\}$.

Intuitively, we say that $f$ satisfies Assumption A if for any $x \in \mathbf{B}$ the distance from $x$ to $S_-$ and $S_+$ is bounded.

### C. Upper Bound

If $f$ satisfies Assumption A then,

$$\begin{aligned}\rho_{\text{adv}}(f) \leq 4^{1-\gamma}\tau(&p_1 E_{\mu_1}(f(x)) \\ &- p_{-1}E_{\mu_{-1}}(f(x)) + 2\|f\|_{\infty}R(f))^{\gamma}.\end{aligned} \tag{34}$$

Therefore, we have found an upper bound of machine learning model's robustness against adversarial perturbations. This bound is done with respect to the risk as well as the difference between the expectations of the values of the classifiers computed on the distributions $\mu_1$ and $\mu_{-1}$. We will not worry about what the value of the right hand side of the inequality is, just that the value $\rho_{\text{adv}}(f)$ is bounded.

This ends our survey on this topic, however there is more about robustness of linear and quadratic classifiers in Fawzi et al (2015). There is also a proof of equation (34) and a similar upper bound for robustness of linear classifiers against random uniform noise. Furthermore the paper goes on to explain that, for the particular case of linear classifiers, the robustness against random uniform noise is greater than the robustness against adversarial perturbations by a factor of $\sqrt{d}$, where $d$ is the dimension of the inputs.

## VII. LAW OF LARGE NUMBERS

Aequitas also relies on the Law of Large Numbers, as the idea behind the Global search step is that repeated sampling of the input space guarantees that at least one discriminatory input will be found.

Thus, as a group, we decided to investigate more about this concept, and this is what we found[12].

### A. Weak Law of Large Numbers

Let $(X_1, X_2, \ldots)$ be an independent and identically distributed sequence of random variables with finite expectation $\nu$. That is $E(X_1) = E(X_2) = \cdots = \nu$. The sequence of sample averages $(\overline{X}_1, \overline{X}_2, \ldots)$, where,

$$\overline{X}_i = \frac{X_1 + \cdots + X_i}{i}$$

satisfies,

$$\lim_{n\to\infty} P(|\overline{X}_n - \nu| < \varepsilon) = 1 \ \ \forall \varepsilon > 0. \tag{35}$$

What this means is that there is a very high probability that the sample mean is within an arbitrary margin of the expected mean as the number of experiments increases.

### B. Strong Law of Large Numbers

With the same notation as the Weak Law of Large Numbers, we claim,

$$P\left(\lim_{n\to\infty} \overline{X}_n = \mu\right) = 1. \tag{36}$$

This result implies if we define the sample space of all infinite sequences of experiments $(X_1, X_2, \ldots)$, then the probability of sampling a sequence whose sample mean converges to $\mu$ is 1.

We omit the proof to these theorems as they are out of the scope of this paper. However, the importance of this result is as follows: say that we have a population of interest with unknown mean $\mu$. If we sample $n$ elements from this population at random and measure their values in a sequence $X_1, X_2, \ldots, X_n$ then the mean of this sample, $\overline{X}_n$, will almost certainly be arbitrarily close to $\mu$ as $n$ increases.

## VIII. INTERLUDE

This concludes the exploratory part of our Integrative Exercise. In the following section we study our primary resource, Udeshi et al., 2017[6]. Our goal with this exploration was to learn about the different, and for the most part unrelated, topics on which this paper relies. This is helpful because Udeshi et el. describes an algorithm called *Aequitas*, whose goal is to improve the fairness of machine learning models. Thus, the topics of machine learning and machine learning fairness are relevant in our discussion. As we will also see, machine learning robustness and the law of large numbers, as introduced in the sections above, serve as the mathematical background for *Aequitas*.

## IX. UNDERSTANDING *AEQUITAS*

In this section we provide a discussion of *Aequitas* as presented in our primary source, "Automated Directed Fairness Testing." (Udeshi et al., 2017)[6]

As mentioned previously, there is no consensus within the field of machine learning on how to solve the issue of bias in models, especially since there can be conflicting definitions and forces pulling potential solutions. Thus, we must state which kind of fairness *Aequitas* will work with. Firstly, *Aequitas* defines fairness as individual fairness, which we will explain in detail below. At a high level though, individual fairness is the notion that we should classify people with similar characteristics but different sensitive features (e.g. race, gender, etc.) equally. *Aequitas* also believes that model bias stems from the preprocessing part of a learning algorithm, In other words, in the data collection. *Aequitas* also believes in counterfactual fairness, which means that we would like models to treat people belonging to protected categories better than what may happen in reality. Lastly, *Aequitas* works with a fairness constraint, it tries to maximize accuracy within a predetermined numerical fairness threshold.

Thus, the goals of *Aequitas*, given a potentially biased machine learning model, is to increase it individual fairness within the specified numerical threshold. The specific inputs of *Aequitas* are the model itself, its input features, and the dataset that was used to train it. The preprocessing assumption tells us that bias may underlie the training dataset as a reflection of the bias of the real world and therefore the model that was trained based on it may have learned those biases.

In particular, *Aequitas* will systematically explore the input space and find inputs that induce discrimination. *Aequitas* combines the found inputs and the original training dataset to automatically retrain the model, in the hopes of improving its fairness.

The novelty of this approach is that the retraining dataset is generated by a directed approach, in contrast to randomly selecting inputs as done by other state-of-the-art fairness algorithms. Furthermore, the directed algorithm relies on the concept of robustness of machine learning models to find discriminatory inputs for the retraining dataset. Robustness states that small changes to the input of a classifier will not change its estimated label. Even though Fawzi et al. (2015) found that there is an upper bound to robustness against adversarial inputs, these inputs are found only in small regions of the input space. *Aequitas*' directed approach will eventually avoid these regions if it fails to find discriminatory inputs.

### A. Outline of the Algorithm

First, we start with a motivating example. Suppose that we count with a machine learning model that takes in the characteristics of a person, such as their education and general qualifications as well as their gender and race, and outputs a hiring recommendation. If we have two inputs to this model which have the same qualifications and requirements relevant for a job, we would like the model to recommend hiring both. However, if these inputs differ by a sensitive feature, let's say gender, and are classified differently, then the model is biased against gender. Then, we say the the inputs are discriminatory and the model violates individual fairness.

Note that in the above scenario, any difference of model classification is considered discriminatory since we only consider two options: hiring or not hiring. Sometimes we will consider other kinds of numerical outputs and in those cases we use a fairness threshold: if the difference between the outputs is within the threshold, then we do not consider the inputs discriminatory.

Henceforth, we describe an algorithm, *Aequitas*, to systematically find discriminatory inputs to retrain our machine learning model and improve individual fairness. Overall, *Aequitas* has three big steps:

*a. Global search.* In this step, Aequitas uniformly samples the input space and saves any discriminatory inputs that it finds. The authors of the paper claim that if there are any discriminatory inputs, then the algorithm is almost guaranteed to find at least one. Note that this step is repeated several times. Moreover, an input $I$ is considered discriminated against if there is an input $I'$ that is the same as $I$ except for the value of a protected category such as gender or race. If $|f(I) - f(I')|$ is larger than a predefined fairness threshold, then $I$ is considered a discriminatory input.

*b. Local search.* Given the set of discriminatory inputs from the previous step, we claim, by robustness, that in the neighborhood of each individual discriminatory input there will be more discriminatory inputs.

Below is an overview of the local search step in limited detail. The notation below will be used throughout this section.

Let $\mathbb{I}$ represent the input domain. That is, $\mathbb{I}$ is the set of all possible inputs to the model. Let $P_1, \dots, P_n$ denote each the value of each parameter of a particular input $I \in \mathbb{I}$. Let $\mathbb{I}_k$ be the set of all values that the parameter $P_k$ can take. Then note that $\mathbb{I} = \mathbb{I}_1 \times \cdots \times \mathbb{I}_n$ as $\mathbb{I}$ is the set of all possible inputs, and thus it is also the Cartesian product of all of the possible values of each individual parameter. An input parameter $p$ from $P_1, \dots, P_n$ can potentially be discriminatory if the classifier should not be biased against specific values in $I_p$. For example $p$ could be gender in a creditworthiness classifier—we do not want classifiers predicting the income of a person based on gender.

Let $I \in \mathbb{I}$, let $I_k$ be the value of the parameter $P_k$ in $I$, let $I^{(d)}$ be a discriminatory input that we found on the previous part of the algorithm, and let $P_{\text{disc}} \subseteq \bigcup_{i=1}^{n} P_i$ be the set of parameters that we hypothesize might induce discrimination.

There are three flavors of Aequitas' local search:

1. Aequitas randomly chooses a parameter
$$p \in \bigcup_{i=1}^{n} P_i \backslash P_{\text{disc}}.$$
Then a small perturbation $\delta$ usually chosen at random from $\{1, -1\}$ is added to $I_p^{(d)}$.
2. The parameter $p$ is still chosen uniformly at random, but given a perturbation $\delta$ of $I_p^{(d)}$ that consistently yields discriminatory inputs, then $\delta$ is chosen with higher probability.
3. We augment the previous approach by also picking parameters $p$ that consistently yield discriminatory inputs with higher probability.

After we find the perturbed input, we add it to the retraining dataset if it is a discriminatory input.

The approach described works because of robustness (see Section VI). That is, we expect that slightly perturbed inputs in the neighborhood of initially found discriminatory inputs will also induce discrimination.

Moreover, by the law of large numbers, the average amount discriminatory inputs found after a large number of iterations will approximate the true ratio of discriminatory inputs in $\mathbb{I}$.

*c. Retraining.* We add a portion of the discriminatory inputs found to the original dataset and train a new model with it. We only add a portion of the inputs because these inputs are artificially generated so they do not "represent" reality. Since this new dataset could skew the behavior of the model, we try to add different amounts of discriminatory inputs and gauge the sensitivity of the model. Note that this reflects Counterfactual fairness: we add these discriminatory inputs, which are generated and not "factual," to the original dataset to improve the fairness of a model.

In summary, *Aequitas* begin by sampling uniformly at random from the input space of the model in question. Whenever *Aequitas* finds a discriminatory input, it adds it to the retraining set. By the Law of Large Numbers (see Section VII) there is a high probability that we will find discriminatory inputs if we sample enough times. We look at the neighborhood of each input in previous step to find more discriminatory inputs. We do this by perturbing the inputs, where the perturbations are probabilistic and learn from previous iterations. After this is done, we use the discriminatory inputs found to retrain the model.

### B. Aequitas, formal declaration

Now we proceed with the formal declaration of the algorithm. The discussion in the previous discussion is detailed enough for a superficial understanding of the algorithm. Here we concern ourselves with the mathematical definitions and justifications of *Aequitas*. Note that we use the notation that we used in the previous section.

We start by giving more notation and definitions that we will need.

Let $P = \bigcup_{i=1}^{n} P_i$ be the set of parameters of inputs in $\mathbb{I}$.

**Definition IX.1** (Individual Fairness, Discriminatory Input)**.** Let $f$ be a classifier under test, and let $\gamma$ be a user defined discrimination threshold. Let $I \in \mathbb{I}$. Suppose there is $I' \in \mathbb{I}$ such that there is a non-empty subset $Q \subseteq P_{\text{disc}}$ and such that for every $q \in Q$, $I_q \neq I'_q$ and for all $p \in P \backslash Q$, $I_q = I'_q$. If $|f(I) - f(I')| > \gamma$ then $I$ is a discriminatory input of the classifier $f$ and reflects the violation of Individual Fairness in the classifier $f$.

A perturbation $g$ is a function $g : \mathbb{I} \times (P \backslash P_{\text{disc}}) \times \Gamma \to \mathbb{I}$ where $\Gamma = \{+1, -1\}$ defines the directions in which we can perturb an $I \in \mathbb{I}$. Then $g(I, p, \delta)$ is the input $I' \in \mathbb{I}$ such that $I'_p = I_p + \delta$ and such that for all $q \in P \backslash \{p\}$ we have that $I'_q = I_q$.

Now we formally define *Aequitas*.

*a. Global search.* (See Algorithm 1) As mentioned before, we want to take a subset of discriminatory inputs from $\mathbb{I}$ to drive our local search algorithm. To this end, we select an input $I \in \mathbb{I}$ at random, and generate a set of inputs $\mathbb{I}^{(d)}$ from $I$ that cover all possible values of $P_{\text{disc}}$. Finally, from $\mathbb{I}^{(d)}$ we find discriminatory inputs as defined previously.

**Algorithm 1:** Global Search

**procedure** `GLOBAL_EXP`($P$, $P_{disc}$)
    `disc_inputs` $\leftarrow \emptyset$
    Let $P' = P \backslash P_{\text{disc}}$
    `//` $N$ `is the number of trials`
    **foreach** $i$ *in* $(0, N)$ **do**
        Select $I \in \mathbb{I}$ at random
        `//` $\mathbb{I}^{(d)}$ `extends` $I$ `with all possible values of` $P_{\text{disc}}$
        $\mathbb{I}^{(d)} \leftarrow \{I' | \forall p \in P', I_p = I'_p\}$
        **if** $\exists I', I'' \in \mathbb{I}^{(d)}, |f(I') - f(I'')| > \gamma$ **then**
            `disc_inputs` $\leftarrow$ `disc_inputs` $\cup \{I\}$
        **end**
    **end**
    **return** `disc_inputs`
**end**

*b. Local search.* (See Algorithm 2) Let `disc_inputs` be the set of discriminatory inputs found on the previous step. Now we want to find discriminatory inputs in the neighborhood of inputs in `disc_inputs`. Remember that we can do this because of the robustness of machine learning models.

Thus, Aequitas perturbs an input $I \in$ `disc_inputs` by changing the value of $I_p$ by $\delta \in \{+1, -1\}$ where $p \in P \backslash P_{\text{disc}}$. Note that by changing $I_p$, $I$ is now a different input, and it will be further perturbed in the inner loop of Algorithm 2.

Now, the perturbations on the inputs are chosen at random. Perturbations are defined by picking a feature to perturb and a direction or value by which to perturb so we randomly pick both values separately. In this regard, let $\sigma_{\text{pr}}$ be the array such that $\sigma_{\text{pr}}[p]$ is the probability of picking $p$ for any $p \in P \backslash P_{\text{disc}}$. The probability of picking a value of perturbation will depend on which feature we choose to perturb, so let $\sigma_v$ be the array such that $\sigma_v[p]$ is the chance of picking a perturbing the feature denoted by $p$ by $\delta = -1$ for any $p \in P \backslash P_{\text{disc}}$.

**Algorithm 2:** Local Search

**procedure** LOCAL_EXP(*disc_inputs*, $P$, $P_{disc}$, $\Delta_{\upsilon}$, $\Delta_{pr}$)
  Test $\leftarrow \emptyset$
  Let $P' = P \backslash P_{\text{disc}}$
  Let $\sigma_{\text{pr}}[p] = \frac{1}{|P'|}$ for all $p \in P'$
  Let $\sigma_{\upsilon}[p] = 0.5$ for all $p \in P'$
  **foreach** $I \in$ *disc_inputs* **do**
    // $N$ is the number of trials
    **foreach** $i$ *in* $(0, N)$ **do**
      Select $p \in P'$ with probability $\sigma_{\text{pr}}[p]$
      Select $\delta = -1$ with probability $\sigma_{\upsilon}[p]$
      // $I$ is modified as a side effect of modifying $I_p$
      $I_p \leftarrow I_p + \delta$
      // $\mathbb{I}^{(d)}$ extends $I$ with all possible values of $P_{\text{disc}}$
      $\mathbb{I}^{(d)} \leftarrow \{I' | \forall p \in P', I_p = I'_p\}$
      **if** $\exists I', I'' \in \mathbb{I}^{(d)}, |f(I') - f(I'')| > \gamma$ **then**
        // Add perturbed input $I$
        Test $\leftarrow$ Test $\cup \{I\}$
      **end**
    **end**
    update_prob($I, p,$ Test$, \delta, \Delta_{\upsilon}, \Delta_{\text{pr}}$)
  **end**
  **return** Test
**end**

As noted before, the parameter $p$ is chosen with probability $\sigma_{\text{pr}}[p]$ where initially $\sigma_{\text{pr}}[p] = \dfrac{1}{|P \backslash P_{\text{disc}}|}$. Once $p$ is chosen, we need to choose $\delta \in \{+1, -1\}$ where the probability of $\delta = +1$ is $\sigma_{\upsilon}[p]$ and $\delta = -1$ is $1 - \sigma_{\upsilon}[p]$. Initially, $\sigma_{\upsilon}[p] = 0.5$ for all $p \in P \backslash P_{\text{disc}}$.

There are three kinds of Aequitas based on how we update $\sigma_{\text{pr}}[p]$ and $\sigma_{\upsilon}[p]$.

**Aequitas Random**. This is when $\sigma_{\text{pr}}[p]$ and $\sigma_{\upsilon}[p]$ are not updated. This is equivalent to sampling inputs uniformly at random from the neighborhood of an input $I \in$ `disc_inputs`.

**Algorithm 3:** Aequitas semi-directed update probability

**procedure** UPDATE_PROB($I$, $p$, *Test*, $\delta$, $\Delta_{\upsilon}$, $\Delta_{pr}$)
  **if** $(I \in Test \wedge \delta = -1) \vee (I \notin Test \wedge \delta = +1)$ **then**
    $\sigma_{\upsilon}[p] \leftarrow \min(\sigma_{\upsilon}[p] + \Delta_{\upsilon}, 1)$
  **end**
  **if** $(I \notin Test \wedge \delta = -1) \vee (I \in Test \wedge \delta = +1)$ **then**
    $\sigma_{\upsilon}[p] \leftarrow \max(\sigma_{\upsilon}[p] - \Delta_{\upsilon}, 0)$
  **end**
**end**

**Aequitas semi-directed**. (See Algorithm 3) In this case, we drive the test generated by updating $\sigma_{\upsilon}$. Note that $p$ is still chosen uniformly at random. For each $p \in P \backslash P_{\text{disc}}$ we initialize the probability of the perturbation value $\delta = -1$ as $\sigma_{\upsilon}[p]$ and $\delta = +1$ as $1 - \sigma_{\upsilon}[p]$. Based on whether a particular $\delta$ led to a discriminatory input, $\sigma_{\upsilon}[p]$ is increased or decreased by a user determined value $\Delta_{\upsilon}$.

**Aequitas fully-directed**. (See Algorithm 4) This approach extends Aequitas semi-directed by systematically updating $\sigma_{\text{pr}}[p]$. For any $p \in P \backslash P_{\text{disc}}$ we initialize $\sigma_{\text{pr}}[p] = \dfrac{1}{|P \backslash P_{\text{disc}}|}$. If the perturbation of $p$ by $\delta$ (which is chosen as noted in Aequitas semi-directed) leads to a discriminatory input then we add a user determined value $\Delta_{\text{pr}}$ to $\sigma_{\text{pr}}[p]$. To reflect this change in probability we normalize $\sigma_{\text{pr}}[p'] = \dfrac{\sigma_{\text{pr}}[p']}{\sum_{x \in P \backslash P_{\text{disc}}} \sigma_{\text{pr}}[x]}$ for every $p' \in P \backslash P_{\text{disc}}$. We do this because we still need the probabilities to add up to 1 even after updating $\sigma_{\text{pr}}[p]$.

**Algorithm 4:** Aequitas fully-directed update probability

**procedure** UPDATE_PROB($I$, $p$, *Test*, $\delta$, $\Delta_{\upsilon}$, $\Delta_{pr}$)
  **if** $(I \in Test \wedge \delta = -1) \vee (I \notin Test \wedge \delta = +1)$ **then**
    $\sigma_{\upsilon}[p] \leftarrow \min(\sigma_{\upsilon}[p] + \Delta_{\upsilon}, 1)$
  **end**
  **if** $(I \notin Test \wedge \delta = -1) \vee (I \in Test \wedge \delta = +1)$ **then**
    $\sigma_{\upsilon}[p] \leftarrow \max(\sigma_{\upsilon}[p] - \Delta_{\upsilon}, 0)$
  **end**
  **if** $I \in Test$ **then**
    $\sigma_{\text{pr}}[p] \leftarrow \sigma_{\text{pr}}[p] + \Delta_{\text{pr}}$
    $\sigma_{\text{pr}}[q] \leftarrow \frac{\sigma_{\text{pr}}[q]}{\sum_{x \in P \backslash P_{\text{disc}}} \sigma_{\text{pr}}[x]}$ for all $q \in P \backslash P_{\text{disc}}$
  **end**
**end**

*c. Estimation using the Law of Large Numbers.* Using Aequitas we can estimate the percentage of discriminatory inputs in $\mathbb{I}$. Let $X_1, X_2, \ldots X_K$ be a sequence of random variables, each counting the proportion of discriminatory inputs in a random sample of $m$ inputs from the input space. Note that $E[X_i] = m^*$ for all $1 \leq i \leq K$, where $m^*$ is the true proportion of discriminatory inputs in the input space. Then by the law of large numbers, the average of the samples, $\overline{X} = K^{-1} \sum_{i=1}^{K} X_i$, will tend to be close to the expected value as $K$ increases. Hence, $\overline{X} \to m^*$ as $K \to \infty$. Whence, we call `LLN_Fairness_Estimation` the function that takes in a model and samples its input space uniformly at random multiple times to find discriminatory inputs. If we do this enough times, we will get a "good" estimation of the true ratio of discriminatory inputs in the input space. For a usage of this function, see Algorithm 5, line 14.

*d. Automatic retraining of the model.* (See Algorithm 5) Let `Test` be the set of generated test inputs from the local search step. We use `Test`—whose elements show the violation of the desired properties that we want our model to have—to retrain our machine learning model. The strategy we use is to add portions of `Test` to the original training dataset. This is because adding all of the elements in `Test` to the training dataset may skew the model, as the elements in `Test` may be an over represented region in the input space.

Now we describe how we select portions of `Test` to add to the training dataset. Suppose that $M = |\texttt{Test}|$. Then for $i \in [2, 7]$, we choose $p_i$ randomly in a range between $[2^{i-2}, 2^{i-1}]$ and select $\dfrac{M * p_i}{100}$ elements from `Test` at random. Intuitively, $p_i$ represents the percentage of elements from `Test` that we will select. After all of the iterations, we keep the retrained model that had the least percentage of discriminatory inputs in its input space, as computed by `LLN_Fairness_Estimation`.

The intuition behind picking $p_i$ from an exponentially increasing range $[2^{i-2}, 2^{i-1}]$ is to maintain our level of fairness relatively high while also making the execution of the program fast.

**Algorithm 5:** Retraining

**procedure** `Retraining`($f$, $Test$, $training\_data$)
  $N \leftarrow \infty$
  $f_{\text{cur}} \leftarrow f$
  **foreach** $i$ *in* $[2,7]$ **do**
    $p_i \leftarrow$ a real number between $[2^{i-2}, 2^{i-1}]$
    **if** $p_i > 100$ **then**
      Exit the loop
    **end**
    $k \leftarrow \text{len}(\texttt{training\_data})$
    $n_{\text{addn}} \leftarrow \frac{p_i \cdot k}{100}$
    $TD_{\text{addn}} \leftarrow n_{\text{addn}}$ randomly selected inputs from `training_data`
    $TD_{\text{new}} \leftarrow TD_{\text{addn}} \cup \texttt{training\_data}$
    $f_{\text{new}} \leftarrow$ model trained using $TD_{\text{new}}$
    `// LLN_Fairness_Estimation estimates the proportion of discriminatory inputs using the law of large numbers`
    $fair_{\text{cur}} \leftarrow \text{LLN\_Fairness\_Estimation}(f_{\text{cur}})$
    $fair_{\text{new}} \leftarrow \text{LLN\_Fairness\_Estimation}(f_{\text{new}})$
    **if** $fair_{cur} > fair_{new}$ **then**
      $f_{\text{cur}} \leftarrow f_{\text{new}}$
    **else**
      Exit the loop
    **end**
  **end**
  **return** $f_{\text{cur}}$
**end**

## C. Conclusion

We have described *Aequitas* in its entirety and given a brief survey on the theoretical topics on which it relies. In particular, we discussed the issue that arises when one wants to define fairness and how to solve that specific kind of fairness. Each one of these solutions is different and at times even contradictory. In this way, *Aequitas* is opinionated: it attempts to improve a model's Individual Fairness and it assumes that the bias that happens comes from the training dataset itself.

We can see how these two forces, Preprocessing and Individual Fairness, play a role in the statement of *Aequitas*: we define discriminatory inputs in terms of Individual Fairness and retrain the model by adding a set of them to the training dataset.

Within this realm of machine learning fairness algorithms, *Aequitas* is special because the retraining dataset, the one containing all of the discriminatory inputs, is not generated completely at random. A portion of the dataset, specifically the one that is found in the Local Search step, is found in a directed way: *Aequitas* looks for discriminatory inputs in the neighborhood of other discriminatory inputs.

Of course, there are limitations to *Aequitas*. In Udeshi et. al. there is no elaboration in the retraining techniques that *Aequitas* uses and they justify this by claiming that the essence of *Aequitas* is automated training dataset generation. On the other hand, the formal statement of *Aequitas* in the primary source assumes integer-valued features in the model's inputs, whereas the discussion on this article assumed that the inputs could be continuous.

Another shortcoming to *Aequitas* is its dependence on robustness. Recall that the Local Search step of the algorithm assumes that the neighborhood of discriminatory inputs will behave similarly, i.e. we will be able to find more discriminatory inputs. We found the citations from Udeshi et. al. on this topic not satisfactory and we question whether the notion of robustness is justified correctly.

All in all, fairness remains an important topic in machine learning as this technologies become more and more ubiquitous in our daily lives. We hope that *Aequitas* can be applied in real world scenarios and moreover that these conversations around fairness stay relevant.

Lastly, Udeshi et. al. provided a Python proof-of-concept implementation of their algorithm. The next section, which we have decided to include sort of as an appendix, explores the code in some detail.

### Appendix A: Understanding *Aequitas* Code

This section will highlight some important details of the Aequitas implementation. We begin by introducing the modules that were used to run Aequitas, after which we look at the core snippets of the code, comparing it to the intentions of the original algorithm.

#### 1. Modules Used

```
sklearn
```

Sklearn (Scikit Learn) module contains several machine learning algorithms that can be used to train models, including SVM, Ensemble, Decision Tree, and Neural Networks.

```
numpy
```

Numpy (Numerical Python) is a module that is frequently used in machine learning to manipulate the input and output and perform matrix operations during training. In Aequitas, Numpy plays an integral part in evaluating whether an arbitrary input is discriminatory.

```
scipy
```

Scipy (Scientific Python) is also used frequently during machine learning training. In the Aequitas code, scipy's "basin hopping" algorithm is used during local perturbation to discover more discriminatory inputs.

#### 2. Structure of the Code

Aequitas parameters are set up using a configuration file. Parameters defined in the configuration file persist throughout the Aequitas code.

```
params = 13

sensitive_param = 9 # Starts at 1.

input_bounds = []
input_bounds.append([1, 9])
input_bounds.append([0, 7])
...
input_bounds.append([0, 39])

classifier_name =
    'Decision_tree_standard_unfair.pkl'

threshold = 0

perturbation_unit = 1

retraining_inputs = 'Retrain_Example_File.txt'
```

Listing 1. config.py

`params` refers to the number of parameters in the dataset. `sensitive_param` is the index (based on 1) of the sensitive text (i.e. gender). `input_bounds` are the domain, or the value range of each of the parameters. `classifier_name` is the name of the file containing the sklearn-trained classifier to test fairness of. `threshold` is the discrimination threshold. `perturbation_unit` is the unit by which Aequitas perturbs the input in the local search. `retraining_inputs` is the name of the dataset used for the retraining. This dataset is the result of Aequitas algorithm execution.

#### 3. Code Snippets

Now we look at the code snippets that are integral to each part of the Aequitas algorithm.

##### *a. Training a Model*

In the initial training of a model, $X$ and $Y$, which are input features and prediction, respectively, are extracted from the input dataset, and are used to train a model such as a Decision Tree Classifier.

```
X = np.array(X)
Y = np.array(Y)
model = DecisionTreeClassifier()
model.fit(X, Y)
```

Listing 2. Generate_Sklearn_Classifier.py

##### *b. Evaluating Fairness*

Throughout Aequitas, evaluating whether a given input is 'biased' plays an integral role in determining 1) how fair the model is, and 2) what input gets added to the new retraining dataset. The series of code below illustrates this procedure.

```
def evaluate_input(inp):
    inp0 = [int(i) for i in inp]
    inp1 = [int(i) for i in inp]
```

Listing 3. Sklearn_Estimation.py

In the original code, it is hard coded that the sensitive feature (i.e. gender) must be binary, meaning only two values are possible. This is why the original input *inp* is cloned twice. If the sensitive feature is non-binary, the number of clones will match the number of possible values in the sensitive feature's domain.

Once there are two clones, only the value of the sensitive feature is changed so that each one has one of the two possible values, in this case 0 and 1.

```
    inp0[sensitive_param - 1] = 0
    inp1[sensitive_param - 1] = 1
```

Then, the model predicts the outcomes ($y$) of the two clones, and if the outcomes differ more than the threshold, we deem *inp* discriminatory.

```
    out0 = model.predict(inp0)
    out1 = model.predict(inp1)
    return abs(out0 - out1) > threshold
```

### c. Discriminatory Input Search

The aforementioned algorithm is used both in the *global* search and *local* search to detect discriminatory inputs. Aequitas first performs global search, where discriminatory inputs are found by sampling uniformly at random from the input space. Scipy's *basinhopping*, an algorithm similar to simulated annealing, is the algorithm that is used for this purpose. Simulated Annealing is an algorithm that takes into account both optimized direction of search and randomness to reach a goal (in this case, finding more discriminatory inputs). When global search finishes, local search uses *basinhopping* again, to find more discriminatory inputs in the neighborhood of these aforementioned inputs.

```
basinhopping(evaluate_global, initial_input,
    stepsize=1.0, take_step=global_discovery,
    minimizer_kwargs=minimizer,
           niter=global_iteration_limit)
```

Listing 4. Global Discovery

The parameter `take_step` takes in a function that performs the random displacement of features within the input space. In our case, that function is `Global_Discovery`, which given an input, randomly modifies the values of each of the parameters within the respective ranges, returning a new input sourced from the input space. This random sampling characterizes global search. When this new input is created, whether or not it is a discriminatory put is determined using the `evaluate_global` function, which employs a similar mechanism of evaluating an input as one that was introduced earlier. The only difference is that this time the discriminatory inputs are saved in an array.

For local search, there are three approaches. The difference between these approaches is whether the probability of choosing the perturbation direction (+1, -1) and/or choosing a feature to perturb are modified throughout the algorithm. In Aequitas Random, the feature to perturb and perturbation direction are chosen uniformly at random and not in a directed way. It represents a random exploration of the neighborhood of discriminatory inputs discovered through global search.

```
feature = random.randint(0, 12)
direction = [-1, +1]
# perturbation
x[feature] = x[feature] + random.choice(direction)
```

Listing 5. Local Discovery-Aequitas Random

In Aequitas Semi-Directed, the feature to perturb is still chosen randomly, but the direction of perturbation is now determined by the *direction_probability* for that specific feature.

```
init_prob = 0.5
direction_probability = [init_prob] * params
```

Direction of -1 is chosen with the probability of `direction_probability[feature]` and direction of +1 is chosen with the probability of `1 - direction_probability[feature]`. At first, the *direction_probability* of each input feature is the same, and choice of direction is random, as it was in Aequitas Random.

```
direction_choice = np.random.choice(direction,
    p=[direction_probability[feature], (1 -
    direction_probability[feature])])

x[feature] = x[feature] + (direction_choice *
    perturbation_unit)
```

However, in Aequitas Semi-Directed, the perturbed input is evaluated, and based on its "biasedness", *direction_probability* is adjusted.

```
ei = evaluate_input(x)

if (ei and direction_choice == -1) or (not ei and
    direction_choice == 1):
   direction_probability[feature] = min(
      direction_probability[feature] +
          (direction_probability_change_size *
          perturbation_unit), 1)

elif (not ei and direction_choice == -1) or (ei and
    direction_choice == 1):
   direction_probability[feature] = max(
      direction_probability[feature] -
          (direction_probability_change_size *
          perturbation_unit), 0)
```

Put simply, if the perturbed input is discriminatory, the *direction_probability* of the `direction_choice` will be rewarded because by robustness, this must mean that neighboring inputs in this direction will also likely be discriminatory. In other words, the algorithm increases `direction_probability[direction_choice]` to incentivize perturbing in this direction. Conversely, if the perturbed input is not discriminatory, `direction_probability[direction_choice]` will be decreased to disincentivize perturbation in this direction.

In Aequitas Fully-Directed, both the direction of perturbation and the feature to perturb are chosen with differing probabilities. The algorithm learns which features are more associated (correlated) with the sensitive feature, (i.e. frequency of parental leave and gender) and attempts to find discriminatory inputs more efficiently by adjusting those parameters with greater probability.

First, *direction_probability* and *param_probability* are initialized to be equal across all features.

```
init_prob = 0.5
direction_probability = [init_prob] * params
direction_probability_change_size = 0.001
param_probability = [1.0/params] * params
param_probability_change_size = 0.001
```

The very first input selection is random, as the probabilities are the same. However, after the initial input is evaluated, both *direction_probability* and *param_probability* are adjusted accordingly, in a manner similar to Aequitas Semi-Directed, in that the algorithm incentivizes the selection of feature and direction that yields a discriminatory input. For brevity I only include how *param_probability* is adjusted, since the adjustment to *direction_probability* have been discussed previously.

```
x[param_choice] = x[param_choice] +
    (direction_choice * perturbation_unit)

ei = evaluate_input(x) #True if biased, False
    otherwise

if ei:
   param_probability[param_choice] =
       param_probability[param_choice] +
       param_probability_change_size
   normalise_probability()
else:
   param_probability[param_choice] =
       max(param_probability[param_choice] -
       param_probability_change_size, 0)
   normalise_probability()
```

#### *d. Retraining*

Instead of naiively adding all the discriminatory inputs into the training set which might result in a skewed representation of the overall data, Aequitas chooses only a subset of the generated inputs to add at each iteration, checking if training with the additional inputs made fairness better or worse. If the newly trained model has better fairness, keep adding more inputs and see if fairness can be further improved. If the newly trained model yields worse fairness, return the current model without updating.

The additive percentage of the generated inputs (how much of it we add) is chosen randomly between

$$[2^i, 2^{i+1}]$$

where $i$ is the 0-based index of iteration.

```
additive_percentage = random.uniform(pow(2, i),
    pow(2, i + 1))
```

```
num_inputs_for_retrain = int((additive_percentage *
    len(X))/100)
```

This means, that the more iteration you go through of this process, more of the remaining generated inputs we add to the training set. Of course, if the percentage becomes greater than a 100, we exit. The code below illustrates the retraining procedure.

After the additive percentage has been decided, Aequitas sources randomly from the retraining dataset and retrains.

```
retrained_model = retrain(X_original, Y_original,
    np.array(X_additional), np.array(Y_additional)
```

It is important to learn about the code implementations of Aequitas not only to appreciate the intricacies of realizing the algorithm but also to detect where it might deviate from the original algorithm. For example, the original algorithm enables non-binary sensitive features to be processed, but the code implementation was limited to binary sensitive features. From this understanding, we were able to figure out how we might improve the code to stay truer to the original intentions of the algorithm.

## ACKNOWLEDGMENTS

We would like to thank Professor Anna Rafferty for the conversation on machine learning fairness, and Professor Dave Musicant for the discussion about support vector machines.

## REFERENCES

[1] P. Olson, “The Algorithm That Beats Your Bank Manager,” (2011).
[2] E. Reicin, “AI Can Be A Force For Good In Recruiting And Hiring New Employees,” (2021).
[3] K. Ferral, “Wisconsin Supreme Court allows state to continue using computer program to assist in sentencing,” (2016).
[4] J. Angwin, J. Larson, S. Mattu, and L. Kirchner, “Machine Bias: There’s software used across the country to predict future criminals. And it’s biased against blacks.” (2016).
[5] J. Dastin, “Amazon scraps secret AI recruiting tool that showed bias against women,” (2018).
[6] S. Udeshi, P. Arora, and S. Chattopadhyay, “Automated directed fairness testing,” ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering , 98–108 (2018).
[7] S. Russell and P. Norvig, *Artificial Intelligence: a Modern Approach*, 4th ed. (Pearson, 2020).
[8] S. Galhotra, Y. Brun, and A. Meliou, “Fairness testing: testing software for discrimination,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (ACM, New York, NY, USA, 2017).
[9] M. B. Zafar, I. Valera, M. G. Rodriguez, and K. P. Gummadi, “Fairness Constraints: Mechanisms for Fair Classification,” (2015).
[10] B. Hutchinson and M. Mitchell, “50 Years of Test (Un)fairness: Lessons for machine learning,” in *FAT* 2019 - Proceedings of the 2019 Conference on Fairness, Accountability, and Transparency* (Association for Computing Machinery, Inc, 2019) pp. 49–58.
[11] A. Fawzi, O. Fawzi, and P. Frossard, “Analysis of classifiers’ robustness to adversarial perturbations,” (2015).
[12] R. P. Dobrow, *Probability with applications in R* (2013).