Algorithmic Bias, Statistical Notions of Fairness, and the Seldonian Framework

Dasha Asienga
April 17, 2024

Submitted to the Department of
Mathematics and Statistics
of Amherst College in partial fulfillment
of the requirements for the degree of
Bachelor of Arts with honors.

Advisor:
Professor Katharine Correia

Abstract

The abstract should be a short summary of your thesis work. A paragraph is usually sufficient here.

Acknowledgments

Use this space to thank those who have helped you in the thesis process (professors, staff, friends, family, etc.). If you had special funding to conduct your thesis work, that should be acknowledged here as well.

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

The public and private sector are increasingly turning to data-driven methods to automate and to guide simple and complex decision-making. However, this trend raises an important question of bias. There is a lot of misinterpretation when it comes to the collection of data in many application areas, and there is a major concern for data-driven methods to further introduce and perpetuate discriminatory practices, or to otherwise be unfair because of the social and historical processes that operate to the disadvantage of certain groups.

For example, within healthcare, using mortality or readmission rates to measure hospital performance penalizes hospitals serving poor or non-White populations as those inherently have higher mortality and readmission rates due to confounding societal factors. Outside healthcare, credit-scoring algorithms predict outcomes based on income, which disadvantages low-income groups further perpetuating economic immobility. Policing algorithms result in increased scrutiny of Black neighborhoods because of the bias against Black people that is already present in the U.S. policing system, and hiring algorithms, which predict employment decisions, are affected by historical race and gender biases.

Yet, these algorithms are often regarded as ground truth and free of human limitations because they are based on mathematics, statistics, and computer science – otherwise regarded as objective disciplines. In theory, this should lead to greater fairness. However, left unregulated, these mathematical models privilege majority

groups and discriminate against minority groups because they often learn from inherently biased data. If the data used to train models contains bias, then the resulting algorithms will learn the bias and reflect it into their predictions. In many cases, this can be detrimental.

While there are widely-accepted, though sometimes disputed, societal notions of fairness, one key question emerges: are there any established statistical notions of fairness and bias? Is it possible to mathematically and statistically define algorithmic bias and unfairness, thereby paving a way for addressing the challenges they pose? And if so, are there ways to leverage statistical tools to resolve such bias and unfairness? This thesis paper aims to explore and answer precisely these questions.

1.1 Algorithmic Bias

There are multiple different types and sources of bias in the realm of statistics. In particular, algorithmic bias arises when an algorithm's decisions are skewed towards a particular group of people, either positively or negatively (Mehrabi, Morstatter, Saxena, Lerman, & Galstyan, 2021). The danger with biased algorithmic outcomes is that they generate a feedback loop. Take, for example, a hiring algorithm that discriminates against female applicants for a specific job. In the long run, this algorithm can perpetuate, and even amplify, existing gender biases by further widening the gender-based class imbalance.

One such key example of algorithmic bias often cited in literature is regarding the broad use of the COMPAS – or the Correctional Offender Management Profiling for Alternative Sanctions – tool to predict a defendant's risk of recidivism (committing another crime) within two years. COMPAS is more likely to have higher false positive rates for African-American offenders than Caucasian offenders (Mehrabi et al., 2021).

Across the country, scores of similar assessments are given to judges, which injects bias into courts (Angwin, Larson, Mattu, & Kirchner, 2016).

COMPAS is based on data from 7000 people arrested in Broward County, Florida in 2013 and 2014 (Angwin et al., 2016). The response variable, recidivism, was encoded based on who was charged with new crimes over the next two years. Analyses on the predictive efficacy of the COMPAS algorithm found that the algorithm was 61% accurate for a full range of crimes, including misdemeanors, and only 20% of people forecasted to commit violent crimes actually went on to do so. While the overall accuracy rate for the full range of crimes is better than a coin flip, there exists room for enhancing the predictive performance, especially for a decision as critical as whether or not to grant a defendant bail or parole.

The COMPAS algorithm is a color-blind model – race was not included directly as a predictor during its development. However, a statistical analysis showed that even when the effects of race, age, and gender are controlled for through their inclusion as variables in a logistic regression model, Black defendants were still 77% more likely to be predicted at higher risk of committing a future violent crime and 45% more likely to be predicted of committing a future crime of any kind as compared to white defendants (Larson, Mattu, Kirchner, & Angwin, 2016). The table in Figure 1.1 highlights the performance discrepancy across race.

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

Figure 1.1: Prediction Fails Differently for Black v White Defendants (Angwin et al., 2016)

Although the tool has 61% accuracy, Black defendants are almost twice as likely

to be labeled as higher risk without re-offending than White defendants as observed in Figure 1.1. It makes the opposite mistake among White defendants. The reason for this is that classification models are trained to minimize average error, which fits majority populations (Chouldechova & Roth, 2018).

In truth, however, various societal factors contribute to distinct environmental and social realities for Black and White individuals. These factors, including an offender's personal and familial background, as well as their residential environment, are incorporated into the COMPAS tool to forecast recidivism. Consequently, it appears justifiable to adjust the calibration of the relationship between an offender's social context and their propensity for recidivism differently for Black and White offenders, acknowledging these inherent disparities. A group-blind classifier algorithm that fails to do so could unfairly disadvantage one group over the other – COMPAS is just one such algorithm. For example, in the education sector, different factors lead to different SAT performance between students from majority versus minority populations. It would, thus, also seem fair that the relationship between SAT and college admissions be calibrated differently for each demographic group.

Therefore, the question becomes, can we modify these algorithms to be group-blind but also fair? In order to do so, fairness constraints that reduce, or even correct for, algorithmic bias during the modeling process must be set. However, one must first define fairness mathematically, statistically, and quantifiably.

1.2 Statistical Definitions of Fairness

Statistical notions of fairness can be defined at a group level or an individual level. *Group notions* fix a few demographic groups and assess the parity of some statistical measures across all the groups (Chouldechova & Roth, 2018). Note that group

measures, on their own, do not guarantee fairness to individuals or structured subgroups within protected demographic groups, but rather, give guarantees to "average" numbers of protected groups. These notions are the focus of this thesis paper.

Individual notions, on the other hand, are assessed on specific pairs of individuals rather than averaged across groups (Chouldechova & Roth, 2018). In other words, similar individuals should be treated similarly along some defined similarity or inverse distance metrics. Counter-factual fairness, for example, relies on the intuition that a decision is fair towards an individual if it's the same in both the real world and a counter-factual world where the individual belongs to a different demographic group (Mehrabi et al., 2021). This can be impractical, relies on strong assumptions about the data, and approaches the realm of causality (Chouldechova & Roth, 2018). Moreover, there is a gap in literature with regard to individual notions of fairness.

Ultimately, group notions and individual notions are not in conflict per se. Instead, they are on the same spectrum of how much dependence is allowed between predictions and the sensitive attribute (Castelnovo et al., 2022). Subgroup fairness is an alternative notion that intends to obtain the best properties of both, for example, by picking a group fairness constraint and assessing whether it holds over a large collection of subgroups (Mehrabi et al., 2021). Group and individual fairness notions can be defined in both classification settings and regression settings, although most of the literature focuses on fairness within classification.

1.2.1 Group Fairness in Regression Settings

Fair regression is the quantitative notion of fairness of real-valued targets (Agarwal, Dudík, & Wu, 2019). Consider a general prediction setting where the training set consists of X, a feature vector with all the predictor variables, A, the levels of the protected attribute/ demographic group, and Y, the real-valued continuous response

variable. F is a set of possible prediction models, and the goal is to find $f \in F$ that is a good predictive model of Y given X and some fairness constraints. The accuracy of a prediction $f(X) = \hat{Y}$ on Y is measured by the mean squared error (MSE) as the loss function $l(Y, f(X))$. The goal is to minimize $l(Y, f(X))$, hence, maximizing accuracy.

Statistical parity refers to minimizing the expected loss function (MSE) such that the probability that each predicted $f(X) = \hat{Y}$ is above a certain threshold z for each sensitive attribute is the same as the probability over the entire data set, given some margin ϵ_a that is dependent on the protected attribute (Agarwal et al., 2019):

$$\min_{f \in F} E[l(Y, f(X))] \text{ such that } \forall a \in A, z \in [0, 1] : \tag{1.1}$$

$$|P[f(X) \geq z | A = a] - P[f(X) \geq z]| \leq \epsilon_a.$$

This is akin to the classification setting where it may be desirable to have the probability of being in the positive class be above some certain threshold for each group as well as across the entire data set. A similar notion, known as *bounded loss*, requires that the MSE for each group be below some pre-specified level c_a that is dependent on the protected attribute (Agarwal et al., 2019):

$$\min_{f \in F} E[l(Y, f(X))] \text{ such that } \forall a \in A : \tag{1.2}$$

$$E[l(Y, f(X)) | A = a] \leq c_a.$$

1.2.2 Group Fairness in Classification Settings

Group notions of fairness in classification, at the core, refer to treating different groups equally. They aim to remedy or prevent disparate impact, which is a setting where there is unintended disproportionate adverse impact on a particular group

(Chouldechova, 2017). There are three broad notions of observational group fairness: independence, separation, and sufficiency (Castelnovo et al., 2022).

Independence

This fairness definition requires predictions, $\hat{Y}$, to be independent of any sensitive attribute, A, that is, $\hat{Y} \perp\!\!\!\perp A$ (Castelnovo et al., 2022). Thus, it relies only on the distribution of features and decisions, that is, A, X, and $\hat{Y}$, and focuses on the equality of the predictions themselves by satisfying the following equation:

$$P(\hat{Y} = 1|A = a) = P(\hat{Y} = 1|A = b),\ \forall a, b \in A, \tag{1.3}$$

where a, b are the two demographic groups in question.

This definition is also known as *demographic parity*, *statistical parity*, or generally, group fairness. It requires equal positive prediction ratios (PPR), where PPR is the ratio of the probability of a positive prediction $\frac{P(\hat{Y}=1|A=a)}{P(\hat{Y}=1|A=b)}\ \forall\ a, b \in A$, across all demographic group pairings (Castelnovo et al., 2022). In other words, the likelihood of a positive prediction should be the same regardless of the demographic group.

In the COMPAS data set, independence would be satisfied if the probability of predicted recidivism is the same for both Black and White defendants in the data set. That is, the probability that a Black defendant is predicted to recommit a crime within the next two years should be the same as the probability that a White defendant is predicted to recommit a crime.

The visual example in Figure 1.2 illustrates a toy scenario where independence is met (Durahly, 2023).

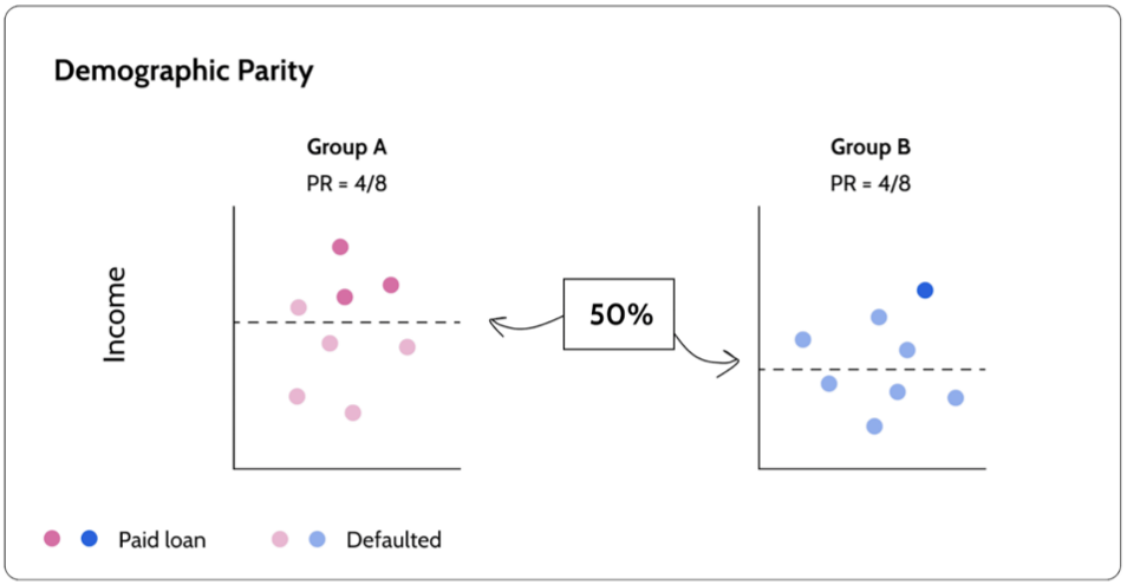


Figure 1.2: An Example of Demographic Parity (Durahly, 2023)

The dashed line represents the decision boundary. In both group A and group B, four out of the eight participants were predicted to repay a loan. The other half of the participants were predicted to default. Notice, however, that the class imbalance in this toy credit lending example results in a higher error rate within group B than group A.

A difference in demographic parity $|P(\hat{Y} = 1|A = a) - P(\hat{Y} = 1|A = b)|$ close to 0 or a ratio $\frac{P(\hat{Y}=1|A=a)}{P(\hat{Y}=1|A=b)}$ close to 1 by some defined margin is considered a fair solution (Castelnovo et al., 2022). To achieve demographic parity, the different demographic groups must be treated differently, which may seem contrary to societal pre-conceived notions of fairness. Therefore, demographic parity should be used when the primary objective is to enforce some form of equality between groups regardless of all other information and when the objectivity of the target variable, Y, is under question, perhaps because of historical biases. This, however, can unknowingly amplify biases if used in the wrong setting. For example, when imposing demographic parity on a hiring algorithm, if qualifications are different across a protected attribute, then less-qualified

candidates may be hired. If these candidates end up being low-performers, then this can perpetuate stereotypes about their demographic group.

In the above example of using a hiring algorithm with gender as the protected attribute, it may then seem fairer to require independence on gender only for men and women with the same rating or qualification, that is, $\hat{Y} \perp\!\!\!\perp A|R$. This is known as *conditional demographic parity* and requires that the following equation is satisfied (Castelnovo et al., 2022):

$$P(\hat{Y} = 1|A = a, R = r) = P(\hat{Y} = 1|A = b, R = r),\ \forall a, b \in A, \forall r. \tag{1.4}$$

This idea can be generalized more to condition on all attributes, that is, $\hat{Y} \perp\!\!\!\perp A|X$. As this is more generalized, however, it begins to satisfy individual fairness (Castelnovo et al., 2022). This type of individual fairness is also referred to as "fairness through unawareness" (FTU), which requires that any protected attributes, or their covariates, are not explicitly used in the decision-making process (Mehrabi et al., 2021). This definition of fairness requires that the following equation be satisfied (Castelnovo et al., 2022):

$$P(\hat{Y} = 1|A = a, X = x) = P(\hat{Y} = 1|A = b, X = x),\ \forall a, b \in A, \forall x \in X. \tag{1.5}$$

Separation

Independence does not make use of the true target Y and simply requires equality of predictions. However, as observed in Figure 1.2, this can lead to different error rates between different groups. In other words, the model is more accurate for one group than it is for another group. Separation precisely focuses on equality of the error rates

and is widely known as the *equality of odds* (Castelnovo et al., 2022). This definition requires the same type I and type II error rates, precisely, the same false positive rate (FPR) and false negative rate (FNR) across all demographic groups. FPR and FNR are defined by:

$$FPR = P(\hat{Y} = 1|Y = 0) = \frac{FP}{FP + TN} \tag{1.6}$$

$$FNR = P(\hat{Y} = 0|Y = 1) = \frac{FN}{TP + FN} \tag{1.7}$$

where FP refers to false positive predictions, TP refers to true positive predictions, FN refers to false negative predictions, and TN refers to true negative predictions. These metrics can be understood through a confusion matrix as in Figure 1.3 (Mohajon, 2021).

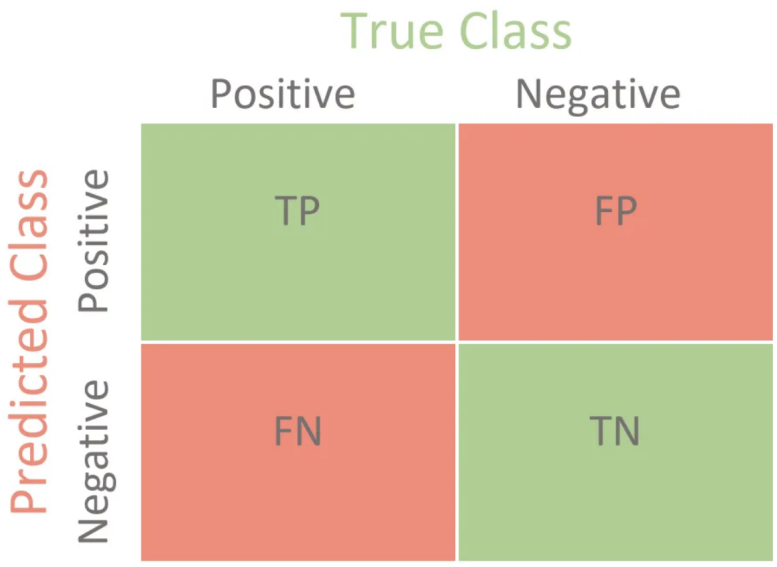


Figure 1.3: A Confusion Matrix (Mohajon, 2021)

In the COMPAS data set, separation would be satisfied if both Black defendants and White defendants had equal error rates. However, as observed in Figure 1.1, Black defendants had an FPR of 45% while White defendants had an FPR of 24% – the FPR

in this context refers to the percentage of times the algorithm predicted the defendants had recidivated when they hadn't. Similarly, Black defendants had an FNR of 28% while White defendants had an FNR of 48% – the FNR in this context refers to the percentage of times the algorithm predicted the defendants had not recommitted a crime when they had.

Separation requires independence of the predictions $\hat{Y}$ and the sensitive attribute A conditioned on the true value of the target variable Y, that is, $\hat{Y} \perp\!\!\!\perp A|Y$ (Castelnovo et al., 2022). In other terms, the following equation must be satisfied:

$$P(\hat{Y} = 1|A = a, Y = y) = P(\hat{Y} = 1|A = b, Y = y),\ \forall a, b \in A,\ y \in \{0, 1\}, \quad (1.8)$$

where 0 is a negative outcome and 1 is a positive outcome. This is a reasonable fairness metric, as long as the objectivity of the target variable is trusted, as it ensures that the model optimizes performance for all groups, not just majority groups.

The visual example in Figure 1.4 illustrates a toy scenario where separation is met (Castelnovo et al., 2022). The dashed line represents the decision boundary. Filled in circles represent positive predictions and empty circles represent negative predictions. The error rates are consistent between both men and women. Notice, however, that the different demographic groups were treated differently to achieve separation as observed in the different decision boundaries (dashed lines).

Figure 1.4: An Example of Equality of Odds (Castelnovo et al., 2022)

There are two relaxed versions of this measure depending on which outcome is most important to predict (Castelnovo et al., 2022):

i) *Predictive Equality*: equality of false positive rates (FPR) across groups:

$$P(\hat{Y} = 1|A = a, Y = 0) = P(\hat{Y} = 1|A = b, Y = 0), \ \forall a, b \in A. \tag{1.9}$$

ii) *Equality of Opportunity*: equality of false negative rates (FNR) across groups:

$$P(\hat{Y} = 0|A = a, Y = 1) = P(\hat{Y} = 0|A = b, Y = 1), \ \forall a, b \in A. \tag{1.10}$$

Sufficiency

Finally, sufficiency takes the perspective of people that receive the same model prediction and requires parity among them regardless of sensitive features (Castelnovo

et al., 2022). This is also knows as *predictive parity* and requires that the precision be the same across sensitive groups, that is, $Y \perp\!\!\!\perp A|\hat{Y}$. In other words, the following equation must be satisfied:

$$P(Y = y|A = a, \hat{Y} = y) = P(Y = y|A = b, \hat{Y} = y),\ \forall a, b \in A,\ \text{for } y \in \{0, 1\}. \quad (1.11)$$

Simply put, the probability of a positive outcome given a positive prediction, and that of a negative outcome given a negative prediction, should be equal across all sensitive groups.

Consistent with this line of reasoning, many fairness metrics can be defined. This begs the fundamental question: can multiple definitions be simultaneously enforced?

1.3 Fairness Conflicts

Because of the way different fairness definitions are defined, it can be impossible to simultaneously enforce multiple definitions, and unexpected behavior may result from a particular definition of fairness. This section highlights some conflicts that arise both in the regression and classification setting.

1.3.1 Fairness Conflicts in Regression

The UFRGS Entrance Exam and GPA Data contains entrance exam scores of students applying to the Federal University of Rio Grande do Sul in Brazil, along with the students' GPAs during their first three semesters at the university (Silva, 2019). Each student's score in nine different entrance exams is used to predict their GPA during their first 3 semesters of study at the university. Gender and race are the protected

attributes.

Taking gender as the protected attribute in a gender-blind model, independence, in this setting, would require that the average predictions be the same for each gender. That is,

$$E[\hat{Y}|G = Male] = E[\hat{Y}|G = Female]. \tag{1.12}$$

Independence is violated if, on average, a model predicts a higher or lower GPA based on gender.

Separation, on the other hand, would require that the average error of predictions be the same for each gender. In defined notion,

$$E[\hat{Y} - Y|G = Male] = E[\hat{Y} - Y|G = Female]. \tag{1.13}$$

Separation is violated if, on average, the model over-predicts for one gender but under-predicts for another gender or the model either over-predicts or under-predicts more for one gender.

However, a study found that because male and female applicants had different GPAs in the original data set, these two fairness definitions cannot be simultaneously satisfied (P. Thomas, 2020). A similar result in the Section 1.3.2 will explain this in a mathematically tractable way when working in a classification setting.

1.3.2 Fairness Conflicts in Classification

Define prevalence p as the probability of a positive outcome given the demographic group (Chouldechova, 2017). It directly relates to the class distribution of the outcome. $p \in (0, 1)$ and can be denoted by:

$$p_a = P(Y = 1|A = a). \tag{1.14}$$

Further define the positive predictive value (PPV) of a prediction as the probability of a positive outcome given a positive prediction (Chouldechova, 2017):

$$PPV(\hat{Y}|A = a) \equiv P(Y = 1|\hat{Y} = 1, A = a). \tag{1.15}$$

Similarly, the negative predictive value (NPV) of a prediction is the probability of a negative outcome given a negative prediction and can be denoted as:

$$NPV(\hat{Y}|A = a) \equiv P(Y = 0|\hat{Y} = 0, A = a). \tag{1.16}$$

Sufficiency would require equal PPV and equal NPV across the different demographic groups. Note that NPV and PPV can be computed from a confusion matrix (Figure 1.3) (Saeed, Alireza, Mohamed, & Ahmed, 2015):

$$PPV = \frac{TP}{TP + FP} \; ; \; NPV = \frac{TN}{TN + FN}. \tag{1.17}$$

Now, given values of the $PPV \in (0, 1)$ and $p \in (0, 1)$, it can be shown that (Chouldechova, 2017):

$$FPR = \frac{p}{1 - p}\frac{1 - PPV}{PPV}(1 - FNR). \tag{1.18}$$

Appendix A provides the details for the derivation of this equation. However, its direct implication is that if the prevalence differs between two groups, then it is impossible to satisfy sufficiency (equal PPV across all groups) and separation (equal FPR and FNR across all groups) simultaneously. For example, in the COMPAS data set, if recidivism rates differ between Black and White offenders, then an algorithm that guarantees predictive parity/ equal precision for both Black and White offenders cannot simultaneously satisfy equality of odds. Indeed, the recidivism rate for Black

defendants in the data is 51%, compared to 39% for White defendants, and hence, the disparate impact of the COMPAS tool as observed in Figure 1.1 (Chouldechova, 2017).

Figure 1.2 illustrates a similar conflict between independence and separation. Satisfying independence resulted in an imbalance of error rates between group A and group B because of the difference in the prevalence of loan repayment between both groups.

Unfortunately, the distribution of the outcome of interest often differs for different demographic groups, posing the all-important question: how can fairness be achieved in the face of this conflict?

1.3.3 On Fairness Conflicts

As observed, disparate impact can result from the use of a prediction tool that is perceived to be free from predictive bias. Just because an algorithm satisfies a particular definition of fairness doesn't infer that the algorithm is *fair* in every sense of that word. Balancing overall error rates alone is not enough as it does not produce models that are free from bias or that guarantee fairness at finer levels of granularity. This highlights the need for human value and domain expertise in defining fairness within the context of a particular problem before the fairness constraints can be set. Chapter 2 introduces a framework for setting these constraints.

Chapter 2 Seldonian Algorithms

Chapter 1 introduced the problem of algorithmic bias, discussed existing statistical definitions of fairness both in regression and classification settings, and finally, highlighted fairness conflicts that can arise in certain settings. Of important note is that there are a plethora of fairness definitions that have been developed in statistical machine learning, many of which have been shown to be incompatible in ways similar to the illustration in Appendix A. In any effort to enforce fairness on machine learning models, a critical first step is to define what fairness means in the specific context (P. Thomas, 2020). This responsibility falls on domain experts, social scientists, and regulators. Once there is consensus on that, machine learning researchers can work to develop appropriate algorithms that enforce the chosen definition of fairness. The Seldonian framework, introduced in this chapter, offers one such way to place probabilistic fairness constraints on traditional algorithms. However, because Seldonian algorithms place constraints on traditional machine learning (ML) algorithms, an initial in-depth understanding of the standard approach is key. Section 2.1 discusses the typical ML approach before diving into the Seldonian framework in Section 2.2.

2.1 The Standard Machine Learning Approach

When designing a machine learning algorithm, the first step is to mathematically define what the algorithm should do, in other words, the goal of the algorithm (P.

S. Thomas et al., 2019b). At an abstract level, this goal is identical for all machine learning problems: find a solution θ^*, within some feasible set Θ, that maximizes some objective function $f : \Theta \to \mathbf{R}$, where $\mathbf{R}$ is the set of real numbers. Precisely, the goal of the algorithm is to search for an optimal solution

$$\theta^* \in \arg\max_{\theta \in \Theta} f(\theta). \tag{2.1}$$

For example, let X and Y be dependent real-valued random variables in a regression setting with the goal of estimating Y given X. In this setting, Θ is the set of feasible functions that model the relationship between X and Y. Feasible functions are of the form $\theta(X) = \beta_0 + \beta_1 X = \hat{Y}$. Each function $\theta \in \Theta$ takes a real number as input and produces a real number as output; therefore, $\theta : \mathbf{R} \to \mathbf{R}$. A reasonable objective function would then be the negative mean squared error (MSE):

$$f(\theta) := -E[(\theta(X) - Y)^2]. \tag{2.2}$$

In this case, minimizing MSE is equivalent to maximizing -MSE, defining the goal of the regression algorithm as finding the solution with the least average error. Note that the true value of $f(\theta)$ is unknown and can only be estimated from the data (P. S. Thomas et al., 2019a). For a sample with n observations, that is, (x_i, y_i) for $i = 1, 2, ..., n$, the objective function can be estimated by:

$$\hat{f(\theta)} = -\frac{1}{n} \sum_{i=1}^{n} (\theta(x_i) - y_i)^2. \tag{2.3}$$

However, defining objective functions in this way can sometimes lead to undesirable behavior as illustrated in Section 2.1.1.

2.1.1 Limitations of the Standard Approach

Consider a linear regression example to predict the qualifications of job applicants based on information on their resumes. Let G encode the gender of each applicant, with $G = 0$ if the applicant is female and $G = 1$ if the applicant is male. Let X encode a summary measure of an applicant's qualification based on information on their resume – a simple example would be a measure of how many job-relevant key words appear on their resume. Let Y encode their actual qualification for the job as determined by their observed performance.

If this linear regression estimator is designed to be used to filter which resumes submitted to a company will be forwarded for human review, it is worthwhile to ensure that the algorithm does not produce racist or sexist behavior. Drawing from definitions in Chapter 1.2, it might be less important to ensure that the algorithm, on average, has the same predictions for applicants of both genders because the distribution of qualifications may be different for both genders. However, of more concern is whether the algorithm, on average, predicts too high for one gender and too low for the other gender.

Suppose that the data has the following distribution: $Y \sim N(1, 1)$ if $G = 0$ and $Y \sim N(-1, 1)$ if $G = 1$, that is, Y is a normal variable $N(\mu, \sigma)$ with different means μ for different genders but with the same standard deviation σ for both genders. Further define $X \sim N(Y, 1)$, that is, an applicant's resume quality is equal to their true qualification plus some random noise. Figure 2.1 displays a scatterplot of 1000 such data points, 500 from each gender. The black solid line is the least squares fit on this data using a gender-blind model.

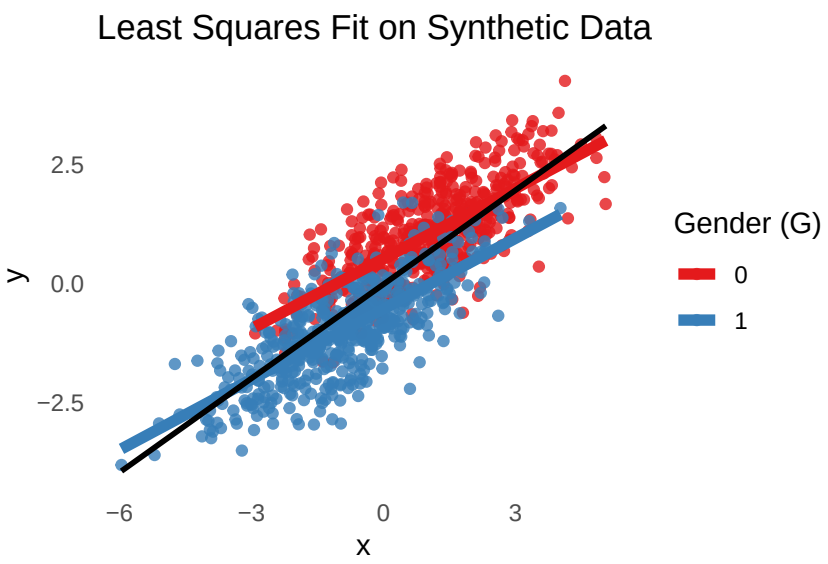


Figure 2.1: Least Squares Fit on Synthetic Data Drawn from Different Distributions

The least squares fit on Figure 2.1 is impartial to an observation's gender with an objective to make the most accurate predictions. While it may be expected that impartiality would produce fair results, observe that the linear model tends to over-predict if $G = 1$ and under-predict if $G = 0$, producing discriminatory behavior. In fact, by defining a discrimination statistic, $d(\theta)$, that measures whether the model satisfies separation (equal error rates), the discrimination statistic for the synthetic data set in Figure 2.1 can be shown to be -0.719, suggesting that the model predictions are in favor of $G = 1$:

$$d(\theta) = E[\hat{Y} - Y|G = 0] - E[\hat{Y} - Y|G = 1]. \tag{2.4}$$

In crucial applications such as hiring, this is concerning and highlights how group-blind linear regression algorithms designed using the standard approach and following statistical best practices can result in predictions that systematically discriminate against a demographic group.

2.1.2 Potential Remedies

In an attempt to remedy this undesirable behavior, a number of approaches can be taken. One potential remedy is to identify the root cause of the undesirable behavior such as class imbalance in the training data, bias in the data set, the choice of linear estimator, the model's blindness to the demographic group, or insufficient data, to name a few (P. S. Thomas et al., 2019b). For instance, in the example set up in Section 2.1.1 and displayed in Figure 2.1, the root cause of the discriminatory behavior when using ordinary least squares linear regression was the fact that the objective function was designed to minimize MSE, which was at odds with minimizing the discrimination statistic. However, even though it might be possible to determine and correct the root cause of the undesirable behavior, doing so can be difficult, error-prone, and require extensive data analysis, rendering the central goal of machine learning algorithms, which are designed to automate and make decision-making processes simpler, obsolete.

Assuming that the problem is with the objective function and provided that detailed knowledge of the problem is available, hard constraints may be placed on the objective function, for example, requiring that MSE is minimized only on the set of solutions with a discrimination value $d(\theta)$ less than some value ϵ (P. S. Thomas et al., 2019b). Additionally, rather than placing hard constraints on the set of solutions, soft constraints that penalize undesirable behavior may also be placed on f, the objective function (Boyd & Vandenberghe, 2004). Although such penalty functions can be effective, they require a careful choice of the value of the parameter λ that places relative importance on the objective function and the constraint. For the linear regression example, the new objective function with a soft constraint would now be:

$$f(\theta) = -MSE(\theta) - \lambda d(\theta). \tag{2.5}$$

Observe that as λ increases, MSE increases and the discrimination statistic decreases. Cross-validation techniques can be employed to find optimal values for λ. Other remedies include maximizing multiple objective functions or allowing constraints on the probability that a solution with undesirable behavior will be returned, both of which may require detailed knowledge of the application problem and underlying distribution of the data (P. S. Thomas et al., 2019b).

In principle, there might be definitions of Θ or f that prevent the algorithm from converging on solutions that exhibit undesirable behavior (P. S. Thomas et al., 2019a). However, in practice and as explained, this might require extensive domain expertise and data analysis in order to properly balance the relative importance of the objective function and the constraints, which can be at odds with each other. These techniques may also require knowledge of the probability distribution from which the data is sampled, which is not always available and limits applications to parametric statistics.

Seldonian algorithms address this problem precisely by allowing probabilistic constraints on undesirable behavior to be placed more easily without detailed knowledge of the specific problem or the distribution of the data, shifting the burden from the domain experts who use these tools to the experts in ML and statistics (P. S. Thomas et al., 2019a). It's named after Isaac Asimov's fictional character, Hari Seldon [1] (Asimov, 1994). It's important to note that while Seldonian algorithms allow for more seamless implementation in practice, domain experts are still needed to define the relevant fairness constraints for a given context.

[1] In the fictional book, Hari Seldon was a resident of a fictional planet where he develops psychohistory, an algorithmic science that allows him to predict the future in probabilistic terms.

2.2 The Seldonian Framework

The first step of the Seldonian framework is to define mathematically the goal of the algorithm design (P. S. Thomas et al., 2019b). Define $\mathbf{D}$ as the set of all possible inputs (data sets) to the algorithm. Θ, as previously defined, is the set of all possible outputs (solutions) of the algorithm. Each solution is referred to as $\theta \in \Theta$. D is the data set (input) given to the algorithm and is the only random variable. Now, $a : \mathbf{D} \to \Theta$ is a machine learning algorithm which takes in a data set $D \in \mathbf{D}$ as an input and returns a solution $\theta \in \Theta$ as an output. $\mathbf{A}$ is the set of all possible machine learning algorithms. Synthesizing this, $f : \mathbf{A} \to \mathbf{R}$ is the objective function of the algorithm design, where $f(a) \in \mathbf{R}$ is a real-valued measure of the utility of the algorithm, such as the value of the objective function for the solution returned by this algorithm. This objective function is optimized – either minimized or maximized – to select a desired machine learning algorithm from the set $\mathbf{A}$.

Contrary to the standard ML approach, however, n behavioral constraints can then be specified (P. S. Thomas et al., 2019b). Specifically, $(g_i, \delta_i)_{i=1}^n$ can be defined as a set of n constraints, each of which contains a constraint function $g_i : \Theta \to \mathbf{R}$ and a desired confidence level δ_i. The constraint function takes in a solution returned from the chosen machine learning algorithm as an input and returns a real value encoding the "fairness" of the algorithm according to the fairness definition defined by the function. $(g_i, \delta_i)_{i=1}^n$ is defined such that:

- The i^{th} constraint function measures an undesirable behavior. Specifically, $\theta \in \Theta$ produces undesirable behavior if and only if $g_i(\theta) > 0$. This is to ensure that undesirable behavior is defined in a mathematically tractable way such as how the discrimination statistic $d(\theta)$ was defined in Section 2.1.1.

- The i^{th} confidence level specifies the maximum probability that an algorithm can return a solution θ where $g_i(\theta) > 0$. In other words, $1 - \delta_i$ specifies the minimum probability that desirable behavior ($g_i(\theta) \leq 0$) is met. Smaller values of δ_i are preferred.

In summary, a Seldonian algorithm ensures that for all $i \in \{1, 2, \ldots, n\}$:

$$P(g_i(a(D)) \leq 0) \geq 1 - \delta_i. \tag{2.6}$$

Section 2.2.1 goes into further detail about the Seldonian framework and how these probabilistic behavioral constraints are guaranteed.

2.2.1 The Seldonian Optimization Problem

As detailed, the Seldonian framework is different from current potential remedies of undesirable behavior because it defines a search over a possible set of algorithms with constraints, rather than over a possible set of solutions. This means that the constraints require that the probability that a machine learning algorithm returns an unsafe solution be bounded by some desired level of confidence, rather than the probability that a solution itself is unsafe. In summary, the Seldonian optimization problem (SOP) can be written as (P. S. Thomas et al., 2019a):

$$\arg\max_{a \in \mathbf{A}} f(a) \tag{2.7}$$

$$\text{s.t. } \forall\ i \in \{1, 2, \ldots, n\},\ P(g_i(a(D)) \leq 0) \geq 1 - \delta_i.$$

A Seldonian algorithm a, thus, returns, with high probability, a solution that guarantees desirable behavior. If one were to apply machine algorithm a to obtain a solution from a large number of different data sets D drawn from the same distribution,

then it would be expected that at most $100\delta_i\%$ solutions (models) would produce undesirable behavior.

Taking the previous regression example and turning it into a Seldonian optimization problem using the discrimination statistic in Section 2.1.1, f would still be an objective function like the MSE, Θ would still be the set of all possible linear models, and D would be the data set as described. There would be 1 behavioral constraint, $g_1(a(D)) = |d(a(D))| - \epsilon$, to guarantee with probability at least $1 - \delta_1$, that the absolute value of the discrimination statistic would be at most ϵ, where ϵ and δ_1 are chosen by domain experts based on the specific application. Note that the user of the machine learning algorithm need not perform data analysis to determine whether $g_1(\theta) \leq 0$ for a particular solution $\theta \in \Theta$ returned. The computation algorithm guarantees this with some desired level of probability.

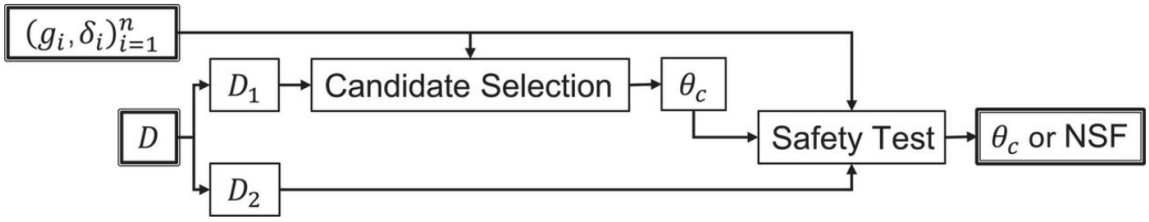


Figure 2.2: Overview of the Seldonian Framework (P. S. Thomas et al., 2019a)

Figure 2.2 illustrates how this is achieved at a high level. A Seldonian algorithm takes in n behavioral constraints $(g_i, \delta_i)_{i=1}^n$ and a data set D as the inputs and returns either a solution (model) θ or NSF, which means "No Solution Found". An NSF result means no algorithm was found that returned a model which satisfied the behavioral constraints with the desired probability, so solutions are not guaranteed when employing Seldonian algorithms.

First, the data D is partitioned into 2 sets D_1 and D_2 that essentially serve as

train and test sets, respectively. D_1 is then passed through the candidate selection mechanism, which performs a search over algorithms to settle on a candidate solution θ_c. θ_c is selected not only so that it optimizes the primary objective function f, but also so that it is predicted to pass the subsequent safety test. D_2 is then passed through the safety test to check whether θ_c indeed satisfies the n behavioral constraints with the desired confidence for each, that is $P(g_i(\theta_c) \leq 0) \geq 1 - \delta_i$ for each constraint $i \in \{1, 2, \ldots, n\}$. If so, θ_c is returned as the desired solution, and otherwise, NSF (P. S. Thomas et al., 2019a).

Note that finding exact confidence bounds may be impractical and require large amounts of data. Quasi-Seldonian algorithms, thus, are an extension of this idea that rely on standard statistical tools to transform sample statistics computed from D into approximate bounds on the probability of undesirable behavior (P. S. Thomas et al., 2019a). Section 2.2.2 discusses the statistical framework employed to achieve this.

2.2.2 Quasi-Seldonian Algorithms

Recall that the Seldonian goal is to create an algorithm a that is an approximate solution to the Seldonian optimization problem defined in Equation 2.7. This framework is non-parametric and relies on exact values of the objective function and fairness constraints. However, these values are often unknown and need to be estimated from the data provided.

For example, $f(a)$, the objective function, can be estimated from the data provided such that $\hat{f} : \Theta \text{ x } \mathbf{D} \rightarrow \mathbf{R}$ serves as a measure of the utility of the algorithm that returns a solution θ when given input D (P. S. Thomas et al., 2019a). In a linear regression setting, the MSE for a data set of size m can be estimated by

$$\hat{f}(\theta, D) = -\frac{1}{m} \sum_{i=1}^{m} (\hat{y}(X_i, \theta) - Y_i)^2. \tag{2.8}$$

In a similar fashion, the following section discusses how the candidate selection and safety test mechanisms further employ statistical estimation techniques to estimate the confidence bounds of the fairness constraints and probabilistically guarantee safe behavior.

The Safety Test Mechanism

Seldonian algorithms ensure that $P(g_i(\theta_c) \leq 0) \geq 1 - \delta_i$ for each constraint $i \in \{1, 2, \ldots, n\}$ and the safety test mechanism is the component that verifies whether these behavioral constraints actually hold (P. S. Thomas et al., 2019a). This is achieved by computing an upper bound for each $g_i(\theta)$ using the data and a confidence interval derived from the Student t-statistic. If the high confidence upper bound is less than or equal to 0, then the solution is safe to return, otherwise, no solution will be returned.

Let $X = (X_1, \ldots, X_m)$ be m independent and identically distributed ($i.i.d.$) random variables. Under the assumption that $\frac{1}{m}\sum_{i=1}^{m} X_i$ is normally distributed or if m is sufficiently large – by the Central Limit Theorem –, then the Student t-statistic can be used to compute an upper bound of the expected value of these random variables as follows:

$$P(E[X_1] \leq \hat{\mu}(X) + \frac{\hat{\sigma}(X)}{\sqrt{m}} t_{1-\delta,m-1}) \geq 1 - \delta, \tag{2.9}$$

where

- $\hat{\mu}(X) = \bar{X}$ and $\hat{\sigma}(X) = s$ are the sample mean and standard deviation, respectively, of a vector X. That is, $\hat{\mu}(X) = \frac{1}{m}\sum_{i=1}^{m} X_i = \bar{X}$ and $\hat{\sigma}(X) = \sqrt{\frac{\frac{1}{m}\sum_{i=1}^{m}(X_i - \hat{\mu}(X))^2}{m-1}} = s$.
- $t_{1-\delta,m-1}$ is the $100(1-\delta)$ percentile of the Student t-distribution with $m-1$ degrees of freedom.

Before constructing the safety test mechanism, recall from Section 2.2.1 that (P. S.

Thomas et al., 2019b):

- The safety test will be applied to a single solution θ_c selected by the candidate selection mechanism. This process is explained in the following section.
- The safety data, D_2, is used to verify that the behavioral constraints hold.
- $\hat{g}_i(\theta_c, D_2) = (\hat{g_{i,1}}(\theta_c, D_2), \ldots, \hat{g_{i,m}}(\theta_c, D_2))$ contains m $i.i.d$ values of $\hat{g}_i(\theta_c)$ for each of the m observations in D_2. $|D_2|$ will be used to denote the number of observations in D_2 for consistency in notation.
- $E[\hat{g}_i(\theta_c, D_2)] = g_i(\theta_c)$.

By substituting the respective pieces into the Student t high confidence upper bound discussed above, then:

$$P(g_i(\theta_c) \leq \hat{\mu}(\hat{g}_i(\theta_c, D_2)) + \frac{\hat{\sigma}(\hat{g}_i(\theta_c, D_2))}{\sqrt{|D_2|}} t_{1-\delta_i, |D_2|-1}) \geq 1 - \delta_i. \tag{2.10}$$

Notice that $\hat{\mu}(\hat{g}_i(\theta_c, D_2)) + \frac{\hat{\sigma}(\hat{g}_i(\theta_c, D_2))}{\sqrt{|D_2|}} t_{1-\delta_i, |D_2|-1}$ is an upper bound of the confidence interval with confidence $1 - \delta_i$. If this upper bound is less than or equal to zero, then the i^{th} behavioral constraint $g_i(\theta_c)$ is less than or equal to zero with at least probability $1 - \delta_i$. Therefore, θ_c is only returned if $\hat{\mu}(\hat{g}_i(\theta_c, D_2)) + \frac{\hat{\sigma}(\hat{g}_i(\theta_c, D_2))}{\sqrt{|D_2|}} t_{1-\delta_i, |D_2|-1} \leq 0$. Specifically, this holds only under the assumption that $\hat{\mu}(\hat{g}_i(\theta_c, D_2))$ is normally distributed or if the size of the safety data D_2 is sufficiently large, hence the name *quasi*-Seldonian (P. S. Thomas et al., 2019a). The next section now discusses precisely how θ_c is selected before being passed into the safety test mechanism.

The Candidate Selection Mechanism

With the safety test in place, any algorithm will be Seldonian regardless of how θ_c is computed, as long as θ_c is computed using a different subset of the data, hence the

partition into D_1 (candidate data) and D_2 (safety data) (P. S. Thomas et al., 2019b). However, if θ_c is computed using the standard ML approach, then it will likely be unsafe as was illustrated in Section 2.1.1, resulting in an NSF output. Instead, θ_c will be computed as follows:

$$\theta_c \in \arg\max_{\theta \in \Theta} \hat{f}(\theta, D_1) \tag{2.11}$$

$$\text{s.t. } \theta_c \text{ is predicted to pass the safety test.}$$

Thus, only solutions likely to pass the safety test will be considered by predicting the result of the safety test using D_1 (the candidate data) instead of D_2. In formal notation,

$$\theta_c \in \arg\max_{\theta \in \Theta} \hat{f}(\theta, D_1) \tag{2.12}$$

$$\text{s.t. } \forall\, i \in \{1, 2, \ldots, n\},\ \hat{\mu}(\hat{g}_i(\theta_c, D_1)) + \frac{\hat{\sigma}(\hat{g}_i(\theta_c, D_1))}{\sqrt{|D_2|}} t_{1-\delta_i, |D_2|-1} \leq 0.$$

Notice that while the sample mean $\hat{\mu}$ and the sample standard deviation $\hat{\sigma}$ are computed over D_1, the size of D_2 is still used to correct the standard deviation and compute the Student t percentile, in order to ensure that the solution is properly predicted to pass the safety test.

The process defined can work well when the objective function f and the behavioral constraints are aligned. However, when they are in conflict, the candidate selection mechanism tends to be over-confident that θ_c will pass the safety test and a safe solution will be returned (P. S. Thomas et al., 2019b). Doubling the width of the confidence level is a proposed solution to produce more conservative predictions and better guarantees of θ_c passing the safety test. Therefore, a black-box optimization algorithm is used to compute

$$\theta_c \in \arg\max_{\theta \in \Theta} \hat{f}(\theta, D_1) \tag{2.13}$$

$$\text{s.t. } \forall\ i \in \{1, 2, \ldots, n\},\ \hat{\mu}(\hat{g}_i(\theta_c, D_1)) + 2\frac{\hat{\sigma}(\hat{g}_i(\theta_c, D_1))}{\sqrt{|D_2|}} t_{1-\delta_i, |D_2|-1} \leq 0.$$

This concludes the discussion on the Seldonian theoretical framework at a high level. To conclude this chapter, Section 2.3 walks through how this is implemented computationally using a toy regression example.

2.3 Toy Example: A Quasi-Seldonian Regression Algorithm

The tutorial in this section follows the presentation by P. S. Thomas (n.d.) on the AI Safety webpage focusing on the key computational aspects of the Seldonian framework. Consistent with the linear regression set-up in this chapter, consider $X, Y \in \mathbf{R}$ as two dependent random variables with the goal of estimating Y given X through a sample of m observations. X is drawn synthetically from a $N(0, 1)$ distribution and Y is dependent on X with a $N(X, 1)$ distribution. Figure 2.3 displays 5000 such points. $\hat{y}(X, \theta) = \theta_1 X + \theta_2$ and $MSE = E[(\hat{y}(X, \theta) - Y)^2]$ are computed as previously defined.

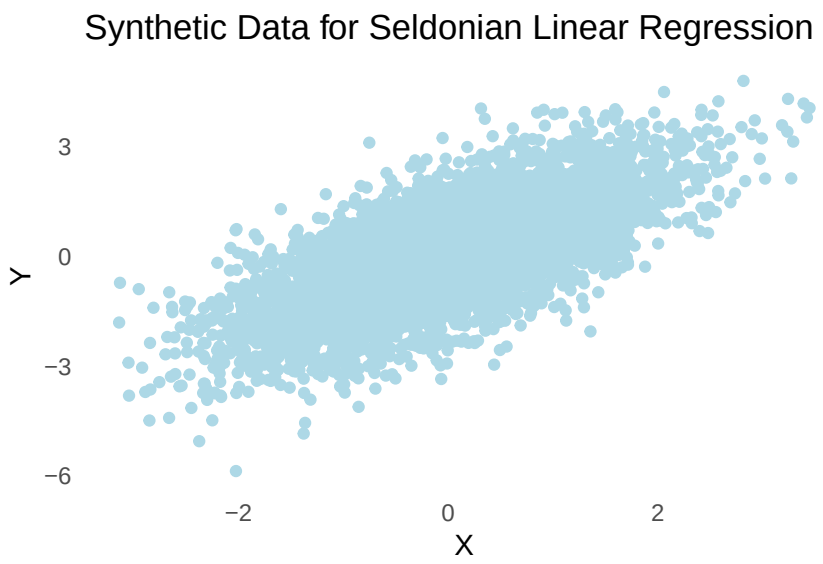


Figure 2.3: Synthetic Data for Quasi-Seldonian Linear Regression Tutorial

In this example, the goal of the linear regression algorithm is two-fold: to minimize MSE (or equivalently, maximize -MSE) while ensuring, with probability at least 0.9, that $1.25 < MSE < 2$. Note that this behavioral constraint is not practical in an application setting, but it's deliberately designed to be at odds with the objective function in order to test behavior when such conflict arises. Additionally, it'll be simple to verify satisfaction of the behavioral constraint for demonstration purposes. The behavioral constraint needs to be mathematically represented in a way that defines $g(\theta) \leq 0$ as safe behavior. Therefore, n will be set to 2 such that:

- $g_1(\theta) = MSE(\theta) - 2.0$; $\delta_1 = 0.1$.
- $g_2(\theta) = 1.25 - MSE(\theta)$; $\delta_2 = 0.1$.

Unbiased estimates of the MSE and each $g_i(\theta)$ will be computed from the data set as elucidated in Section 2.2.2. The Python code used to implement and compute the quasi-Seldonian linear regression algorithm is displayed in detail in Appendix B. To reduce computational burden, all the computation was performed on the Amherst

College High-Performance Computing System. In this case, a solution that minimizes the MSE while satisfying the 2 behavioral constraints, $g_1(\theta)$ and $g_2(\theta)$, was found – the MSE was 1.385. Figure 2.4 visually compares the quasi-Seldonian solution (blue) with the ordinary least squares solution (red).

```
A solution was found: [0.6050927032, 1.0201681981]
fHat of solution (computed over all data, D): -1.3854947860210223
```

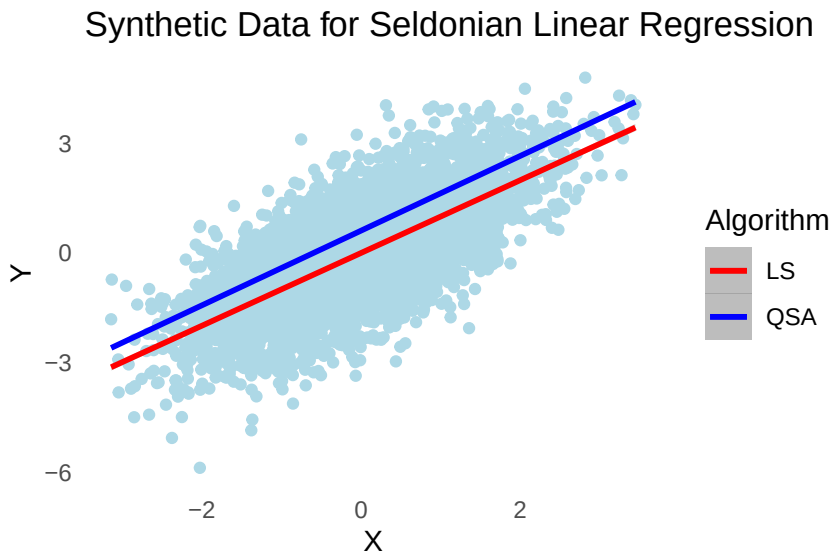


Figure 2.4: Quasi-Seldonian Linear Regression

To conclude this chapter, Section 2.3.1 scales this process to repeatedly run a quasi-Seldonian linear regression algorithm using different amounts of data and analyzes the results.

2.3.1 Experimentation

Consistent with the set-up in Section 2.3 and following the presentation by P. S. Thomas (n.d.), the aim of the experimentation in this section is to assess three aspects of the quasi-Seldonian linear regression algorithm defined in Section 2.3: performance loss, frequency of solutions, and frequency of undesirable behavior for varying sample sizes.

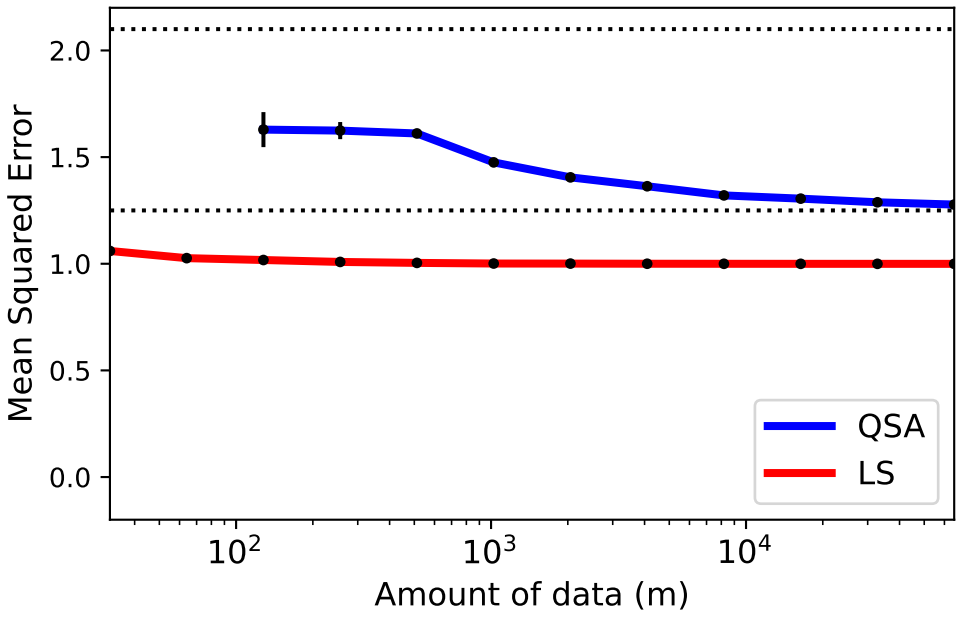


Figure 2.5: QSA Experiment: Performance Loss

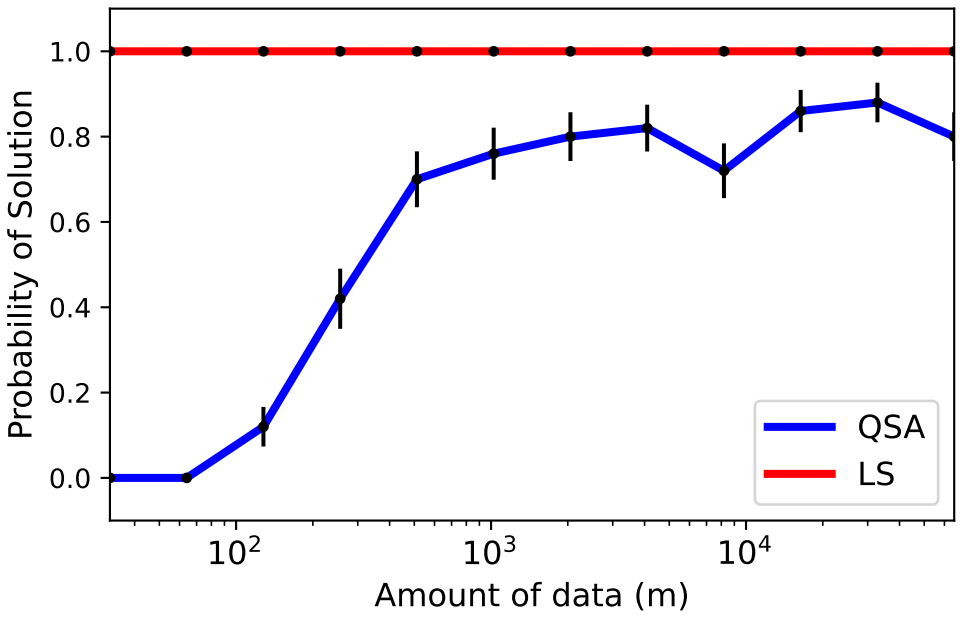


Figure 2.6: QSA Experiment: Probability of a Solution

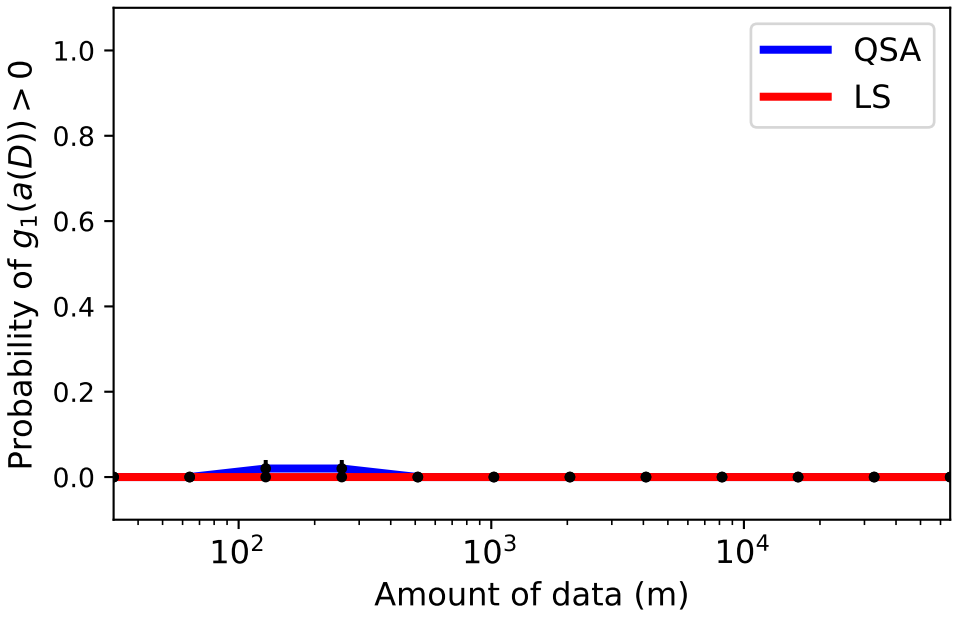


Figure 2.7: QSA Experiment: Satisfaction of 1st Behavioral Constraint

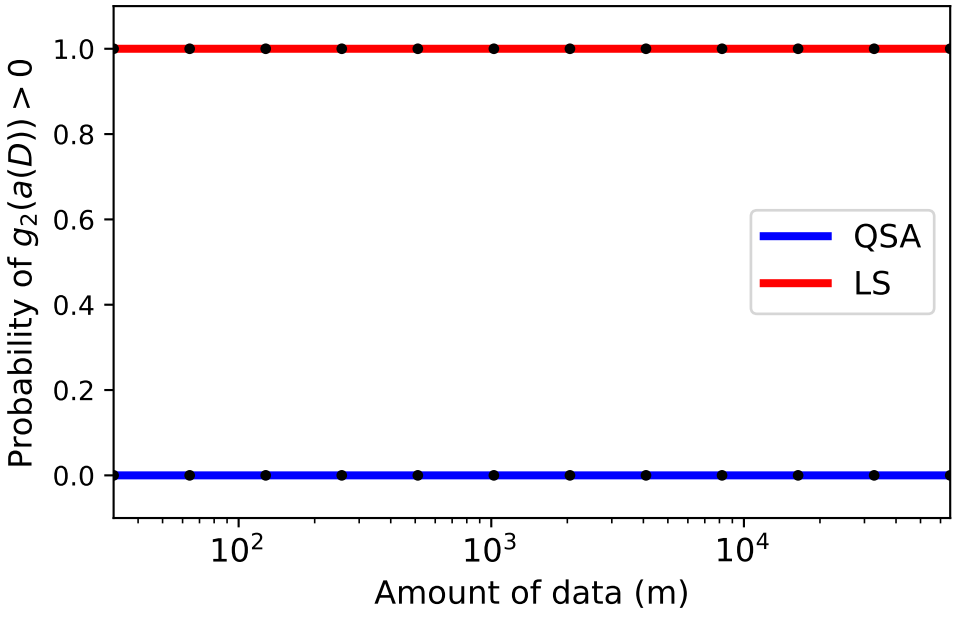


Figure 2.8: QSA Experiment: Satisfaction of 2nd Behavioral Constraint

Each of the figures show how different properties of the quasi-Seldonian linear

regression (QSLR) algorithm vary for different data set sizes, m. The horizontal axis is on a logarithmic scale, with values of m starting at $m = 32$ and going up to $m = 65,536$, doubling each time such that $m \in \{32, 64, 128, \ldots, 65536\}$. 50 trials are run for each value of m. Each figure plots the mean and includes a standard error bar to visualize the variance of results.

Performance Loss

A Seldonian algorithm that has probabilistic guarantees of safe or fair behavior will typically perform worse, with regard to accuracy or error, than an algorithm that is purely focused on optimizing performance (P. S. Thomas, n.d.). In Figure 2.5, the dotted lines represent the desired range for the MSE as defined by the behavioral constraints set in Section 2.3. In this setting, MSE was forced to be higher. Notice that the QSLR did not return a solution for small amounts of data, but when a solution was returned, it was always within the MSE bounds set by the constraints. Additionally, the bigger the sample size, the closer the MSE was to the lower boundary. This highlights the role that the primary objective plays by encouraging solutions with lower error, albeit still within the desired window. In a different setting where the behavioral constraints are not at odds with the primary objective function, it would be worthwhile to investigate performance loss when using a QSLR algorithm rather than ordinary least squares (OLS). Chapter 3 attempts to answer this question by investigating performance loss for Seldonian algorithms in classification settings.

Probability of a Solution

Quasi-Seldonian algorithms don't always return a solution, especially with little data because there is insufficient confidence that any solution would satisfy the behavioral constraints. Figure 2.6 illustrates that with more data, there is a higher probability of

a solution. Nevertheless, there isn't a definitive threshold ensuring a solution for data sizes exceeding it. However, when provided with ample data, the results suggested that the likelihood of obtaining a solution tends to exceed 0.8.

Probability of Undesirable Behavior

Finally, Figures 2.7 and 2.8 plot the probability that each algorithm produced undesirable behavior, that is, the probability that each algorithm had a solution with $g_1(a(D)) > 0$ as in Figure 2.7 and the probability that each algorithm had a solution with $g_2(a(D)) > 0$ as in Figure 2.8. Recall that $\delta_1 = \delta_2 = 0.1$, so for the QSA solution, the probability of undesirable behavior should lie below 0.1, or at least around 0.1. QSA always satisfied both constraints with probability at least 0.9 when a solution was returned. Notice, however, that since OLS does not take behavioral constraints into account, it frequently violates the second behavioral constraint that requires MSE to be greater than 1.25. This is expected because this constraint is in conflict with the primary objective function.

The toy example in this Chapter illustrates some of the desirable qualities of Seldonian (or quasi-Seldonian) algorithms and some of the limitations. The focus was in the quantitative setting when using linear regression. Chapter 3, on the other hand, delves deeper into the properties of Seldonian algorithms with a particular focus on classification.

Chapter 3 Seldonian Algorithms for Classification

Appendix A Sufficiency v Separation Fairness Conflict Equation

Recall from Chapter 1 that (Castelnovo et al., 2022):

$$PPV = P(Y = 1|\hat{Y} = 1),$$

$$FPR = P(\hat{Y} = 1|Y = 0),$$

$$FNR = P(\hat{Y} = 0|Y = 1),$$

$$p = P(Y = 1).$$

Using Bayes' rule,

$$PPV = P(Y = 1|\hat{Y} = 1) = \frac{P(\hat{Y} = 1|Y = 1)P(Y = 1)}{P(\hat{Y} = 1|Y = 1)P(Y = 1) + P(\hat{Y} = 1|Y = 0)P(Y = 0)}$$

$$\Rightarrow PPV = \frac{P(\hat{Y} = 1|Y = 1)p}{P(\hat{Y} = 1|Y = 1)p + P(\hat{Y} = 1|Y = 0)(1 - p)}$$

$$\Rightarrow PPV = \frac{(1 - FNR)p}{(1 - FNR)p + FPR(1 - p)}$$

$$\Rightarrow (1-FNR)p + FPR(1-p) = \frac{(1-FNR)p}{PPV}$$

$$\Rightarrow FPR(1-p) = \frac{(1-FNR)p}{PPV} - (1-FNR)p$$

$$\Rightarrow FPR = \frac{(1-FNR)p}{PPV(1-p)} - \frac{(1-FNR)p}{(1-p)}$$

$$\Rightarrow FPR = \frac{p}{1-p}\left[\frac{(1-FNR)}{PPV} - (1-FNR)\right]$$

$$\Rightarrow FPR = \frac{p}{1-p}\left[\frac{(1-FNR) - PPV(1-FNR)}{PPV}\right]$$

$$\Rightarrow FPR = \frac{p}{1-p}\left[\frac{(1-FNR)(1-PPV)}{PPV}\right]$$

$$\Rightarrow FPR = \frac{p}{1-p}\frac{1-PPV}{PPV}(1-FNR)\blacksquare.$$

A similar equation can be derived relating $NPV = P(Y = 0|\hat{Y} = 0)$ and both FPR and FNR.

Additionally, in conventional statistics notation, the sensitivity of a prediction tool can be defined as $P(\hat{Y} = 1|Y = 1) = 1 - FNR$ and its specificity can be defined as $P(\hat{Y} = 0|Y = 0) = 1 - FPR$. Given a prevalence p, sensitivity s_e, and specificity s_p, then:

$$PPV = \frac{s_e p}{s_e p + (1-s_p)(1-p)}.$$

Similarly, it can shown that:

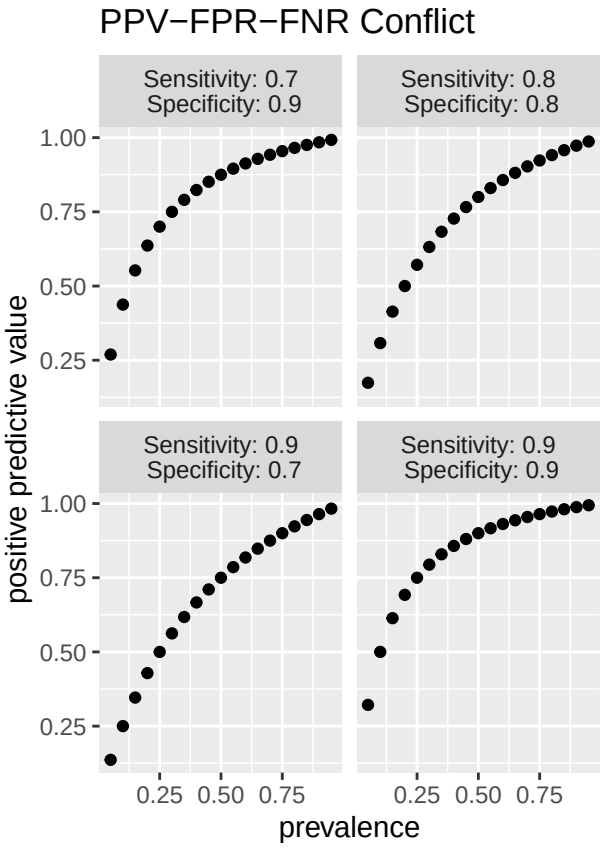
$$NPV = \frac{s_p(1-p)}{(1-s_e)p + s_p(1-p)}.$$

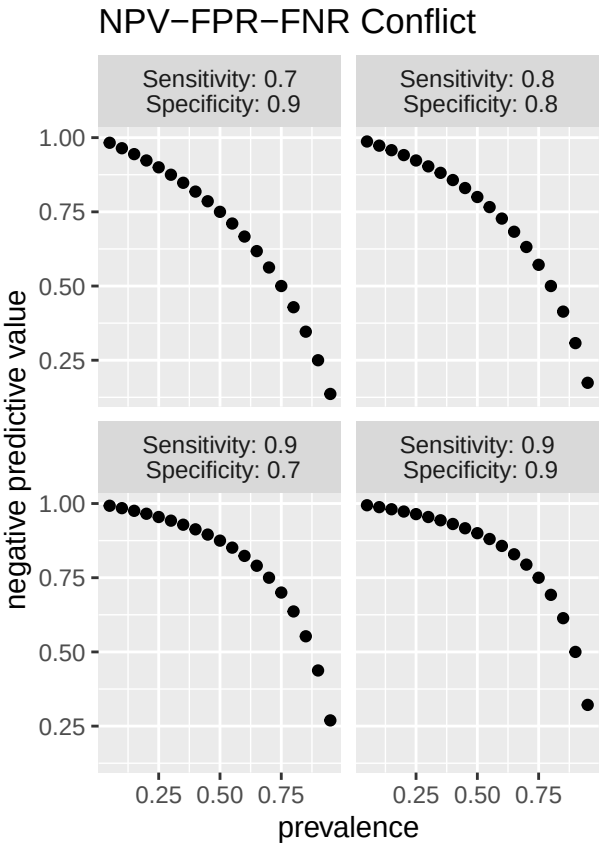
The code chunk below fixes arbitrary sensitivity (1 - FNR) and specificity (1 - FPR) values to illustrate through the proceeding plots that as prevalence varies, then PPV/ NPV varies and cannot be equal as long as sensitivity and specificity are held constant, hence a conflict.

```
library(dplyr)
library(ggplot2)
library(gridExtra)
```

```
ppv <- function(p, sens, spec){
  ppv <- (sens*p)/((sens*p) + ((1-spec)*(1-p)))
  return(ppv)
}

npv <- function(p, sens, spec){
  npv <- (spec*(1-p))/(((1-sens)*p) + (spec*(1-p)))
  return(npv)
}

dat_8080 <- data.frame(prevalence = seq(0.05,0.95,0.05)
                       , sens=0.80
                       , spec=0.80
                       , ppv = ppv(p=seq(0.05,0.95,0.05),
                                   sens=0.80,
                                   spec=0.80)
                       , npv = npv(p=seq(0.05,0.95,0.05),
                                   sens=0.80,
                                   spec=0.80))

dat_9090 <- data.frame(prevalence = seq(0.05,0.95,0.05)
                       , sens=0.90
                       , spec=0.90
                       , ppv = ppv(p=seq(0.05,0.95,0.05),
                                   sens=0.90,
                                   spec=0.90)
                       , npv = npv(p=seq(0.05,0.95,0.05),
                                   sens=0.90,
                                   spec=0.90))
```

```r
dat_9070 <- data.frame(prevalence = seq(0.05,0.95,0.05)
                       , sens=0.90
                       , spec=0.70
                       , ppv = ppv(p=seq(0.05,0.95,0.05),
                                   sens=0.90,
                                   spec=0.70)
                       , npv = npv(p=seq(0.05,0.95,0.05),
                                   sens=0.90,
                                   spec=0.70))

dat_7090 <- data.frame(prevalence = seq(0.05,0.95,0.05)
                       , sens=0.70
                       , spec=0.90
                       , ppv = ppv(p=seq(0.05,0.95,0.05),
                                   sens=0.70,
                                   spec=0.90)
                       , npv = npv(p=seq(0.05,0.95,0.05),
                                   sens=0.70,
                                   spec=0.90))

dat_all <- bind_rows(dat_8080, dat_7090, dat_9070, dat_9090) |>
  mutate(sens_spec = paste0("Sensitivity: ", sens,
                            "\n Specificity: ", spec)
         , fpr = 1 - spec
         , fnr = 1 - sens)

g1 <- ggplot(dat_all, aes(x=prevalence, y=ppv)) +
        geom_point() +
        labs(x="prevalence", y="positive predictive value",
             title = "PPV-FPR-FNR Conflict") +
        facet_wrap(~sens_spec)

g2 <- ggplot(dat_all, aes(x=prevalence, y=npv)) +
        geom_point() +
        labs(x="prevalence", y="negative predictive value",
             title = "NPV-FPR-FNR Conflict") +
        facet_wrap(~sens_spec)

grid.arrange(g1,g2, nrow=1, ncol=2)
```


PPV−FPR−FNR Conflict
Sensitivity: 0.7
Specificity: 0.9
Sensitivity: 0.8
Specificity: 0.8
Sensitivity: 0.9
Specificity: 0.7
Sensitivity: 0.9
Specificity: 0.9
positive predictive value
1.00
0.75
0.50
0.25
0.25 0.50 0.75
prevalence


NPV−FPR−FNR Conflict
Sensitivity: 0.7
Specificity: 0.9
Sensitivity: 0.8
Specificity: 0.8
Sensitivity: 0.9
Specificity: 0.7
Sensitivity: 0.9
Specificity: 0.9
negative predictive value
1.00
0.75
0.50
0.25
0.25 0.50 0.75
prevalence

Appendix B QSLR Python Code

The `reticulate` package is useful for setting up the correct Python environment within RStudio.

```
library(reticulate)
use_python("/cm/shared/apps/amh-Rstudio/python-3.11.4/bin/python3",
           required = TRUE)
#py_config()
#conda_list()
```

Python packages not already pre-installed need to be imported into the `reticulate` package first before being imported into the Python environment, as illustrated using `sklearn` below.

```
sklearn <- import("sklearn")
```

```
import sklearn
```

`math` provides access to the standard mathematical functions. `numpy` supports large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. `sys` provides functions and variables used to manipulate different parts of the Python run-time environment. `sklearn` features various classification, regression and clustering algorithms. `scipy.stats` contains a large number of probability distributions, summary and frequency statistics,

correlation functions and statistical tests, masked statistics, kernel density estimation, quasi-Monte Carlo functionality, and more. `scipy.optimize` provides functions for minimizing (or maximizing) objective functions, possibly subject to constraints. It includes solvers for nonlinear problems (with support for both local and global optimization algorithms), linear programing, constrained and nonlinear least-squares, root finding, and curve fitting.

```
import math
import numpy as np
import sys
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from scipy.stats import t
from scipy.optimize import minimize
```

```
# display 5 decimal places for all results
np.set_printoptions(precision=5, suppress=True)
```

The `tinv` function returns the inverse of `Student's t` CDF using the degrees of freedom in `nu` for the corresponding probabilities in `p`.

```
def tinv(p, nu):
    return t.ppf(p, nu)
```

The `stddev` function computes the sample standard deviation of the vector v, with Bessel's correction. In statistics, Bessel's correction is the use of $n - 1$ instead of n in the formula for the sample variance and sample standard deviation, where n is the number of observations in a sample. This method corrects the bias in the estimation of the population variance.

```
def stddev(v):
    n = v.size
    variance = (np.var(v) * n) / (n-1)
    return np.sqrt(variance)
```

The `ttestUpperBound` function computes a $(1-\delta)$-confidence upper bound on the expected value of a random variable using Student's t-test. It analyzes the data in v, which holds i.i.d. samples of the random variable. The upper confidence bound is given by sampleMean $+ \frac{\text{sampleStandardDeviation}}{\sqrt{(n)}} * tinv(1-\delta, n-1)$, where n is the number of observations in v.

```
def ttestUpperBound(v, delta):
    n  = v.size
    res = v.mean() + stddev(v) / math.sqrt(n) * tinv(1.0 - delta, n - 1)
    return res
```

The `predictTTestUpperBound` function works similarly to `ttestUpperBound`, but returns a more conservative upper bound. This function uses data in the vector v to compute all relevant statistics (mean and standard deviation) but assumes that the number of points being analyzed is k instead of $|v|$.

This function is used to estimate what the output of `ttestUpperBound` would be if it were to be run on a new vector, v, containing values sampled from the same distribution as the points in v. The 2.0 factor in the calculation is used to double the width of the confidence interval when predicting the outcome of the safety test in order to make the algorithm less confident/ more conservative.

```
def predictTTestUpperBound(v, delta, k):

    res = v.mean() + 2.0 * stddev(v) / math.sqrt(k) * tinv(1.0 - delta, k - 1)
    return res
```

The function `main()` below is set up to run a simple experiment.

```
def main():
    np.random.seed(123)
    numPoints = 5000

    (X,Y)  = generateData(numPoints)

    gHats  = [gHat1, gHat2]
    deltas = [0.1, 0.1]

    (result, found) = QSA(X, Y, gHats, deltas)

    if found:
        print("A solution was found: [%.10f, %.10f]" % (result[0], result[1]))
        print("fHat of solution (computed over all data, D):",
        fHat(result, X, Y))
    else:
        print("No solution found")
```

The `generateData` function samples data as described in the problem description.

```
def generateData(numPoints):
    X =     np.random.normal(0.0, 1.0, numPoints)
    Y = X + np.random.normal(0.0, 1.0, numPoints)
    return (X,Y)
```

The `predict` function takes in a solution θ and an input X, and produces as output the prediction of Y. In other words, this function will implement $\hat{y}(X, \theta)$.

Recall $\hat{y}(X, \theta) = \theta_1 X + \theta_2$.

```
def predict(theta, x):
    return theta[0] + theta[1] * x
```

The `fHat` function specifies the primary objective: to minimize the sample mean squared error. since the attempt is to maximize $\hat{f}$, however, the negative sample mean squared error is returned, so that maximizing $\hat{f}$ corresponds to minimizing the mean squared error.

```
def fHat(theta, X, Y):
    n = X.size
    res = 0.0
    for i in range(n):
        prediction = predict(theta, X[i])
        res += (prediction - Y[i]) * (prediction - Y[i])
    res /= n
    return -res
```

The gHat1 and gHat2 functions set up the behavioral constraints.

```
def gHat1(theta, X, Y):
    n = X.size
    res = np.zeros(n)
    for i in range(n):
        prediction = predict(theta, X[i])
        res[i] = (prediction - Y[i]) * (prediction - Y[i])
    res = res - 2.0
    return res

def gHat2(theta, X, Y):
    n = X.size
    res = np.zeros(n)
    for i in range(n):
        prediction = predict(theta, X[i])
        res[i] = (prediction - Y[i]) * (prediction - Y[i])
    res = 1.25 - res
    return res
```

The leastSq function implements least squares linear regression, which will be used as a starting point in the search for a candidate solution.

```
def leastSq(X, Y):
    X = np.expand_dims(X, axis=1)
    Y = np.expand_dims(Y, axis=1)
    reg = LinearRegression().fit(X, Y)
    theta0 = reg.intercept_[0]
    theta1 = reg.coef_[0][0]
    return np.array([theta0, theta1])
```

The `QSA` function is the shell code that partitions the data set, gets a candidate solution, and runs the safety test.

```
def QSA(X, Y, gHats, deltas):

    candidateData_len = 0.40
    candidateData_X, safetyData_X, candidateData_Y, safetyData_Y =
    train_test_split(X, Y, test_size=1-candidateData_len, shuffle=False)

    candidateSolution = getCandidateSolution(candidateData_X, candidateData_Y,
    gHats, deltas, safetyData_X.size)

    passedSafety      = safetyTest(candidateSolution, safetyData_X,
    safetyData_Y, gHats, deltas)

    return [candidateSolution, passedSafety]
```

The `safetyTest` function uses the previously defined functions to implement the safety test.

```
def safetyTest(candidateSolution, safetyData_X, safetyData_Y, gHats, deltas):

    for i in range(len(gHats)):
        g         = gHats[i]
        delta     = deltas[i]


        g_samples = g(candidateSolution, safetyData_X, safetyData_Y)

        upperBound = ttestUpperBound(g_samples, delta)

        if upperBound > 0.0:
            return False

    return True
```

Finally, the `candidateObjective` and `getCandidateSolution` functions use a black-box optimization algorithm to search for a candidate solution. The black box algorithm

used to search for a candidate solution is called Powell, which is an algorithm designed for finding a local minimum of a function using a bi-directional linear search. Powell, however, is not a constrained algorithm. One way of addressing this limitation is by incorporating the constraint into the objective function as a barrier function. In constrained optimization, a field of mathematics, barrier functions are used to replace inequality constraints by a penalizing term in the objective function that is easier to handle. That is, an approximate solution to the following unconstrained problem:

$$\theta_c \in \underset{\theta \in \mathbb{R}^2}{argmax} \begin{cases} \hat{f}(\theta, D_1) \text{ if } \hat{\mu}(\hat{g}_i(\theta_c, D_1)) + 2\frac{\hat{\sigma}(\hat{g}_i(\theta_c, D_1))}{\sqrt{|D_2|}} t_{1-\delta_i, |D_2|-1} \leq 0 \forall i \in \{1, 2, ..., n\} \\ -100,000 - \sum_{i=1}^{n} max(0, \hat{\mu}(\hat{g}_i(\theta_c, D_1)) + 2\frac{\hat{\sigma}(\hat{g}_i(\theta_c, D_1))}{\sqrt{|D_2|}} t_{1-\delta_i, |D_2|-1})) \text{ otherwise} \end{cases}$$

In this case, solutions that are predicted not to pass the safety test will not be selected by the optimization algorithm because a large negative performance is assigned to them. This barrier functions encourages Powell to tend towards solutions that will pass the safety test.

```
def candidateObjective(thetaToEvaluate, candidateData_X, candidateData_Y, gHats,
deltas, safetyDataSize):

    result = fHat(thetaToEvaluate, candidateData_X, candidateData_Y)

    predictSafetyTest = True

    for i in range(len(gHats)):
        g         = gHats[i]
        delta     = deltas[i]

        g_samples = g(thetaToEvaluate, candidateData_X, candidateData_Y)

        upperBound = predictTTestUpperBound(g_samples, delta, safetyDataSize)

        if upperBound > 0.0:
```

```
            if predictSafetyTest:
                predictSafetyTest = False

                result = -100000.0

            result = result - upperBound

    return -result
```

```
def getCandidateSolution(candidateData_X, candidateData_Y, gHats, deltas,
safetyDataSize):

    minimizer_method = 'Powell'
    minimizer_options={'disp': False}

    initialSolution = leastSq(candidateData_X, candidateData_Y)

    res = minimize(candidateObjective, x0=initialSolution,
    method=minimizer_method, options=minimizer_options,
    args=(candidateData_X, candidateData_Y, gHats, deltas, safetyDataSize))

    return res.x
```

Calling `main()` returns either a solution or NSF.

```
main()
```

The following code chunks utilize the above functions to perform the experimentation in Chapter 2.3.1. `timeit` allows timing of the execution of experiments. `numba` allows the use of a Just-in-Time (JIT) compiler to accelerate Python code.

```
#import necessary packages
import timeit
from numba import jit
```

```
#path where experiment results are saved
bin_path =
'/home/dasienga24/Statistics-Senior-Honors-Thesis/Thesis/index/'
'experiment_results/chapter_2/'
```

```
def run_experiments(worker_id, nWorkers, ms, numM, numTrials, mTest):

    # Results of the Seldonian algorithm runs
    ## The following code initializes an array filled with 0's.
    ## The resulting array will have numTrials rows (each trial)
    ## and numM columns (each data set size).
    ## Default is 0=False.

    seldonian_solutions_found = np.zeros((numTrials, numM))
    seldonian_failures_g1     = np.zeros((numTrials, numM))
    seldonian_failures_g2     = np.zeros((numTrials, numM))
    seldonian_fs              = np.zeros((numTrials, numM))

    # Results of the Least-Squares (LS) linear regression runs
    LS_solutions_found = np.ones((numTrials, numM))
    LS_failures_g1     = np.zeros((numTrials, numM))
    LS_failures_g2     = np.zeros((numTrials, numM))
    LS_fs              = np.zeros((numTrials, numM))


    # Prepares file where experiment results will be saved
    experiment_number = worker_id
    outputFile = bin_path + 'results%d.npz' % experiment_number


    # Generate the data used to evaluate the primary objective and failure rates
    np.random.seed( (experiment_number+1) * 9999 )
    (testX, testY) = generateData(mTest)


    for trial in range(numTrials): #numTrials trials for each value of m
        for (mIndex, m) in enumerate(ms):

            # Generate the training data, D
            base_seed         = (experiment_number * numTrials)+1
            np.random.seed(base_seed+trial)
```

```
            (trainX, trainY)  = generateData(m)

            # Run the Quasi-Seldonian algorithm
            (result, passedSafetyTest) = QSA(trainX, trainY, gHats, deltas)

            if passedSafetyTest:
                seldonian_solutions_found[trial, mIndex] = 1
                trueMSE = -fHat(result, testX, testY)
                seldonian_failures_g1[trial, mIndex] = 1 if trueMSE > 2.0  else 0
                seldonian_failures_g2[trial, mIndex] = 1 if trueMSE < 1.25 else 0
                seldonian_fs[trial, mIndex] = -trueMSE

            else:
                seldonian_solutions_found[trial, mIndex] = 0
                seldonian_failures_g1[trial, mIndex]     = 0
                seldonian_failures_g2[trial, mIndex]     = 0
                seldonian_fs[trial, mIndex]              = None

            # Run the Least Squares algorithm
            theta = leastSq(trainX, trainY)
            trueMSE = -fHat(theta, testX, testY)
            LS_failures_g1[trial, mIndex] = 1 if trueMSE > 2.0  else 0
            LS_failures_g2[trial, mIndex] = 1 if trueMSE < 1.25 else 0
            LS_fs[trial, mIndex] = -trueMSE



    # Save the arrays in a compressed format
    np.savez(outputFile,
             ms=ms,
             seldonian_solutions_found=seldonian_solutions_found,
             seldonian_fs=seldonian_fs,
             seldonian_failures_g1=seldonian_failures_g1,
             seldonian_failures_g2=seldonian_failures_g2,
             LS_solutions_found=LS_solutions_found,
             LS_fs=LS_fs,
             LS_failures_g1=LS_failures_g1,
             LS_failures_g2=LS_failures_g2)
```

```
# Create the behavioral constraints
gHats  = [gHat1, gHat2]
deltas = [0.1, 0.1]

# Initialize one worker because we're not using parallelization
nWorkers = 1

# sample sizes
ms   = [2**i for i in range(5, 17)]
numM = len(ms)

# The number of trials
numTrials = 100

mTest = ms[-1] * 100 # about 5,000,000 test samples

# Run experiments sequentially without parallelization
tic = timeit.default_timer()
for worker_id in range(1, nWorkers + 1):
    run_experiments(worker_id, nWorkers, ms, numM, numTrials, mTest)
toc = timeit.default_timer()
time_sequential = toc - tic # Elapsed time in seconds
```

Finally, the following code chunks compile the results from the experiments and presents them visually. `csv` implements classes to read and write tabular data in CSV format. `glob` finds all the path names matching a specified pattern according to the rules used by the Unix shell. This will be useful for referencing file paths and names. `re` provides regular expression matching operations similar to those found in Perl.

```
#import necessary packages
import csv
import glob
import re
import matplotlib.pyplot as plt
```

```
#specify path names
bin_path =
'/home/dasienga24/Statistics-Senior-Honors-Thesis/Thesis/index/experiment_results/'
'chapter_2/'
csv_path =
'/home/dasienga24/Statistics-Senior-Honors-Thesis/Thesis/index/experiment_results/'
'chapter_2/csv/'
```

```
#parse through files to obtain the experiment numbers
def get_existing_experiment_numbers():
    result_files       = glob.glob(bin_path + 'results*.npz')
    experiment_numbers = [re.search('.*results([0-9]*).*',
    fn, re.IGNORECASE) for fn in result_files]
    experiment_numbers = [int(i.group(1)) for i in experiment_numbers]
    experiment_numbers.sort()
    return experiment_numbers
```

```
#generate the file names for the results
def genFilename(n):
    return bin_path + 'results%d.npz' % n
```

For the `addMoreResults` function, recall that:

- `ms`: data set size
- `seldonian_solutions_found`: stores whether a solution was found (1=True,0=False)
- `seldonian_fs`: stores the primary objective values (fHat) if a solution was found
- `seldonian_failures_g1`: stores whether Seldonian solution was unsafe, (1=True,0=False), for the 1st constraint, g_1
- `seldonian_failures_g2`: stores whether Seldonian solution was unsafe, (1=True,0=False), for the 2nd constraint, g_2
- `LS_solutions_found`: stores whether a solution was found. These will all be true (=1)

- `LS_fs`: stores the primary objective values (f)
- `LS_failures_g1`: stores whether LS solution was unsafe, (1=True,0=False), for the 1st constraint, g_1
- `LS_failures_g2`: stores whether LS solution was unsafe, (1=True,0=False), for the 2nd constraint, g_2

```
def addMoreResults(newFileId, ms, seldonian_solutions_found, seldonian_fs,
seldonian_failures_g1, seldonian_failures_g2, LS_solutions_found, LS_fs,
LS_failures_g1, LS_failures_g2):

    newFile = np.load(genFilename(newFileId))
    new_ms                          = newFile['ms']
    new_seldonian_solutions_found = newFile['seldonian_solutions_found']
    new_seldonian_fs                = newFile['seldonian_fs']
    new_seldonian_failures_g1     = newFile['seldonian_failures_g1']
    new_seldonian_failures_g2     = newFile['seldonian_failures_g2']
    new_LS_solutions_found        = newFile['LS_solutions_found']
    new_LS_fs                       = newFile['LS_fs']
    new_LS_failures_g1              = newFile['LS_failures_g1']
    new_LS_failures_g2              = newFile['LS_failures_g2']

    if type(ms)==type(None):
        return [new_ms, new_seldonian_solutions_found, new_seldonian_fs,
      new_seldonian_failures_g1, new_seldonian_failures_g2,
      new_LS_solutions_found, new_LS_fs, new_LS_failures_g1,
      new_LS_failures_g2]
    else:
        seldonian_solutions_found  =
        np.vstack([seldonian_solutions_found, new_seldonian_solutions_found])
        seldonian_fs                =
        np.vstack([seldonian_fs,              new_seldonian_fs])
        seldonian_failures_g1      =
        np.vstack([seldonian_failures_g1,     new_seldonian_failures_g1])
        seldonian_failures_g2      =
        np.vstack([seldonian_failures_g2,     new_seldonian_failures_g2])
        LS_solutions_found         =
        np.vstack([LS_solutions_found,        new_LS_solutions_found])
        LS_fs                       =
        np.vstack([LS_fs,                     new_LS_fs])
```

```
        LS_failures_g1              =
        np.vstack([LS_failures_g1,            new_LS_failures_g1])
        LS_failures_g2              =
        np.vstack([LS_failures_g2,            new_LS_failures_g2])

        return [ms, seldonian_solutions_found, seldonian_fs, seldonian_failures_g1,
      seldonian_failures_g2, LS_solutions_found, LS_fs, LS_failures_g1, LS_failures_g2]
```

```
def stderror(v):
    non_nan = np.count_nonzero(~np.isnan(v))
    return np.nanstd(v, ddof=1) / np.sqrt(non_nan)
```

The output CSV file will have columns corresponding to:

1. `m` – the size of the data set
2. QSA mean value
3. QSA standard error bar size
4. LS mean value
5. LS standard error bar size

There will be one column per value of `m` (amount of training data).

```
def saveToCSV(ms, resultsQSA, resultsLS, filename):
    nCols = resultsQSA.shape[1]

    with open(filename, mode='w') as file:
        writer = csv.writer(file, delimiter=',')

        for col in range(nCols):

            cur_m          = ms[col]
            seldonian_data = resultsQSA[:,col]
            LS_data        = resultsLS[:,col]

            non_nan = np.count_nonzero(~np.isnan(seldonian_data))
            if non_nan > 0:
                seldonian_mean     = np.nanmean(seldonian_data)
```

```
                seldonian_stderror = stderror(seldonian_data)
            else:
                seldonian_mean     = 'NaN'
                seldonian_stderror = 'NaN'

            LS_mean     = np.mean(LS_data)
            LS_stderror = stderror(LS_data)

            writer.writerow([cur_m, seldonian_mean, seldonian_stderror,
            LS_mean, LS_stderror])
```

```
#gather the results and compile into CSV file
def gather_results():
    ms                          = None
    seldonian_solutions_found = None
    seldonian_fs                = None
    seldonian_failures_g1     = None
    seldonian_failures_g2     = None
    LS_solutions_found          = None
    LS_fs                       = None
    LS_failures_g1              = None
    LS_failures_g2              = None

    experiment_numbers = get_existing_experiment_numbers()

    for file_idx in experiment_numbers:
        res = addMoreResults(file_idx,
            ms,
            seldonian_solutions_found,
            seldonian_fs, seldonian_failures_g1, seldonian_failures_g2,
            LS_solutions_found, LS_fs, LS_failures_g1, LS_failures_g2)

        [ms,
        seldonian_solutions_found, seldonian_fs, seldonian_failures_g1,
        seldonian_failures_g2, LS_solutions_found, LS_fs, LS_failures_g1,
        LS_failures_g2] = res

    saveToCSV(ms,
    -1*seldonian_fs,
    -1*LS_fs,
    csv_path+'fs.csv') # here, negative to return MSE rather than negative MSE
```

```
    saveToCSV(ms,
    seldonian_solutions_found,
    LS_solutions_found,
    csv_path+'solutions_found.csv')

    saveToCSV(ms,
    seldonian_failures_g1,
    LS_failures_g1,
    csv_path+'failures_g1.csv')

    saveToCSV(ms,
    seldonian_failures_g2,
    LS_failures_g2,
    csv_path+'failures_g2.csv')
```

```
csv_path =
'/home/dasienga24/Statistics-Senior-Honors-Thesis/Thesis/index/'
'experiment_results/chapter_2/csv/'
img_path =
'/home/dasienga24/Statistics-Senior-Honors-Thesis/Thesis/index/'
'experiment_results/chapter_2/images/'
```

: Finally, the results are plotted as shown below.

```
def loadAndPlotResults(fileName, ylabel, output_file, is_yAxis_prob, legend_loc):
    file_ms, file_QSA, file_QSA_stderror, file_LS,
    file_LS_stderror = np.loadtxt(fileName, delimiter=',', unpack=True)

    fig = plt.figure()

    plt.xlim(min(file_ms), max(file_ms))
    plt.xlabel("Amount of data (m)", fontsize=12)
    plt.xscale('log')
    plt.xticks(fontsize=12)
    plt.ylabel(ylabel, fontsize=12)

    if is_yAxis_prob:
        plt.ylim(-0.1, 1.1)
    else:
        plt.ylim(-0.2, 2.2)
```

```
        plt.plot([1, 100000], [1.25, 1.25], ':k');
        plt.plot([1, 100000], [2.1,  2.1],  ':k');

    plt.plot(     file_ms,     file_QSA, 'b-', linewidth=3, label='QSA')
    plt.errorbar( file_ms,     file_QSA, yerr=file_QSA_stderror, fmt='.k');
    plt.plot(     file_ms,     file_LS,  'r-', linewidth=3, label='LS')
    plt.errorbar( file_ms,     file_LS,  yerr=file_LS_stderror, fmt='.k');
    plt.legend(loc=legend_loc, fontsize=12)
    plt.tight_layout()

    plt.savefig(output_file)
    plt.show(block=False)
```

```
gather_results()
```

```
loadAndPlotResults(csv_path+'fs.csv',
'Mean Squared Error',
img_path+'tutorial7MSE_py.png',
False,
'lower right')
```

```
loadAndPlotResults(csv_path+'solutions_found.csv',
'Probability of Solution',
img_path+'tutorial7PrSoln_py.png',
True,
'best')
```

```
loadAndPlotResults(csv_path+'failures_g1.csv',
r'Probability of $g_1(a(D))>0$',
img_path+'tutorial7PrFail1_py.png',
True,
'best')
```

```
loadAndPlotResults(csv_path+'failures_g2.csv',
r'Probability of $g_2(a(D))>0$',
img_path+'tutorial7PrFail2_py.png',
True,
'best')
```

References

Agarwal, A., Dudík, M., & Wu, Z. S. (2019). Fair regression: Quantitative definitions and reduction-based algorithms. In *International conference on machine learning* (pp. 120–129). PMLR.

Angwin, J., Larson, J., Mattu, S., & Kirchner, L. (2016). Machine bias risk assessments in criminal sentencing. *ProPublica, May*, *23*.

Asimov, I. (1994). *Forward the foundation* (Vol. 7). Spectra.

Boyd, S. P., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Castelnovo, A., Crupi, R., Greco, G., Regoli, D., Penco, I. G., & Cosentini, A. C. (2022). A clarification of the nuances in the fairness metrics landscape. *Scientific Reports*, *12*(1), 4209.

Chouldechova, A. (2017). Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big Data*, *5*(2), 153–163.

Chouldechova, A., & Roth, A. (2018). The frontiers of fairness in machine learning. *arXiv Preprint arXiv:1810.08810*.

Durahly, L. (2023). A gentle introduction to ML fairness metrics. Retrieved from https://superwise.ai/blog/gentle-introduction-ml-fairness-metrics/

Larson, J., Mattu, S., Kirchner, L., & Angwin, J. (2016). How we analyzed the COMPAS recidivism algorithm. *ProPublica, May, 23*.

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, *54*(6), 1–35.

Mohajon, J. (2021). Confusion matrix for your multi-class machine learning model. Retrieved from https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

Saeed, S., Alireza, B., Mohamed, E., & Ahmed, N. (2015). Evidence based emergency medicine part 2: Positive and negative predictive values of diagnostic tests.

Silva, B. C. da. (2019). UFRGS Entrance Exam and GPA Data (Version V2) [Data set]. Harvard Dataverse. http://doi.org/10.7910/DVN/O35FW8

Thomas, P. (2020). Testimony to the house committee on financial services task force on artificial intelligence hearing: "Equitable algorithms: Examining ways to reduce AI bias in financial services." Retrieved from https://www.congress.gov/116/meeting/house/110499/witnesses/HHRG-116-BA00-Wstate-ThomasP-20200212.pdf

Thomas, P. S. (n.d.). AI safety. https://aisafety.cs.umass.edu/index.html.

Thomas, P. S., Castro da Silva, B., Barto, A. G., Giguere, S., Brun, Y., & Brunskill, E. (2019a). Preventing undesirable behavior of intelligent machines. *Science*, *366*(6468), 999–1004.

Thomas, P. S., Castro da Silva, B., Barto, A. G., Giguere, S., Brun, Y., & Brunskill, E. (2019b). Supplementary materials for preventing undesirable behavior of intelligent

machines. *Science*, *366*(6468), 999–1004.