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

Abstract . . . . . **i**

Acknowledgments . . . . . **iii**

List of Tables . . . . . **vii**

List of Figures . . . . . **x**

Chapter 1: An Introduction to Fairness in Machine Learning **1**

1.1 Algorithmic Bias 2

1.2 Statistical Definitions of Fairness 5

1.2.1 Group Fairness in Regression Settings 6

1.2.2 Group Fairness in Classification Settings 7

1.3 Fairness Conflicts 13

1.3.1 Fairness Conflicts in Regression 14

1.3.2 Fairness Conflicts in Classification 15

1.3.3 On Fairness Conflicts 17

Chapter 2: Seldonian Algorithms **19**

2.1 The Standard Machine Learning Approach 19

2.1.1 Limitations of the Standard Approach 21

2.1.2 Potential Remedies 23

2.2 The Seldonian Framework 25

2.2.1 The Seldonian Optimization Problem 26

2.2.2 Quasi-Seldonian Algorithms 28
2.3 Toy Example: A Quasi-Seldonian Regression Algorithm 32
2.3.1 Experimentation 34
Chapter 3: Applying the Seldonian Framework in a Classification Setting 39
3.1 The COMPAS Data Set 40
3.2 Descriptive Statistics 41
3.2.1 Response Variable 42
3.2.2 Protected Attribute 42
3.2.3 Associations Between the Predictors and Response 42
3.2.4 Multivariate Analysis 44
3.3 Fairness and Demographic Analysis 45
3.3.1 COMPAS Tool Analysis 46
3.3.2 Proxy Relationships 50
3.4 Logistic Regression 53
3.4.1 Fitting the Logistic Regression Model 53
3.4.2 Evaluating the Equality of Odds 54
3.5 Seldonian Classification 56
3.5.1 Formulating the Seldonian Problem 57
3.5.2 Evaluating Performance and Fairness 58
Chapter 4: Seldonian Algorithms for Classification: A Simulation Study 61
4.1 Simulation Design 62
4.1.1 Aims 62
4.1.2 Data-Generation Mechanism 62
4.1.3 Target 64
4.1.4 Methods 64
4.1.5 Performance Measures 65

4.2 Simulation Results 66
4.2.1 Probability of a Seldonian Solution 66
4.2.2 Discrimination 69
4.2.3 Accuracy 70
4.2.4 The Trade-Off Between Discrimination and Accuracy 70
4.2.5 Non-Convergent Seldonian Models 70
4.3 Discussion 70
Appendix A: Sufficiency v Separation Fairness Conflict Equation . . 71
Appendix B: Quasi-Seldonian Linear Regression Theoretical and Python Implementation 77
Appendix C: SQL Script to Retrieve COMPAS Data 95
Appendix D: Logistic Regression on the COMPAS Data Set 99
Appendix E: Seldonian Classification on the COMPAS Data Set . . 101
Appendix F: Generating the Simulation Parent Data Set 109
References 111

List of Tables

3.1 The Number of Defendants who were Labeled Higher/Lower Risk by the COMPAS Tool Stratified by Recidivism Status among 9,387 Defandants in Broward County Florida, 2013-2014 48

3.2 The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by the COMPAS Tool Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014 . . . 49

3.3 COMPAS Logistic Regression Error Rates (Percentage) 56

3.4 The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by Logistic Regression Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014 . . . 56

3.5 The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by a Seldonian Algorithm ($\epsilon = 0.2$) Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014 59

3.6 The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by a Seldonian Algorithm ($\epsilon = 0.1$) Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014 59

3.7 The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by a Seldonian Algorithm ($\epsilon = 0.05$) Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014 59

3.8 The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by a Seldonian Algorithm ($\epsilon = 0.01$) Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014 59

4.1 Probability of Obtaining a Seldonian Solution 67

4.2 Satisfaction of the Behavioral Constraint by Seldonian Solutions that Passed the Safety Test 68

List of Figures

1.1 COMPAS Prediction Fails Differently for Black v White Defendants . 3

1.2 An Example of Demographic Parity 8

1.3 A Confusion Matrix 11

1.4 An Example of Equality of Odds 12

2.1 Least Squares Fit on Synthetic Data Drawn from Different Distributions 22

2.2 Overview of the Seldonian Framework 27

2.3 Synthetic Data for Quasi-Seldonian Linear Regression Tutorial 33

2.4 Quasi-Seldonian Linear Regression 34

2.5 QSA Experiment: Performance Loss 35

2.6 QSA Experiment: Probability of a Solution 35

2.7 QSA Experiment: Satisfaction of 1st Behavioral Constraint 36

2.8 QSA Experiment: Satisfaction of 2nd Behavioral Constraint 36

3.1 Distribution of Age (panel A) and Number of Prior Offenses (panel B) by Recidivism Status among 9,387 Defandants in Broward County Florida, 2013-2014 43

3.2 Conditional Distribution of Age (panel A) and Juvenile Offense (panel B) by Recidivism Status among 9,387 Defandants in Broward County Florida, 2013-2014 44

3.3 Spearman's Correlation Matrix for Age, Number of Prior Offenses, and the COMPAS Tool Decile Scores 45
3.4 Distribution of COMPAS Tool Decile Scores among 9,387 Defandants in Broward County Florida, 2013-2014 46
3.5 Distribution of COMPAS Tool Risk Scores among 9,387 Defandants in Broward County Florida, 2013-2014 47
3.6 Distribution of COMPAS Tool Decile Scores Stratified by Race among 9,387 Defandants in Broward County Florida, 2013-2014 49
3.7 Distribution of COMPAS Tool Proxy Variables Stratified by Race among 9,387 Defandants in Broward County Florida, 2013-2014 . . . 51
3.8 Distribution of Recidivism Prevalence Stratified by Race among 9,387 Defandants in Broward County Florida, 2013-2014 52
3.9 COMPAS Logistic Regression Receiver Operating Characteristic (ROC) Curve 54
4.1 Probability of Returning a Solution and Satisfying the Constraint by Sample Size 69

Chapter 1 An Introduction to Fairness in Machine Learning

In the current landscape, both the public and private sectors are increasingly relying on data-driven methods to automate, streamline, and 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 introduce and perpetuate discriminatory practices inadvertently 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, the resulting algorithms will learn 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, a fundamental question arises: are there any established statistical notions of fairness and bias? Is it possible to mathematically and statistically define algorithmic bias and unfairness, thereby opening avenues to address the challenges they pose? And if so, are there ways to leverage statistical tools to resolve such bias and unfairness? This thesis paper is dedicated to exploring and answering precisely these questions.

1.1 Algorithmic Bias

There are multiple different types and sources of bias in 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 the literature regards 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 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. This is because 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 and 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 performances 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. However, making distinctions based on race is generally not acceptable in society.

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

Understanding statistical notions of fairness at both group and individual levels is crucial. *Group notions* focus on a few demographic groups and assess the parity of statistical measures across all these groups (Chouldechova & Roth, 2018). It is important to note that while providing guarantees to 'average' numbers of protected groups, group measures do not necessarily ensure fairness to individuals or structured subgroups within protected demographic groups. These notions are the focus of this paper.

Individual notions, on the other hand, are assessed on specific pairs of individuals rather than averaged across groups (Chouldechova & Roth, 2018). The principle is that 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, further complicating its application (Chouldechova & Roth, 2018). This highlights the challenges and limitations of individual notions, for which a gap in literature exists.

Ultimately, group notions and individual notions are not in conflict per se. Instead, they exist 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 combine 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 the

majority 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 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

At the core, group notions of fairness in classification refer to treating different groups equally. They aim to remedy or prevent disparate impact, 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).

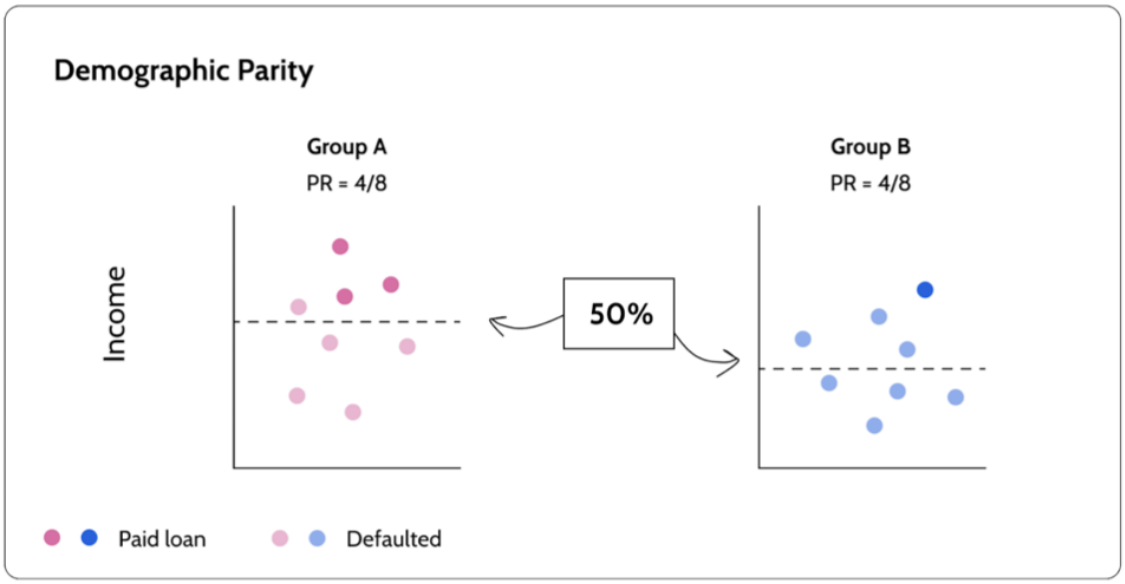


Figure 1.2: An Example of Demographic Parity (Durahly, 2023)

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).

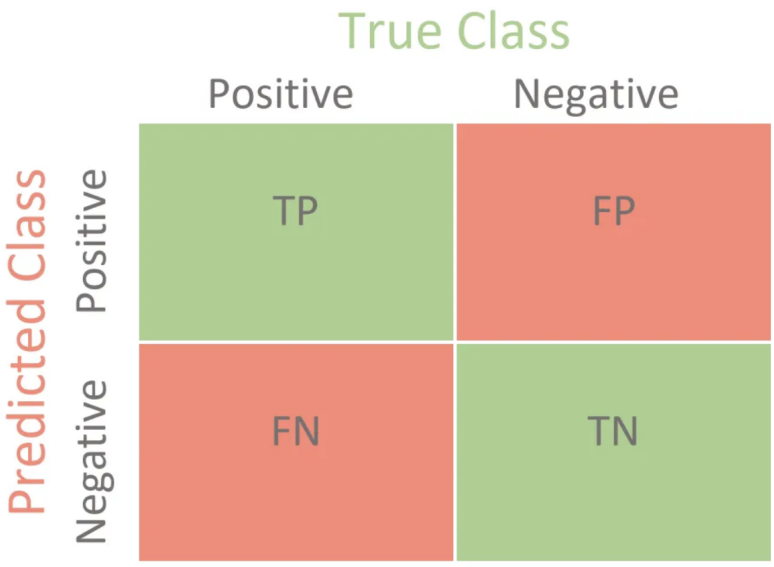


Figure 1.3: A Confusion Matrix (Mohajon, 2021)

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:

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

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 using a prediction tool that is perceived as free from predictive bias. Just because an algorithm satisfies a particular definition of fairness does not infer that the algorithm is *fair* in every sense of that word. Balancing overall error rates alone is insufficient 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.

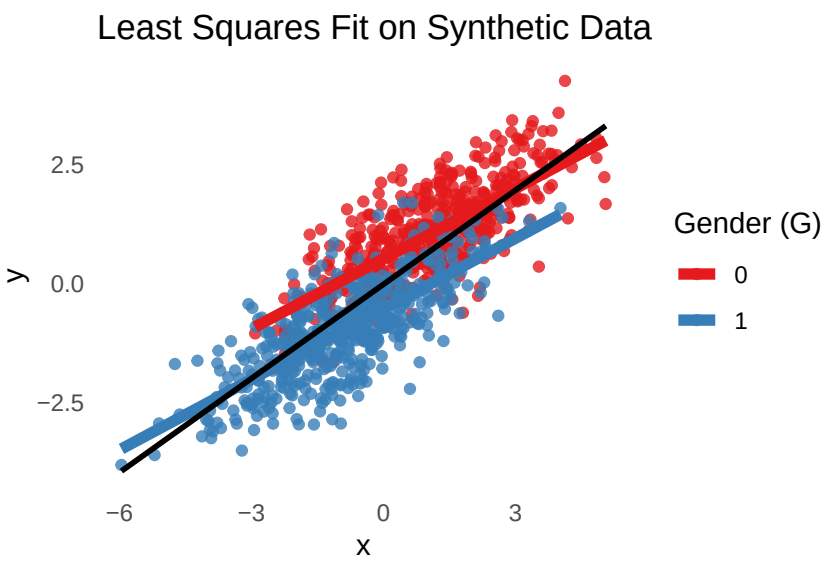


Figure 2.1: Least Squares Fit on Synthetic Data Drawn from Different Distributions

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.

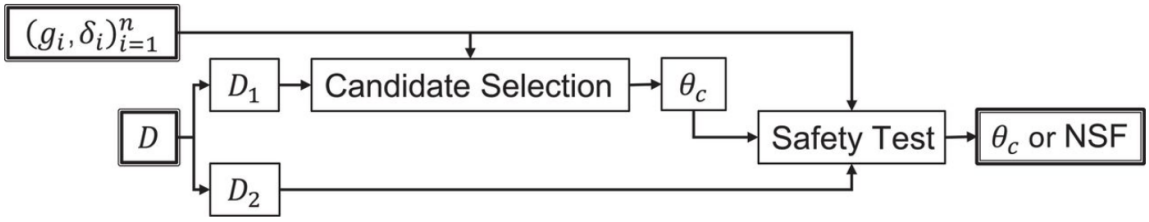


Figure 2.2: Overview of the Seldonian Framework (P. S. Thomas et al., 2019a)

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 without making any assumptions about the data. However, these values are often unknown and need to be estimated from the data provided. This may require making some assumptions about the underlying population distribution the sample was drawn from, hence the name *quasi*-Seldonian.

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 and assumptions of normality 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 observations. 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.

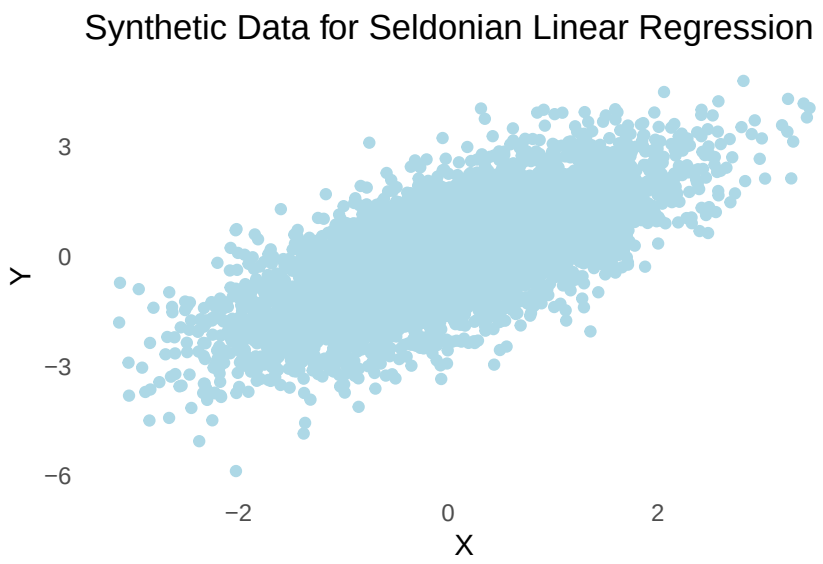


Figure 2.3: Synthetic Data for Quasi-Seldonian Linear Regression Tutorial

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
```

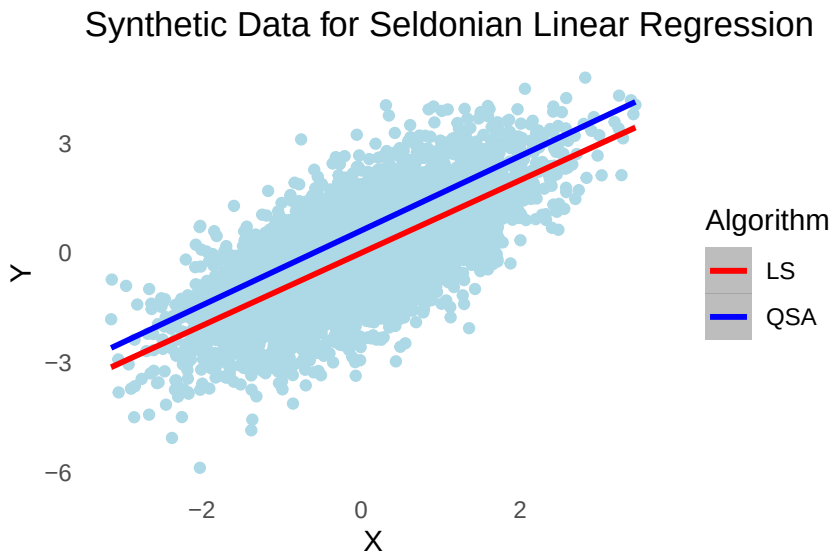


Figure 2.4: Quasi-Seldonian Linear Regression

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.

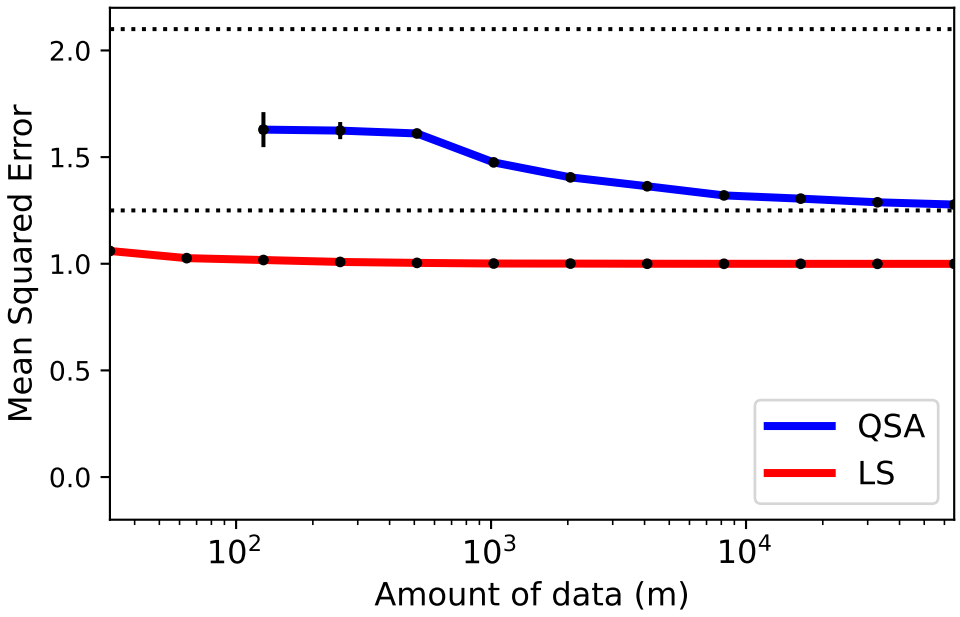


Figure 2.5: QSA Experiment: Performance Loss

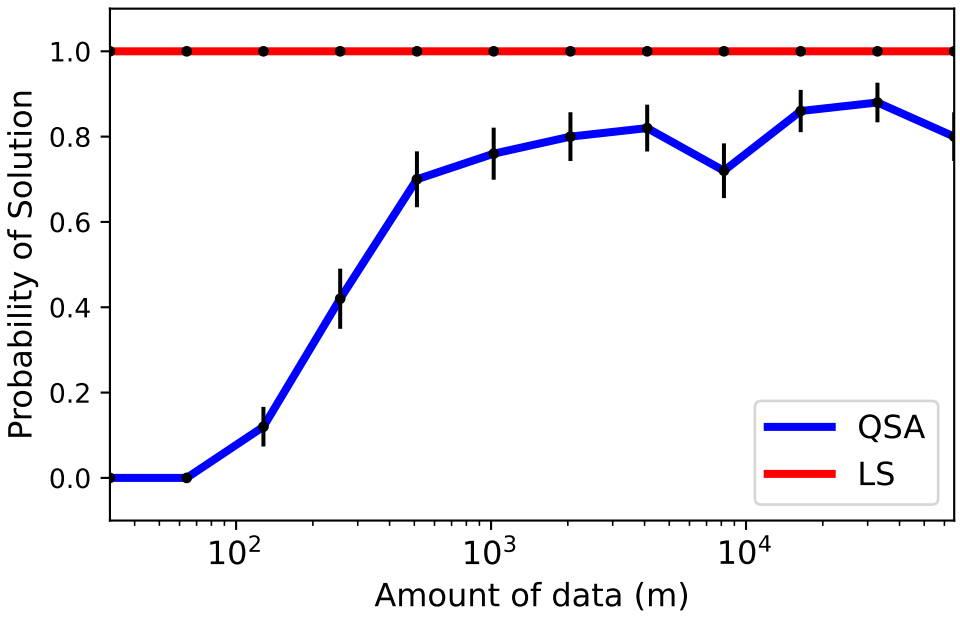


Figure 2.6: QSA Experiment: Probability of a Solution

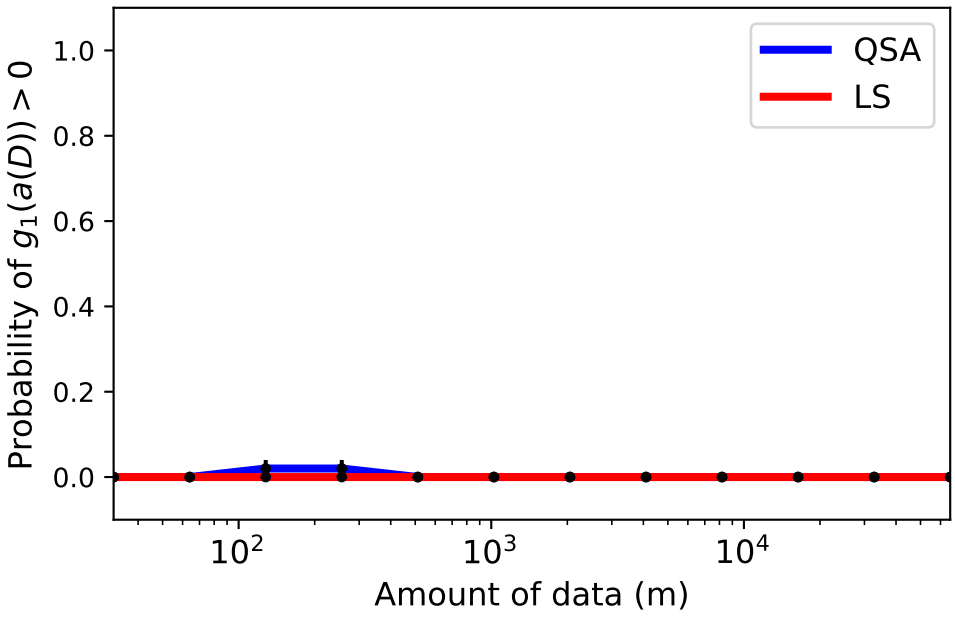


Figure 2.7: QSA Experiment: Satisfaction of 1st Behavioral Constraint

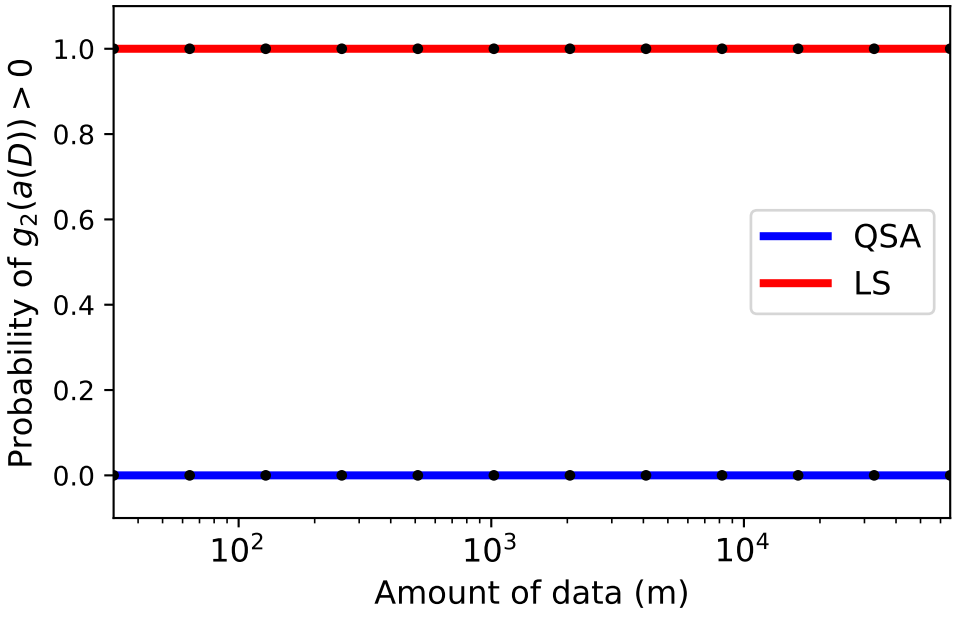


Figure 2.8: QSA Experiment: Satisfaction of 2nd Behavioral Constraint

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 always violated 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. Next, Chapter 3 delves into the properties of Seldonian algorithms within a classification setting.

Chapter 3 Applying the Seldonian Framework in a Classification Setting

Chapter 2 introduced the Seldonian framework, which offers probabilistic guarantees for satisfying defined behavioral constraints. Although impractical, the toy linear regression example demonstrated how Seldonian algorithms may be employed in a setting with a continuous real-valued response variable.

However, machine learning algorithms deal with classification problems in many practical applications. Applications range from risk assessment tools like COMPAS, discussed in Chapter 1, to credit scoring and employment prediction algorithms, to name a few. However, deploying such algorithms raises significant ethical concerns, as discussed in Chapters 1 and 2, primarily regarding fairness and the potential reinforcement of discriminatory practices. Given the historical and social biases inherent in the data used to train these algorithms, there is a pressing need to assess their fairness and mitigate any potential harm they may cause disadvantaged groups.

To offer a path forward in addressing this issue, this chapter aims to apply the Seldonian framework to the COMPAS data set and assess its predictive outcomes compared to the COMPAS tool and logistic regression, a standard ML procedure. Specifically, the objective is to assess whether Seldonian approaches can produce fairer outcomes within the COMPAS setting, drawing on some of the statistical definitions of fairness defined in Chapter 1.2. This chapter will present a framework that can be

emulated in other classification problems where fairness is a concern.

3.1 The COMPAS Data Set

The COMPAS data set, obtained from the ProPublica Data Store (Broward County Clerk's Office, Broward County Sherrif's Office, Florida Department of Corrections & ProPublica, 2024), records information on defendants from Broward County, Florida, that were evaluated for the risk of recidivism by the Correctional Offender Management Profiling for Alternative Sanctions (COMPAS) tool in 2013 and 2014. A SQL query of the COMPAS database retrieved 12,160 observations (Appendix C). Of 29 variables returned, 6 were identified to be useful for predictive analysis: age, age category, sex, marital status, whether the defendant had a history of juvenile offenses or not, and the total number of prior offenses committed by the defendant. The response variable records whether or not a defendant had recommitted a crime within two years. Finally, the protected attribute, race, has six levels: African-American, Caucasian, Hispanic, Asian, Native American, and Other.

Additionally, the COMPAS tool itself maps its raw scores into decile scores ranging from 1 to 10. The decile scores are directly associated with the risk of recidivism: scores between 1 and 4 are labeled as 'low' risk, between 5 and 7 as 'medium' risk, and between 8 and 10 as 'high risk'. After cleaning the data to address anomalies, improve the data quality, and remove duplicate observations, the data set size was reduced to 9387 observations. Section 3.2 examines these variables in more detail and their relationships with each other.

Before proceeding with the analysis, it is crucial to consider that while the recidivism status, as recorded by the response variable, is treated as an objective gold standard truth of whether a defendant reoffended or not for the rest of this thesis, it is, in

itself, likely biased by societal factors. For example, police officers are more likely to arrest Black defendants than White defendants for the same offense, and judges are more likely to convict and grant lengthier sentences to Black defendants than White defendants for the same charges (The Sentencing Project, 2018). While it is tangential to the theoretical underpinnings of the work in the proceeding sections, this context is, overall, important when applying mathematical, statistical, and computer science solutions to addressing algorithmic bias.

3.2 Descriptive Statistics

This section presents a holistic exploratory analysis of the COMPAS data set. The defendants' ages range from 18 to 96, with a median of 32 years and a mean of 34.8 years. There is a right skew, with the middle 50% of the defendants being between the ages of 25 and 42. Similarly, the number of prior offenses has a significant right skew with a median of 1 offense and a mean of 3 offenses. The middle 50% of the defendants have committed between 0 and 4 offenses, and the defendant with the most prior offenses has been convicted of committing 38 crimes.

Of the 9387 total defendants from Broward County, Florida, in 2013 and 2014, 79.4% were male. About 21.4% are < 25 years old, 57.2% are between 25 and 45 years old, and 21.4% are > 45 years old. The majority of defendants are single (76.7%), with the next largest categories being married (12.1%) and divorced (4.2%). The other levels were significant other, separated, unknown, and widowed, respectively. Finally, 87.9% of defendants had no record of juvenile offenses. The remaining 12.1% ranged from having 1 juvenile offense to 21 juvenile offenses on record, with a median of 1, a mean of 1.96, and the middle 50% recording 1 or 2 offenses.

3.2.1 Response Variable

The response variable records whether or not a defendant recommitted any crime within two years. About two-thirds (n = 6199) of the defendants did not recommit a crime within two years, while about one-third (n = 3188) did. The different proportion of observations in each level indicates class imbalance, which often affects the performance of machine learning classification algorithms. Given the class imbalance, analyzing performance relative to the classes will be important when assessing model performance in the proceeding sections.

3.2.2 Protected Attribute

Finally, the protected attribute in this analysis is race. Most of the defendants are African-American (49.8%) and Caucasian (34.6%), with only 8.7% Hispanic, 0.5% Asian, and finally, 0.3% Native American. The remaining 6.1% of defendants identify as 'Other'.

3.2.3 Associations Between the Predictors and Response

With a thorough understanding of the variables, this section examines the relationship between the predictive variables and the response variable, recidivism.

Defendants who recidivated tended to be younger than than those who didn't (median age 29 versus 33, respectively; Figure 3.1A and 3.2A). Defendants who recidivated also tended to have more non-juvenile prior offenses than those who didn't recidivate (median prior offenses 3 versus 1, respectively; Figure 3.1B).

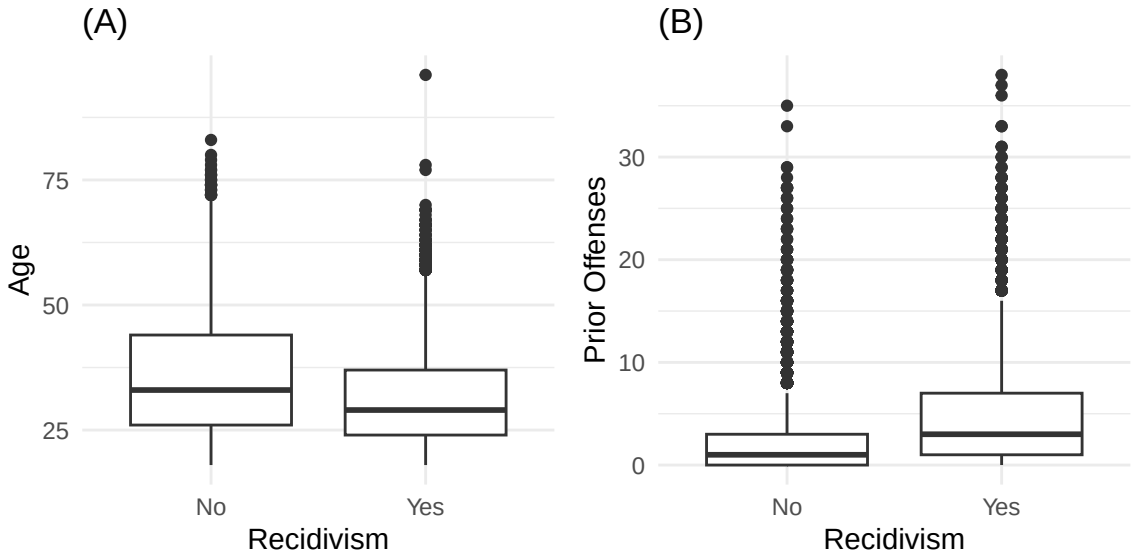


Figure 3.1: Distribution of Age (panel A) and Number of Prior Offenses (panel B) by Recidivism Status among 9,387 Defandants in Broward County Florida, 2013-2014

Sex and marital status were similar between defendants who did and did not recidivate. Defendants who recidivated were more likely to have had a juvenile offense, although the vast majority of defendants in both cases had no juvenile offenses (Figure 3.2B).

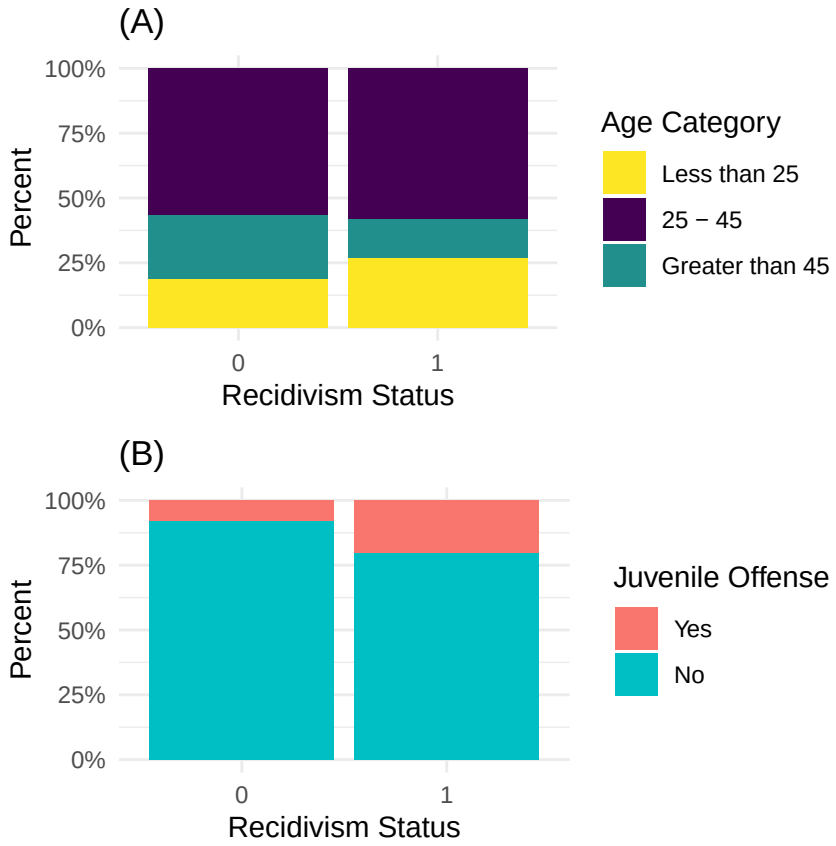


Figure 3.2: Conditional Distribution of Age (panel A) and Juvenile Offense (panel B) by Recidivism Status among 9,387 Defandants in Broward County Florida, 2013-2014

3.2.4 Multivariate Analysis

As detailed in Section 3.2.3, some predictive variables are associated with the response variable, recidivism. A scatterplot matrix examining the relationship between the continuous variables, age and prior offenses, revealed weak correlations (Spearman's rho: $\rho = 0.12$) and nonlinear relationships. There are no significant concerns for multicollinearity. Figure 3.3 illustrates this using Spearman's correlation. In addition, the correlation matrix incorporates the COMPAS tool decile scores to examine the predictive relationship between the continuous variables and the COMPAS tool results. Spearman's correlation revealed a moderate relationship of $\rho = -0.44$ and $\rho = 0.44$

for age and prior offenses, respectively.

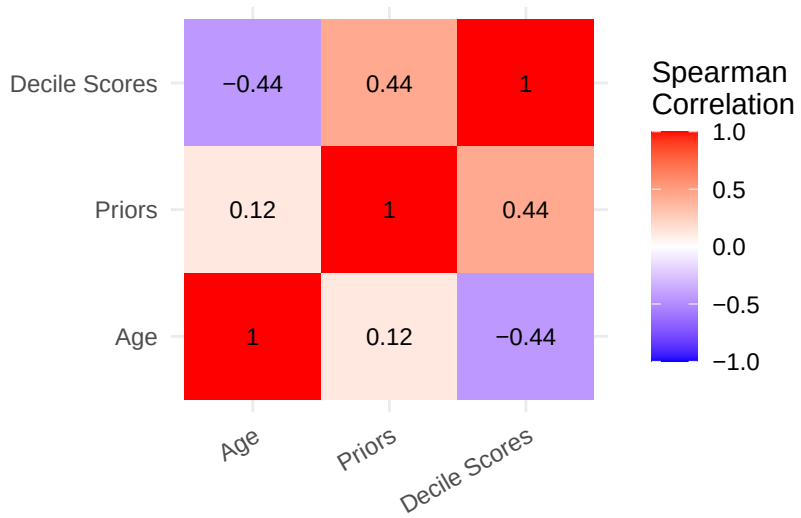


Figure 3.3: Spearman's Correlation Matrix for Age, Number of Prior Offenses, and the COMPAS Tool Decile Scores

With a thorough understanding of the variables, Section 3.3 analyzes the data set and the COMPAS tool decile scores with a focus on fairness and an aim to examine the fairness concerns discussed in Chapters 1 and 2.

3.3 Fairness and Demographic Analysis

Chapter 1.1 introduced algorithmic bias as a situation when an algorithm's decisions are skewed towards a particular group of people, either positively or negatively. With race as the protected attribute, this section aims to examine the COMPAS tool results, the underlying proxy relationships between the variables and the sensitive attribute, race, and create a table similar to the one in Figure 1.1, which highlights the discrepancy in false positive and false negative rates of the COMPAS tool for Black and White defendants.

3.3.1 COMPAS Tool Analysis

Recall that COMPAS decile scores of 1 to 4 are mapped as 'low' risk, 5 to 7 as 'medium' risk, and 8 to 10 as 'high' risk. The median decile score for the entire data set is four, and Figure 3.4 illustrates that the COMPAS tool classifies more than half of the defendants as low risk. In particular, the tool classifies 57.2% as low risk and 17.9% as high risk, with the remaining 24.9% as medium risk. These predictions align with expectations since most defendants did not recommit a crime within the two-year time window, as observed in Section 3.2.1.

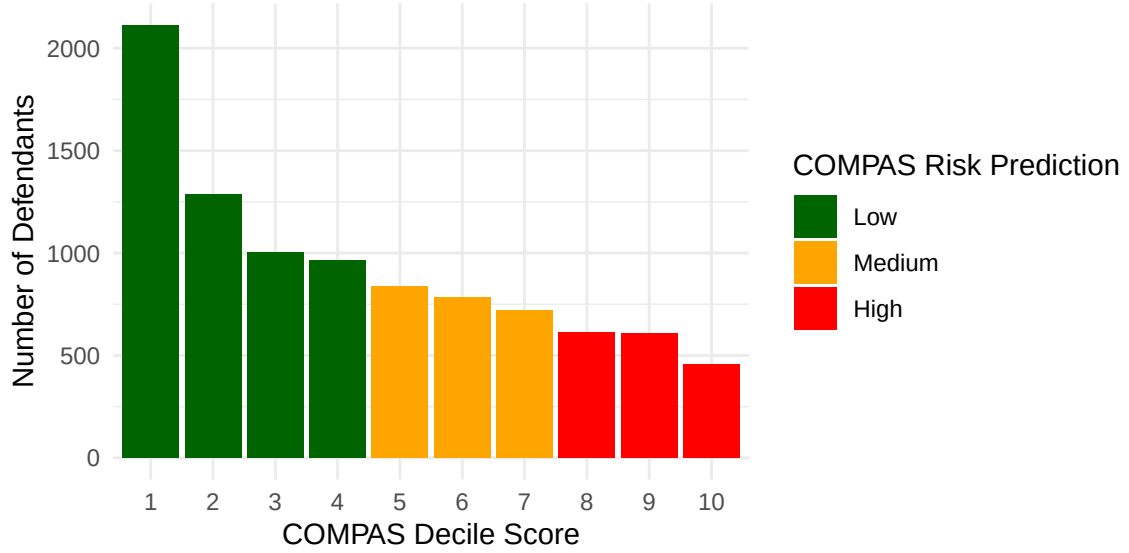


Figure 3.4: Distribution of COMPAS Tool Decile Scores among 9,387 Defandants in Broward County Florida, 2013-2014

However, when these scores are broken down by whether the defendant reoffended within two years, it is revealed that the COMPAS tool performs poorly in classifying participants who recidivate. Such participants are classified almost equally into the three risk categories: low, medium, and high. Furthermore, while most defendants who do not reoffend within two years are classified as low risk, a significant number of them are classified as 'medium' and 'high' risk. The decile scores of defendants who

recidivate and those who do not also range from 1 to 10, although the median decile score is 6 for the former group and 3 for the latter group. Generally, these results raise concerns about the COMPAS tool's predictive performance.

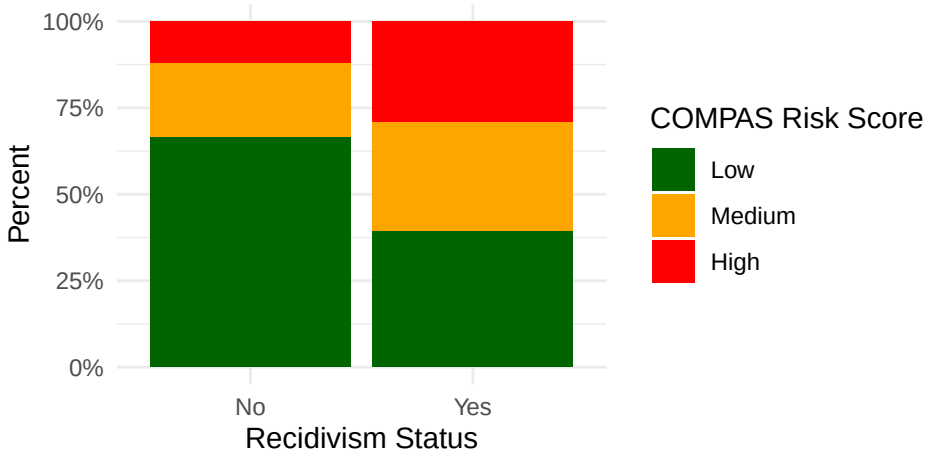


Figure 3.5: Distribution of COMPAS Tool Risk Scores among 9,387 Defandants in Broward County Florida, 2013-2014

A breakdown by race, the protected attribute, will allow for further analysis of the performance of the COMPAS tool. Before that, converting this into a binary prediction problem will make analysis much more tractable. Decile scores of 1 to 5 will be mapped as 'lower' risk, while those ranging from 6 to 10 will be mapped as 'higher' risk.

Table 3.1 displays the confusion matrix of the COMPAS tool's results on this data set, which has 9387 total observations, with this binary definition of risk. The matrix reveals that the model has an overall accuracy of 66.61% on this data set. However, 66.04% of defendants do not recommit a crime within two years, as observed in Section 3.2.1. Therefore, a blind non-informative model that classifies every defendant into the negative class would attain an accuracy of 66%, suggesting that the COMPAS tool's 66.61% accuracy is a negligible improvement.

Notice, furthermore, that the model struggles more in predicting whether

Table 3.1: The Number of Defendants who were Labeled Higher/Lower Risk by the COMPAS Tool Stratified by Recidivism Status among 9,387 Defandants in Broward County Florida, 2013-2014

Risk	Reoffended	Did Not Reoffend
Higher	1618	1564
Lower	1570	4635

defendants will reoffend than in predicting whether defendants will not reoffend. This imbalance in error rates is expected because of the class imbalance observed when performing exploratory data analysis in Section 3.2 – most defendants do not reoffend, so the model maximizes performance for those defendants. In terms of proportions, 49.25% of defendants who reoffended are incorrectly labeled as lower risk (almost equivalent to a flip of a coin), compared to 25.23% of defendants who did not reoffend but are incorrectly labeled as higher risk. This also raises the question of what type of prediction is more important: the risk of recidivism or the risk of non-recidivism. Is wrongly attributing a defendant as higher risk or wrongly attributing a defendant as lower risk worse for society?

While the confusion matrix in Table 3.1 relays information on what types of errors the COMPAS tool tends to make and raises questions about the implications of that, Table 3.2 breaks that down further by race for a more granular assessment. While the overall FPR is 25.3%, it is much higher for Black defendants (37.71%) and much lower for White defendants (16.34%). Similarly, while the overall FNR is 49.25%, it is much higher for White defendants (62.26%) and much lower for Black defendants (39.01%) (Table 3.2). The tool is more than twice as likely to classify a Black defendant who did not reoffend as higher risk compared to a White defendant. It makes the opposite mistake, where it is 1.6 times more likely to classify a White defendant who

Table 3.2: The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by the COMPAS Tool Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014

Recidivism Status	Predicted Risk	Black	White	Hispanic	Asian	Native American
Did Not Reoffend	Higher	37.7	16.3	14.3	7.89	21.1
Reoffended	Lower	39.0	62.3	68.3	50.00	25.0

reoffended as lower risk compared to a Black defendant. This discrepancy aligns with ProPublica's findings in Figure 1.1 and is alarming given that race was not included in the model.

Including the other races reveals that Asian defendants who did not reoffend were the least likely to be labeled as higher risk – Black defendants were the most likely. Conversely, White defendants who reoffended were the most likely to be labeled as lower risk – Native Americans were the least likely. However, it is essential to consider the few observations in both the Asian (n = 48) and Native American groups (n = 27), as detailed in Section 3.2.2.

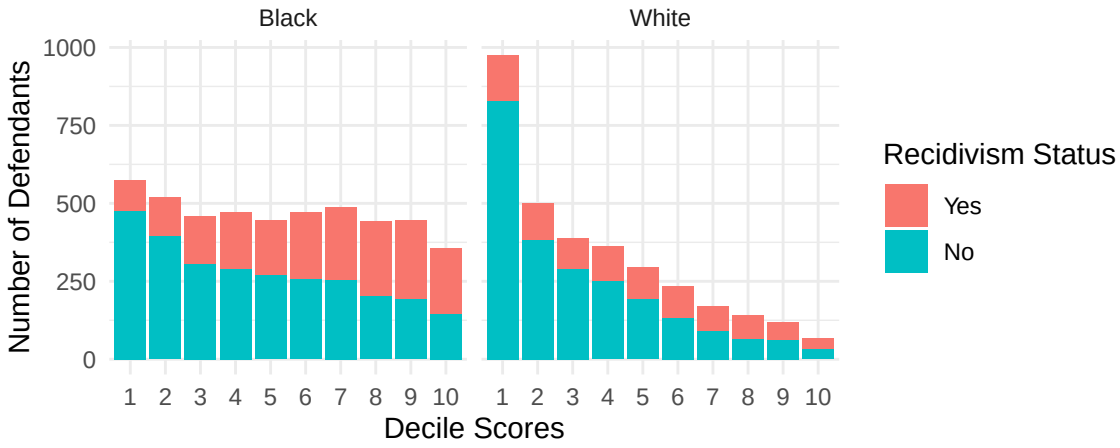


Figure 3.6: Distribution of COMPAS Tool Decile Scores Stratified by Race among 9,387 Defandants in Broward County Florida, 2013-2014

The analysis in this section provides evidence for disparities with favorable COM-

PAS outcomes for White and Asian defendants and unfavorable outcomes for Black and Native American defendants. To visualize these results, Figure 3.6 displays the distribution of the decile scores for the two most populous races in the data set. While there is a similar distribution of recidivism for all races with most defendants not recidivating, Black defendants' decile scores are distributed almost uniformly among the ten decile scores. In contrast, the White defendants' decile scores have a significant right skew, with most observations in lower decile scores. This further emphasizes the racial disparity in employing these risk scores in judicial decisions and the present algorithmic bias.

3.3.2 Proxy Relationships

Section 3.3.1 revealed that the predictions of the COMPAS tool exhibit significant racial discrepancies. Yet, the COMPAS tool is a race-blind model. Race was not included as a variable, so how could a model result in such racially distinct outcomes? To answer this question, this section analyzes the relationship between race and the predictive variables to examine how much information regarding a defendant's race is incorporated into the model via other variables.

There are more male than female defendants for all races, and most defendants are single. However, African-American defendants tend to be, on average, the youngest compared to all the other races. Asian defendants, followed by Caucasian defendants, tend to be the oldest. Nevertheless, there is considerable overlap among all the races, and Figure 3.7 visualizes the relationships. In looking at the age categories by race, more African-American defendants are < 25 than those who are > 45. The converse is true for Caucasians, with more defendants that are > 45 in comparison to those < 25. These distributional differences illustrate that there is some relationship between age and race in this data set, particularly for Black versus White defendants.

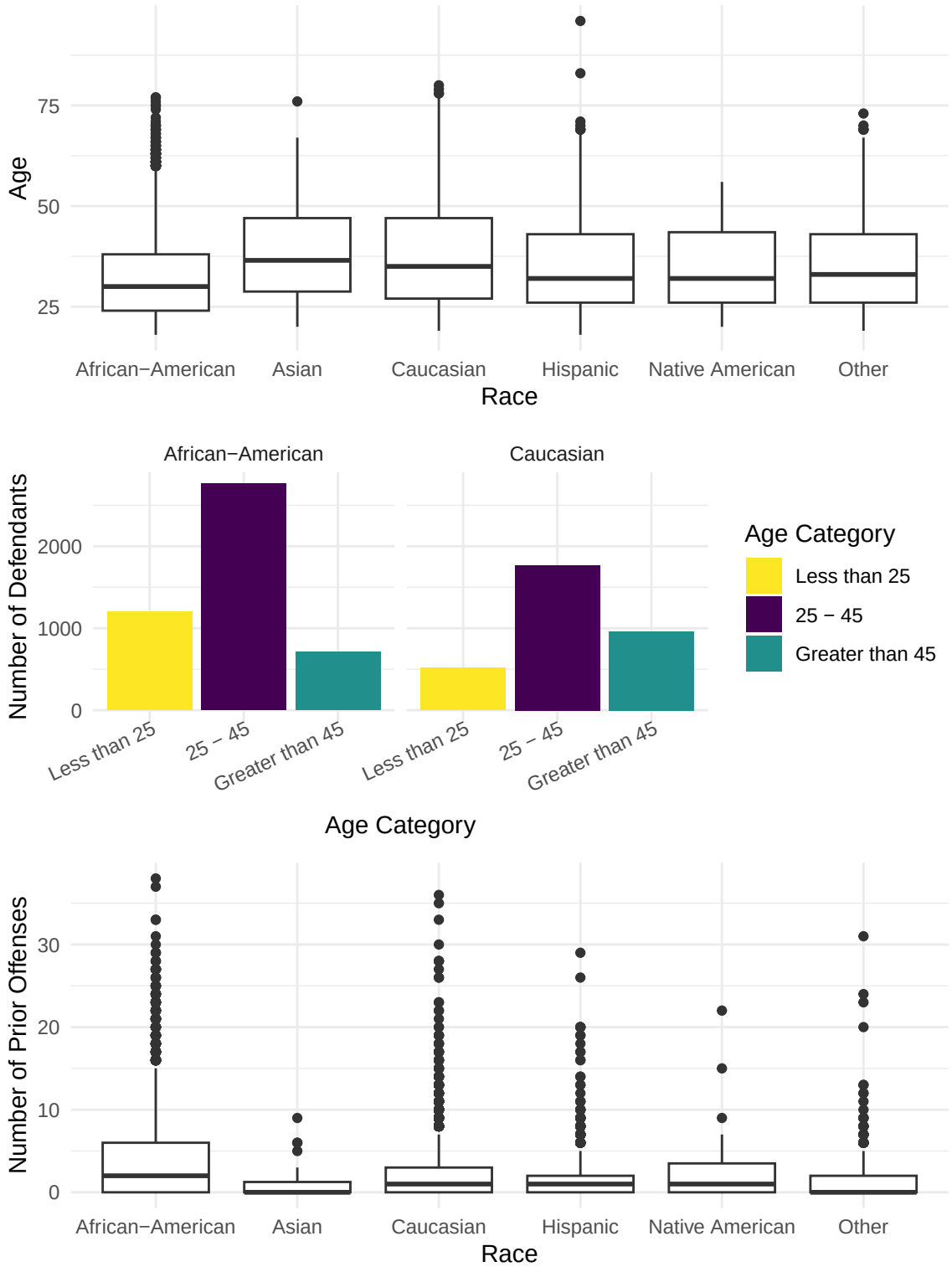


Figure 3.7: Distribution of COMPAS Tool Proxy Variables Stratified by Race among 9,387 Defandants in Broward County Florida, 2013-2014

Additionally, African-American defendants have the most prior offenses, on average, followed by their Native-American counterparts. When compared with Caucasian defendants, African Americans have almost twice as many prior offenses, suggesting a strong proxy relationship between race and prior offenses. Asian defendants have the least prior offenses. This is an important result that illustrates how a system that predisposes certain races to prison can perpetuate that discriminatory trend by using those same variables to predict the risk of committing another crime. The boxplot in Figure 3.7 helps to visualize this relationship more clearly.

This section illustrates that the protected attribute, race (A), has proxy relationships with some predictor variables (X). Even a group-blind classifier will not be entirely blind to race because of the correlations present and the information it gains about race from the proxy variables.

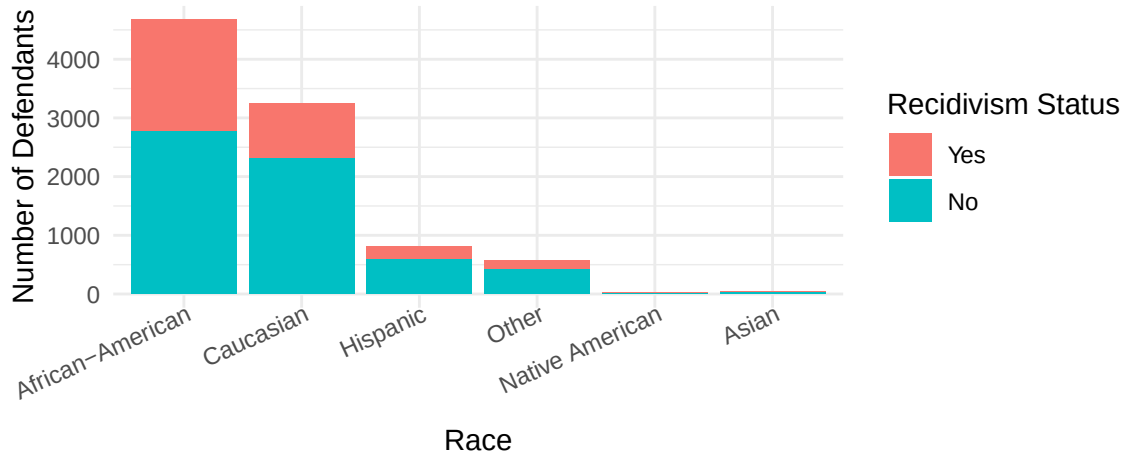


Figure 3.8: Distribution of Recidivism Prevalence Stratified by Race among 9,387 Defandants in Broward County Florida, 2013-2014

Most defendants do not recommit another crime, even when broken down by race as visualized in Figure 3.8. Because the class imbalance is in the same direction, a 'fair' model should not misclassify defendants in different directions based on race. However, the nature of the relationship between race and the predictor variables

results in discriminatory and unfair outcomes, as elucidated in Section 3.3.1.

Before employing the Seldonian framework on this data set, Section 3.4 fits a logistic regression to assess how similarly the COMPAS tool performs to a standard ML procedure.

3.4 Logistic Regression

Logistic regression is a statistical generalized linear model (GLM) specifically designed to predict dichotomous/ binary outcomes. In this case, logistic regression was used to model the probability of a defendant re-committing a crime within two years in Broward County, Florida, using the COMPAS data set. With a cutoff of 0.5, the resulting probabilities were divided into two bins: lower risk ($p < 0.5$) and higher risk ($p >= 0.5$). Given that logistic regression is one of the most widely used classification algorithms, this analysis will provide insights into how state-of-the-art traditional algorithms that follow the standard ML procedure described in Chapter 2.1 and do not consider fairness guarantees may perform on this data set.

3.4.1 Fitting the Logistic Regression Model

Recall that if every observation were classified in the majority class, an accuracy of 66% would be expected. This is a benchmark for the race-blind logistic regression model implemented in this section. The model will be trained on 70% of the data and evaluated on the remaining 30%. Only the five predictors (sex, age, marital status, juvenile offense, and prior offenses) will be fit to predict recidivism, the R code for which is displayed in Appendix D.

The accuracy was 70.2% on the training set, which is ~3% higher than the COMPAS tool's predictions and suggests that ~30% of defendants are misclassified. Overall

accuracy was consistent at 70.1% when evaluated on the testing set, indicating that the model did not over-fit on the training set and had good generalization performance. However, the ROC curve on Figure 3.9 indicates concerns about the model's sensitivity ($1 - FNR$) and specificity ($1 - FPR$). The diagonal line shows how the model would perform with random predictions. Curves that budge closer to the top left are preferred because those types of curves maximize the area under the curve (AUC), and the sensitivity and specificity of the model are closer to 100%. However, the AUC of this model is 68.8%.

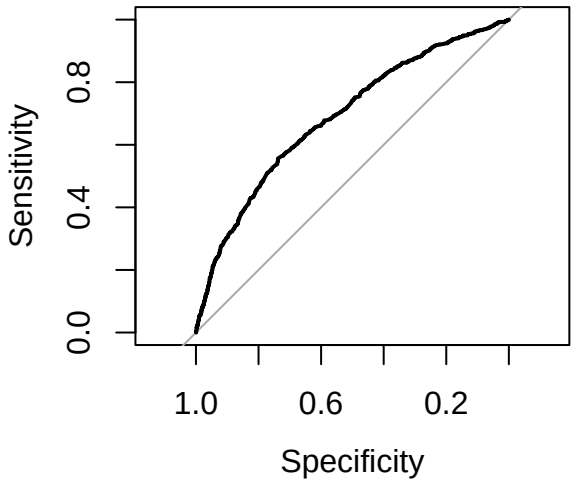


Figure 3.9: COMPAS Logistic Regression Receiver Operating Characteristic (ROC) Curve

3.4.2 Evaluating the Equality of Odds

Despite the improved accuracy, examining the direction of these error rates and performing a demographic analysis by race to assess the model's 'fairness' along racial lines reveals the same discrepancies observed in the COMPAS tool results.

Table 3.3 displays the overall false positive and false negative rates. As expected, the model performs better in classifying defendants who do not reoffend than those who do since that is the majority class. When evaluated on the test set, 91.05%

of defendants who did not reoffend are correctly classified as low-risk. Thus, only 8.95% of participants who do not reoffend are incorrectly classified as high-risk. This is a lower FPR than COMPAS. However, only 29% of defendants who reoffended are correctly classified as high-risk. Thus, 71% of defendants who reoffended are incorrectly classified as low-risk. This is a higher FNR than COMPAS.

Table 3.4 further breaks these results down, focusing on Black and White defendants. While the overall FPR for the model is 8.95%, notice that the FPR for Black defendants is 13.93%, compared to only 5.09% for White defendants. Similar to the COMPAS tool, Black defendants who do not reoffend are incorrectly misclassified as high risk at twice the rate that White defendants are. On the flip side, while the overall FNR for the model is 71%, notice that the FNR for Black defendants is 60.82%, compared to 86.17% for White defendants. Similar to the COMPAS tool, the logistic regression makes the opposite mistake in predicting recidivism for Black v White defendants, with White defendants being more likely to be incorrectly classified as low risk than their Black counterparts.

These results serve to highlight the real danger with using standard ML procedures, even when the sensitive variable itself is not included. These models have potential to perpetuate, and even introduce, harmful and discriminatory practices as observed in this Chapter. Although the details of Northepointe's COMPAS algorithm are kept secret, it's clear that traditional algorithms like logistic regression lead to comparable outcomes. In fact, a probability cutoff of $p = 0.34$, the probability of being in the positive class in this data set, on the logistic regression yielded the same accuracy (66.7%) as the COMPAS tool. Additionally, comparing the COMPAS decile scores to the logistic regression (LR) results reveals that the median decile score is 8 in the high-risk LR prediction and 3 in the low-risk LR prediction, further highlighting the similarity of these results. The next section uses the logistic regression model as a

Table 3.3: COMPAS Logistic Regression Error Rates (Percentage)

Risk	Reoffended	Did Not Reoffend
High	29	8.95
Low	71	91.05

Table 3.4: The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by Logistic Regression Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014

Recidivism Status	Predicted Risk	Black	White
Did Not Reoffend	High	13.9	5.09
Reoffended	Low	60.8	86.17

starting point, but places fairness constraints to enforce equality of odds for Black and White defendants.

3.5 Seldonian Classification

This section aims to conclude the chapter by illustrating how Seldonian algorithms can mitigate undesirable outcomes made by classifiers in practicality. Consistent with the notation in Chapter 2, Θ would be a set of classifiers and f a classifier performance measure, such as logistic loss or the log likelihood. For this application, the Seldonian algorithm will use the logistic loss as its primary objective function. The goal is to use the Seldonian framework to create a model that makes predictions about recidivism that are fair with respect to race, which will be simplified to just two levels: Black and White. Further research can extend this work to include more races.

3.5.1 Formulating the Seldonian Problem

Without fairness constraints, the standard ML problem is finding a solution θ that minimizes logistic loss. However, such a solution, as illustrated in Section 3.4 when fitting a logistic regression model, results in unequal error rates between Black and White defendants when using the COMPAS data set. Defining the discrimination statistic $d(\theta)$ to measure the difference in error rates as in Equation 3.1, then $d(\theta_{LR}) = 0.34$ (or 34.18%). Similarly, for the COMPAS tool, $d(\theta_{COMPAS}) = 0.45$ (or 44.7%) using this data set. This illustrates that the logistic regression performed slightly better than the COMPAS tool, both in overall accuracy and error rate disparities. However, there is still a significant discrepancy between the error rates for both races.

$$d(\theta) = abs[(FPR|\text{Black} - FPR|\text{White}) + (FNR|\text{White} - FNR|\text{Black})] \quad (3.1)$$

To minimize $d(\theta)$, $g(\theta)$ will be defined, for some ϵ, as:

$$g(\theta) = abs[(FPR|\text{Black} - FPR|\text{White}) + (FNR|\text{White} - FNR|\text{Black})] - \epsilon. \quad (3.2)$$

Recall that a Seldonian algorithm ensures that:

$$P(g(\theta) \leq 0) \geq 1 - \delta. \quad (3.3)$$

The problem can now be fully formulated as a Seldonian machine learning problem. That is, using gradient descent, an optimization algorithm for finding a local minimum of a differentiable function, the Seldonian objective is to minimize logistic loss subject

to the constraint:

$$P\{abs[(FPR|\text{Black}-FPR|\text{White})+(FNR|\text{White}-FNR|\text{Black})]-\epsilon \leq 0\} \geq 1-\delta;\ \delta = 0.05. \tag{3.4}$$

δ will be set to 0.05 to attain 95% confidence that separation (equalized odds) as defined in Chapter 1.2.2 is satisfied, that is, $d(\theta) \leq \epsilon$. The data will be partitioned into a training set that will be passed into the candidate selection mechanism to compute θ_c. The remaining partition will be used in the safety test to ensure probabilistic satisfaction of the constraint. ϵ will be set to four values: $0.2, 0.1, 0.05,$ and 0.01. The code to obtain a solution is available as part of the `seldonian-engine` Python library and is displayed in Appendix E, along with the data pre-processing.

3.5.2 Evaluating Performance and Fairness

A solution that passed the safety test was obtained for all four values of ϵ. However, as ϵ got smaller, the logistic loss increased, and the overall accuracy decreased. The accuracy was 68.2%, 65.59%, 64.7%, and 64.4% respectively for $\epsilon = 0.2, 0.1, 0.05, 0.01$. Additionally, the models got progressively worse in correctly classifying observations into the positive class, with the final model classifying every observation into the negative class. Each model had higher overall FNR and lower overall FPR than both the COMPAS tool and the logistic regression model. Tables 3.5, 3.6, 3.7, and 3.8 break up the FPR and FNR for Black and White defendants in each of these four cases. When $\epsilon = 0.2, 0.1, 0.05,$ and 0.01, note that $d(\theta) = 0.22, 0.06, 0.007,$ and 0 respectively.

In conclusion, the Seldonian algorithms successfully met the fairness constraints defined, with the FPR and FNR gaining more parity along racial lines for Black and

Table 3.5: The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by a Seldonian Algorithm ($\epsilon = 0.2$) Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014

Recidivism Status	Predicted Risk	Black	White
Did Not Reoffend	High	8.88	2.24
Reoffended	Low	73.82	88.82

Table 3.6: The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by a Seldonian Algorithm ($\epsilon = 0.1$) Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014

Recidivism Status	Predicted Risk	Black	White
Did Not Reoffend	High	1.87	0.3
Reoffended	Low	93.13	97.4

Table 3.7: The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by a Seldonian Algorithm ($\epsilon = 0.05$) Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014

Recidivism Status	Predicted Risk	Black	White
Did Not Reoffend	High	0.29	0.39
Reoffended	Low	98.73	98.17

Table 3.8: The Percent of Defendants who were Incorrectly Labeled Higher/Lower Risk by a Seldonian Algorithm ($\epsilon = 0.01$) Stratified by Recidivism Status and Race among 9,387 Defandants in Broward County Florida, 2013-2014

Recidivism Status	Predicted Risk	Black	White
Reoffended	Low	100	100

White defendants as ϵ became more conservative. However, there was a tradeoff with model informativeness and performance in enforcing greater fairness. The FNR got larger as the FPR got smaller until, ultimately, the model was non-informative despite perfectly satisfying the constraints (100% FNR and 0% FPR for both races). Accuracy also dropped as the value of ϵ decreased, although the original data set could have been more informative. Notably, considering all races, the lowest achievable accuracy would only be 4% lower than the highest accuracy attained by the logistic regression model.

Looking ahead, there are several avenues for future research. One direction involves extending this study to data sets with less class imbalance and higher overall predictive performance. By doing so, the tradeoffs inherent in incorporating fairness constraints into the traditional objective function in a practical setting can be better elucidated. Chapter 4 contributes to this understanding by conducting a simulation study to explore the theoretical and practical implications of the Seldonian framework in the classification setting.

Chapter 4 Seldonian Algorithms for Classification: A Simulation Study

Chapter 2 introduced the Seldonian framework, which offers probabilistic guarantees for satisfying defined behavioral constraints. However, the toy linear regression example demonstrated some of the limitations of Seldonian algorithms, particularly in scenarios with limited sample sizes where convergence may pose a challenge. In practice, sample sizes vary. Furthermore, Chapter 3 applied the Seldonian framework to a classification setting using the COMPAS data set. The goal was to produce recidivism predictions that elicit fairer outcomes along racial lines, with a specific focus on Black and White defendants. However, the application illustrated that while the behavioral constraint was satisfied in all four cases, there was a tradeoff in the model's accuracy. In other words, a non-informative model can be perfectly Seldonian, but what value does a theoretically fair non-informative model add?

To address this concern and further study these tradeoffs, this chapter investigates the efficacy and applicability of Seldonian algorithms in practical classification settings with class balance and better predictive performance than COMPAS. By conducting a simulation study, the aim is to evaluate whether Seldonian approaches can effectively produce fairer outcomes and mitigate discriminatory tendencies, drawing on the fairness notion of separation (or equality of odds) defined in Chapter 1.2 and set up in Chapter 3.5. Specifically, the objective is to assess the feasibility of leveraging

Seldonian algorithms to enhance the fairness and equity of predictive models across various practical classification tasks.

4.1 Simulation Design

Before conducting the simulation study and analyzing the results, this section provides detailed explanations of the simulation set-up and design.

4.1.1 Aims

This simulation study presents a proof of concept for using the Seldonian framework in classification problems with robust predictive performance. Seldonian algorithms *can* fail, especially with insufficient data, as elucidated in Chapter 2.3. On the flip side, as elucidated in Chapter 3.5, Seldonian algorithms can succeed in their objective for fairer outcomes but fail to produce a useful model. Solutions returned are also probabilistic and may not always satisfy the constraint despite passing the safety test. With these limitations in mind, this simulation study aims to empirically assess the predictive performance of Seldonian algorithms, compared to that of the standard ML approach, in practical classification settings.

4.1.2 Data-Generation Mechanism

Given that this is a proof-of-concept simulation study, the data-generation mechanism will follow a realistic design. Two of the most informative variables from the COMPAS data set were selected for a simpler set-up: the defendant's age (continuous) and whether or not the defendant had committed any prior offenses (binary). The choice to use the COMPAS data set as a starting point was so that the complex relationships between the predictor variables (X) and the protected attribute (A) may be retained,

especially given that the data was collected for a practical application and elucidates relationships that may be expected in the real world. There are only two levels of the protected attribute, race, that were retained for the study: Black and White.

To achieve better predictive performance, a linear combination of age and prior offenses was chosen after searching through a range of plausible values: $logit(p_i) = 5 - 0.2\ Age_i + 0.5\ PriorOffense_i \mid i \in \{1, 2, \ldots, 9387\}$, where p_i is the probability of a defendant recommitting a crime within two years. The response variable – whether or not a defendant recommitted a crime within two years – was then drawn from a Bernoulli distribution with probability of reoffense equal to p_i for each defendant. Finally, class balance was induced on the data set by randomly sampling, with replacement, 1250 observations from each of the four intersections of race and recidivism status: Black and White defendants who recidivated and did not recidivate. Note that this scheme (Appendix F) assumes equal prevalence of the response variable, Y, for both levels of the protected attribute.

The parent simulation data set, thus, contains 5000 observations, achieves perfect class balance, and has a baseline logistic regression accuracy of 77.96%. This offers a realistic improvement from the accuracy of 70.2% obtained from the COMPAS logistic regression. However, because of the class balance, the lowest achievable accuracy is now 50% (a random/ coin-flip model) rather than 64.4%. This will allow more room for the Seldonian algorithm model accuracies to vary.

One thousand total data sets of size $n = 500$, $n = 1000$, $n = 2500$, and $n = 5000$ will be sampled, with replacement, from this parent data set for the simulation study, with 250 data sets in each of the four partitions.

4.1.3 Target

The target of this simulation study is prediction, that is, to evaluate Seldonian algorithms for predictive performance in classification settings.

4.1.4 Methods

Because the Seldonian framework was designed to place probabilistic fairness constraints on traditional algorithms, solutions produced by Seldonian algorithms will be compared to those produced through the standard ML framework, specifically logistic regression, to assess the predictive performance and feasibility of the Seldonian framework.

Drawing from the fairness definitions described in Chapter 1.2 with a particular focus on the COMPAS example in Chapter 3, it may be less important that a model satisfies independence. That is, that the model, on average, has the same likelihood of a positive prediction for all levels of the protected attribute. Instead, it may be more realistic to expect that the model be equally wrong or equally correct in its predictions of Y for each protected attribute. For this reason, one fairness constraint will be set on the Seldonian algorithms to satisfy separation, otherwise known as equality of the error rates or equality of odds, as defined in Equation 4.1 within some margin ϵ and with $1 - \delta$ % confidence. Similar to Chapter 3.5, ϵ will be set to four levels: $0.2, 0.1, 0.05,$ and 0.01. δ will be set to 0.05 to guarantee 95% confidence.

$$g(\theta) : abs[(FPR|\text{A} = \text{a} - FPR|\text{A} = \text{b}) + (FNR|\text{A} = \text{a} - FNR|\text{A} = \text{b})] - \epsilon. \quad (4.1)$$

Simulated data sets will be generated independently before being fed into a logistic regression algorithm and the four Seldonian algorithms. The relevant performance measures, as detailed in Section 4.1.5, will be recorded for each trial. Two hundred

and fifty trials will be run for each sample size as described in Section 4.1.2. The code will be run in Python using the Rstudio software interface and the Amherst College High-Performance Computing System. The `seldonian` package is readily available for installation in Python and is equipped with functions that allow for the implementation of Seldonian algorithms. Other Python packages, such as `pandas`, `numpy`, and `sklearn`, will help conduct the rest of the simulation study.

4.1.5 Performance Measures

Compared to the logistic regression models, the predictive performance of the Seldonian models will be assessed along three dimensions: the probability of a Seldonian solution, the accuracy of the solutions, and the satisfaction or violation of the behavioral constraints set across all trials within a specific setting (sample size).

It is essential to account for the fact that the Seldonian algorithms will not always return a solution. Recording the probability of a solution being returned in each sample setting will be crucial for evaluating the practical feasibility of this framework. Additionally, for fair statistical comparisons, the first and second performance measures will be compared only with Seldonian solutions that converge. However, the performance of the logistic regression models in trials where no solutions were found using the Seldonian framework will be analyzed to elucidate learnings about the nature of those trials.

Additionally, for each simulation trial, the overall accuracy of both the convergent Seldonian models and the logistic regression model will be recorded and eventually averaged over the number of data sets in the specific sample size. Both the mean and the standard error will be reported in tabular format and visualized graphically to compare the models' predictive performances and, potentially, evaluate the trade-offs that may occur by employing the Seldonian framework and enforcing behavioral

constraints.

Finally, the satisfaction or violation of the behavioral constraint identified in Equation 4.1 will be assessed in two ways. First, for each sample size as described above, a count of the times both frameworks satisfied the behavioral constraint, by some margin ϵ, will be reported in tabular format. Additionally, the discrimination statistics as defined in Equation 4.2 will be recorded for each simulation trial. The mean and standard error will be reported and visualized for each sample setting to compare the magnitude and direction of the models' unfairness with regard to the separation fairness definition.

$$d(\theta) = abs[(FPR|\text{A} = \text{a} - FPR|\text{A} = \text{b}) + (FNR|\text{A} = \text{a} - FNR|\text{A} = \text{b})] \quad (4.2)$$

4.2 Simulation Results

As described in Section 4.1, the simulation results will be analyzed along 3 key performance measures: convergence, discrimination, and accuracy.

4.2.1 Probability of a Seldonian Solution

All logistic regression trials are expected to return a solution, thus a 100% convergence rate. However, while all the Seldonian trials will propose a candidate solution predicted to pass the safety test with 95% confidence, not all candidate solutions will pass the safety test itself. Table 4.1 illustrates this. For $\epsilon = 0.2, 0.1, 0.05$, the probability of a Seldonian solution was $>$ 90% for all sample sizes. In fact, for sample sizes $n = 1000, 2500, 5000$, the probability of a Seldonian solution was $> 96\%$, suggesting a higher convergence rate for looser fairness constraints and data set sizes greater

Table 4.1: Probability of Obtaining a Seldonian Solution

Sample Size	LR	SA (0.2)	SA (0.1)	SA (0.05)	SA (0.01)
500	100	99.6	93.2	89.6	71.2
1000	100	100.0	98.8	97.6	86.8
2500	100	99.6	100.0	98.8	56.8
5000	100	99.6	99.6	96.4	24.8

than $n = 500$. This is consistent with the results in Figure 2.6 from Chapter 2.3.1. Regardless, a small data set size did not matter for the loosest constraint of $\epsilon = 0.2$.

However, when $\epsilon = 0.01$, which was the tightest constraint, the probability of a solution dropped significantly, and there were no consistent trends across sample sizes, suggesting that the models had trouble passing the safety test in this case, regardless of sample size, likely because the constraint was too strict and not feasible for this data set. Surprisingly, the lowest observed probability of a solution (24.8%) is recorded when $\epsilon = 0.01, n = 5000$.

Overall, across all sample sizes, the probability of a solution generally reduced as the constraint tightened.

However, the probability of a solution that passes the safety test is not the end of the story. Table 4.2 records the proportion of solutions that satisfied the behavioral constraint among those that passed the safety test (Table 4.1). When $\epsilon = 0.01$, although a low probability of solution was previously observed, $> 92\%$ of returned solutions satisfied the behavioral constraint. Additionally, the probability was largest in the $n = 5000$ case, which previously had the lowest probability of returning a solution across the entire simulation study. Except for $n = 5000$, the probability of a Seldonian solution satisfying the behavioral constraint generally increased as the constraint got tighter, suggesting some trade-off between the probability of

Table 4.2: Satisfaction of the Behavioral Constraint by Seldonian Solutions that Passed the Safety Test

Sample Size	SA (0.2)	SA (0.1)	SA (0.05)	SA (0.01)
500	83.5	88.0	88.4	93.8
1000	85.2	83.4	88.1	93.1
2500	56.2	63.2	76.1	92.2
5000	90.8	24.9	61.4	95.2

returning a solution and the probability of that solution satisfying the behavioral constraints. However, there were no consistent trends as sample size increased across all four levels of ϵ, although the probability actually decreased for $\epsilon = 0.1, 0.05$. This is inconsistent with expectations, but perhaps the increased variability introduced by more observations makes it more difficult to constrain the models' unfairness.

Finally, Figure 4.1 visualizes these results to elucidate the convergence of Seldonian algorithms better. As discussed, the probability of a solution passing the safety test (light blue) generally decreases as the constraint gets tighter, regardless of sample size. Notably, the decrease is most drastic for the tightest constraint ($\epsilon = 0.01$), which has a better probability of a solution for smaller sample sizes than larger sample sizes. However, looking at the proportion of solutions that both pass the safety test and satisfy the defined behavioral constraint (dark blue) raises concerns about the efficacy of Seldonian algorithms, especially for the $n = 2500, 5000$ cases. The following three sections further assess the discrimination and accuracy of the Seldonian models that passed the safety test.

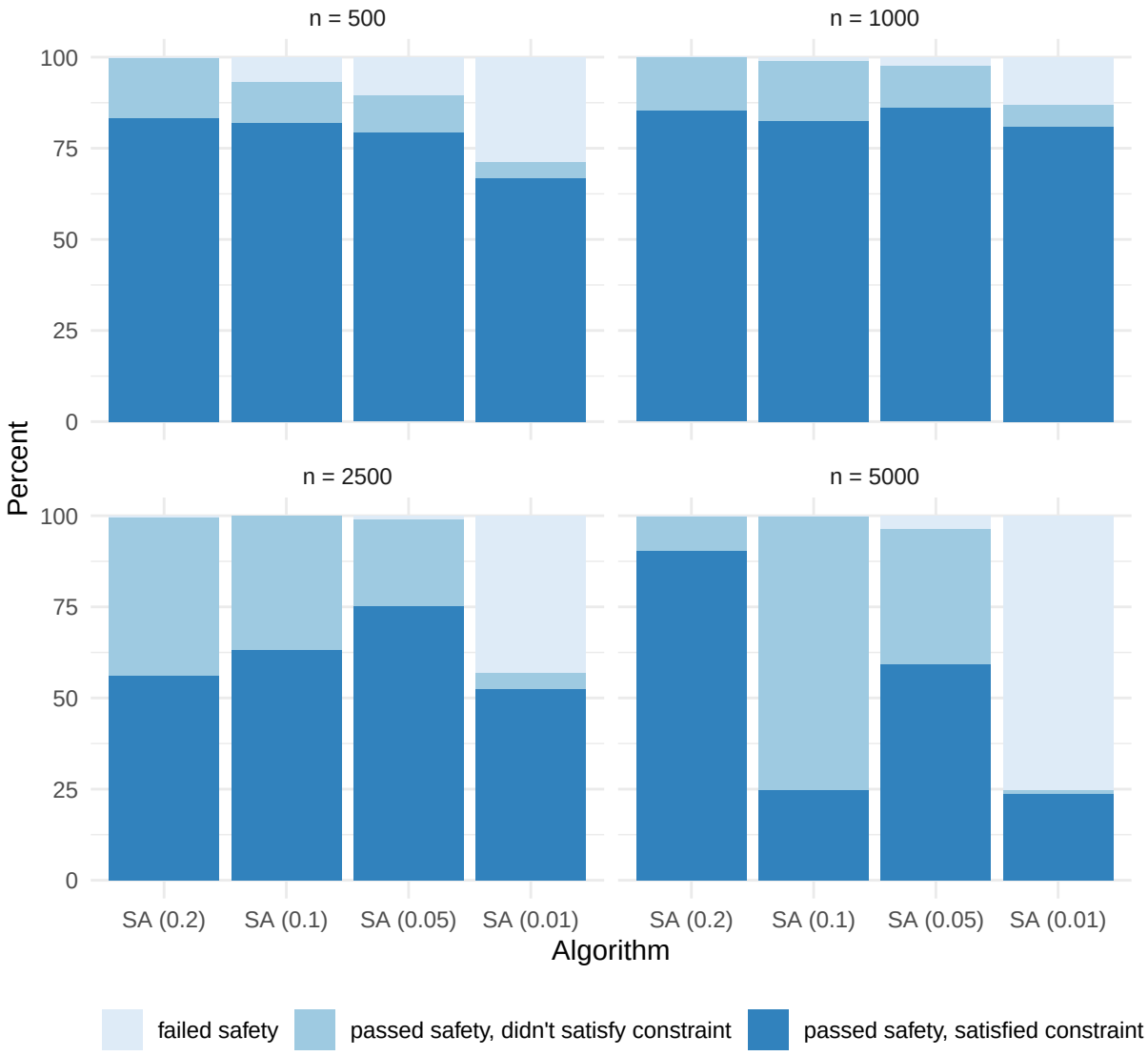


Figure 4.1: Probability of Returning a Solution and Satisfying the Constraint by Sample Size

4.2.2 Discrimination

CHECK blue figure size, adjust legend, axes etcx

4.2.3 Accuracy

4.2.4 The Trade-Off Between Discrimination and Accuracy

4.2.5 Non-Convergent Seldonian Models

4.3 Discussion

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:

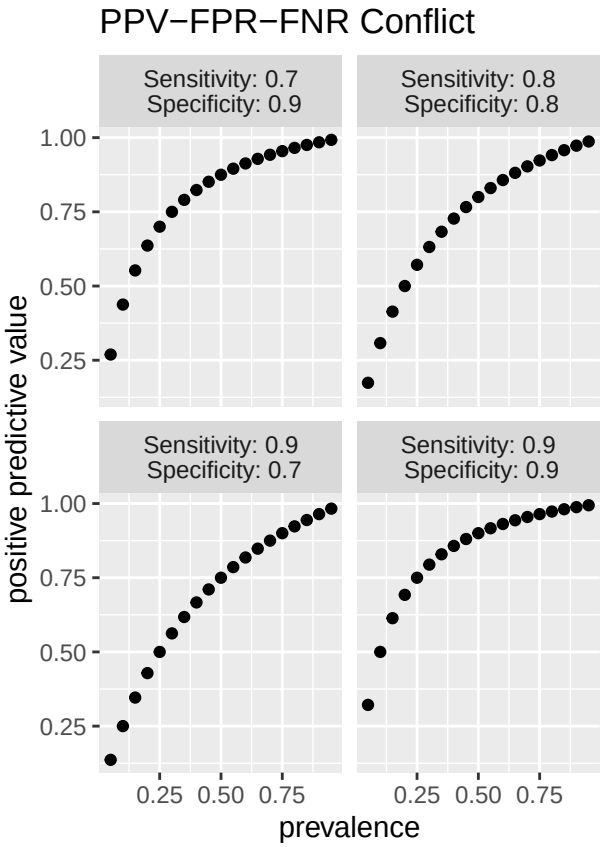
$$NPV = \frac{s_p(1-p)}{(1-s_e)p + s_p(1-p)}.$$

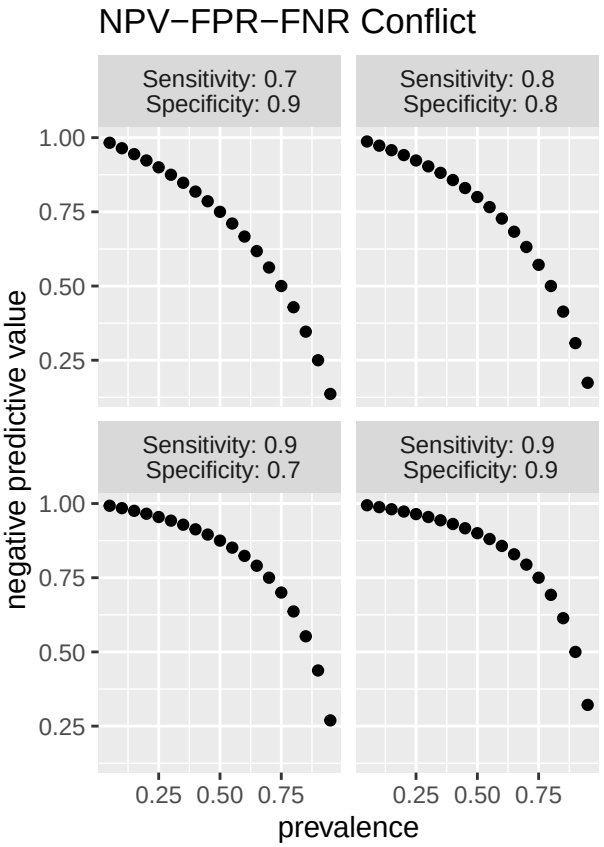
The R code chunk below fixes arbitrary sensitivity (1 - FNR) and specificity (1 - FPR) values to illustrate through the proceeding plots that as prevalence varies, then PPV/ NPV varies and cannot be equal as long as sensitivity and specificity are held constant, hence a conflict.

```
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 Quasi-Seldonian Linear Regression Theoretical and Python Implementation

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 the `sklearn` package 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 `nu` degrees of freedom for the corresponding probabilities `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 desired for the specific problem.

```
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. Because 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') # 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 can be plotted as shown below and in Figures 2.5, 2.6, 2.7, and 2.8.

```
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')
```

Appendix C SQL Script to Retrieve COMPAS Data

The following SQL script was used to retrieve the COMPAS data set used in Chapters 3 and 4 from the public COMPAS database available through the ProPublica Data Store and accessible through GitHub. The database contains seven tables, although the `summary` table is empty. The SQLite code below returns 12160 rows of defendants from Broward County, Florida, who were assessed using the COMPAS tool for risk of recidivating. There are 29 variables of interest returned, though many more could be selected if interested. Further data wrangling and data analysis are conducted in R, and significant findings are communicated in Chapters 3 and 4.

```
select

# select variables of interest
people.id,
compas.compas_person_id,
people.name,
people.first,
people.last,
people.sex,
people.race,
people.age,
people.age_cat,
compas.marital_status,
compas.custody_status,
people.juv_fel_count,
```

```
people.juv_misd_count,
people.juv_other_count,
people.priors_count,
people.days_b_screening_arrest,
people.c_days_from_compas,
people.c_charge_degree,
people.c_charge_desc,
compas.type_of_assessment,
compas.raw_score,
people.decile_score,
compas.score_text,
people.is_violent_recid,
people.num_vr_cases,
people.is_recid,
people.num_r_cases,
round(jail_agg.days_in_jail) as days_in_jail,
round(prison_agg.days_in_prison) as days_in_prison

from compas

# join the 'people' table
inner join people
on compas.person_id = people.id
and compas.first = people.first
and compas.last = people.last
and compas.decile_score = people.decile_score

# join jail history for each defendant
left join  (

select

jailhistory.person_id,
jailhistory.first,
jailhistory.last,
jailhistory.out_custody,
jailhistory.in_custody,
sum(
  distinct(
    julianday(
      jailhistory.out_custody
      ) - julianday(
```

```
        jailhistory.in_custody
        )
    )
  ) as days_in_jail

from jailhistory

inner join compas
on compas.person_id = jailhistory.person_id
and compas.first = jailhistory.first
and compas.last = jailhistory.last
and jailhistory.in_custody <= compas.screening_date

group by 1,2,3

) as jail_agg
on jail_agg.person_id = people.id
and jail_agg.first = people.first
and jail_agg.last = people.last

# join prison history for each defendant
left join  (

select

prisonhistory.person_id,
prisonhistory.first,
prisonhistory.last,
prisonhistory.out_custody,
prisonhistory.in_custody,
sum(
  distinct(
    julianday(
      prisonhistory.out_custody
      ) - julianday(
        prisonhistory.in_custody
        )
    )
  ) as days_in_prison

from prisonhistory
```

```
inner join compas
on compas.person_id = prisonhistory.person_id
and compas.first = prisonhistory.first
and compas.last = prisonhistory.last
and prisonhistory.in_custody <= compas.screening_date

group by 1,2,3

) as prison_agg
on prison_agg.person_id = people.id
and prison_agg.first = people.first
and prison_agg.last = people.last

# filter only for risk of recidivism
where type_of_assessment = 'Risk of Recidivism'
```

Appendix D Logistic Regression on the COMPAS Data Set

The R code chunk below fits a logistic regression model on the COMPAS data set.

```
# set seed
set.seed(123)

# drop missing observations
compas <- tidyr::drop_na(compas)

# train and test split
n <- nrow(compas)
train_index <- sample(1:n, 0.70 * n)
test_index <- setdiff(1:n, train_index)
train <- compas[train_index, ]
test <- compas[test_index, ]

# fit the logistic regression model on identified predictors
glm2 <- glm(is_recid ~ sex + age + marital_status +
              juv_offense + priors_count,
            data = train,
            family = binomial(logit))

# obtain the binary predictions on train set
glm2augment <- glm2 %>%
  broom::augment(type.predict = "response")
glm2augment <- mutate(glm2augment, binprediction = round(.fitted, 0))

# obtain the binary predictions on test set
preds <- predict(glm2, newdata=test, type="response")
test2 <- test %>%
```

```
  mutate(preds = preds,
         prediction = round(preds, 0))

# obtain the ROC curve
roccurve1 <- with(test, roc(is_recid ~ preds))
```

Appendix E Seldonian Classification on the COMPAS Data Set

This appendix section walks through how a Seldonian algorithm was fit on the COMPAS data set in Chapter 3 — the algorithm aimed to produce less disparate outcomes in error rates between Black and White defendants.

First, the Python environment can be set up in R using the `reticulate` package. The necessary Python libraries can then be imported as shown.

```
# set up Python environment in R
library(reticulate)
use_python("/cm/shared/apps/amh-Rstudio/python-3.11.4/bin/python3",
           required = TRUE)
```

```
# import necessary libraries
import numpy as np
import os

import pandas as pd
import json

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer

from seldonian.utils.io_utils import save_json
from seldonian.parse_tree.parse_tree import (ParseTree,
    make_parse_trees_from_constraints)
```

```
from seldonian.dataset import DataSetLoader
from seldonian.utils.io_utils import (load_json,save_pickle)
from seldonian.spec import SupervisedSpec
from seldonian.models.models import (
    BinaryLogisticRegressionModel as LogisticRegressionModel)
from seldonian.models import objectives

import matplotlib.pyplot as plt
```

Next, the data set needs to be formatted as appropriate for the Seldonian framework.

```
# point to the appropriate data file
f_orig = '/home/dasienga24/Statistics-Senior-Honors-Thesis/Data Sets/'
'COMPAS/compas_seldonian_bw.csv'

# list the column names for modeling
columns_orig = [
    "race","sex","age",
    "age_cat","marital_status","juv_offense",
    "priors_count", "is_recid"]

#read in the data
df = pd.read_csv(f_orig, header=1, names=columns_orig)

# split into inputs and outputs
X = df.drop(columns=["is_recid"])
y = df["is_recid"]

# one hot encode categorical features,
# scale numerical features using standard scaler
ct = ColumnTransformer([('c',OneHotEncoder(),
['race', 'sex', 'age_cat', 'marital_status']),
('n',StandardScaler(),['age','priors_count'])])

# apply transformation
X_transformed = ct.fit_transform(X)

# get names after one-hot encoding
output_columns = ct.get_feature_names_out(ct.feature_names_in_)

# make an output dataframe to save transformed X and y
```

```
outdf = pd.DataFrame(X_transformed,columns=output_columns)

# change names of columns
outdf.rename(columns={'c__race_African-American':'Black',
'c__race_Caucasian':'White', 'c__sex_Female':'Female',
'c__sex_Male':'Male', 'c__age_cat_25 - 45':'25_45',
'c__age_cat_Greater than 45':'Greater_than_45',
'c__age_cat_Less than 25':'Less_than_25',
'c__marital_status_Divorced':'Divorced',
'c__marital_status_Married':'Married',
'c__marital_status_Separated':'Separated',
'c__marital_status_Significant Other':'Significant_Other',
'c__marital_status_Single':'Single',
'c__marital_status_Unknown':'marital_status_Unknown',
'c__marital_status_Widowed':'Widowed',
'c__juv_offense_0':'juv_offense_0', 'c__juv_offense_1':'juv_offense_1',
'n__age':'age', 'n__priors_count':'priors_count' },
inplace=True)

# add label column and `juv_offense` back into final dataframe
outdf['juv_offense'] = df['juv_offense']
outdf['is_recid'] = y

# save final data frame
output_path_data="/home/dasienga24/Statistics-Senior-Honors-Thesis/'
'Data Sets/COMPAS/compas_seldonian_numeric_clean_bw.csv"

outdf.to_csv(output_path_data,index=False,header=False)

# save metadata json file
output_path_metadata="/home/dasienga24/Statistics-Senior-Honors-Thesis/'
'Data Sets/COMPAS/metadata_compas_seldonian_bw.json"

metadata_dict = {
  "regime":"supervised_learning",
  "sub_regime":"classification",
  "all_col_names":list(outdf.columns),
  "label_col_names":"is_recid",
  "sensitive_col_names":["Black", "White"]
}

with open(output_path_metadata,'w') as outfile:
```

```
    json.dump(metadata_dict,outfile,indent=2)
```

Next, a specificatio object is needed.

```
import autograd.numpy as np

data_pth = output_path_data
metadata_pth = output_path_metadata
save_dir = "/home/dasienga24/Statistics-Senior-Honors-Thesis/'
'Python/COMPAS Application/"
os.makedirs(save_dir,exist_ok=True)

# create data set from data and metadata file
regime='supervised_learning'
sub_regime='classification'

loader = DataSetLoader(regime=regime)

dataset = loader.load_supervised_dataset(
  filename=data_pth,
  metadata_filename=metadata_pth,
  file_type='csv')

sensitive_col_names = dataset.meta.sensitive_col_names

# use logistic regression model as the starting point
model = LogisticRegressionModel()

# set the primary objective to be log loss
primary_objective = objectives.binary_logistic_loss

from seldonian.spec import createSupervisedSpec

# define behavioral constraints
epsilon = 0.2
constraint_name = "equalized_odds"
if constraint_name == "equalized_odds":
  constraint_strs = [f'abs((FNR | [Black]) - (FNR | [White]))'
  '+ abs((FPR | [Black]) - (FPR | [White])) <= {epsilon}']
deltas = [0.05]
```

```
# create spec file
save_dir = "/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/'
'COMPAS Application/equalized_odds_0.2"

createSupervisedSpec(
            dataset=dataset,
            metadata_pth=metadata_pth,
            constraint_strs=constraint_strs,
            deltas=deltas,
            save_dir=save_dir,
            save=True,
            verbose=False)
```

With the specification object ready, a Seldonian algorithm can now be run.

```
from seldonian.seldonian_algorithm import SeldonianAlgorithm
from seldonian.utils.io_utils import load_pickle

# load the spec file -- can be replicated for the different epsilon values
specfile = '/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/'
'COMPAS Application/equalized_odds_0.2/spec.pkl'
spec = load_pickle(specfile)
SA_02 = SeldonianAlgorithm(spec)
passed_safety, solution = SA_02.run(write_cs_logfile=True)
if passed_safety:
  print("Passed safety test!")
else:
  print("Failed safety test")
print()
print("Primary objective (log loss) evaluated on safety dataset:")
print(SA_02.evaluate_primary_objective(branch='safety_test',
theta=solution))
```

Finally, we used the the inverse logistic function to obtain the predictive values for each observation from the solution return.

```
# get the solution -- can be replicated for the different epsilon values
SA_02.cs_result["candidate_solution"]

# store the coefficients
coefficients = SA_02.cs_result["candidate_solution"]

# get the intercept
intercept = coefficients[0]

# separate the predictor variables from
# the sensitive variable and the response variable
X_outdf = outdf.drop(columns = ['is_recid', 'Black', 'White'])
X_sens = outdf[['Black', 'White']]
y_outdf = outdf['is_recid']

# compute the predictive values
linear_combination = np.dot(X_outdf, coefficients[1:]) + intercept
pred_probs_02 = 1 / (1 + np.exp(-linear_combination))
```

```
# store the results
seldonian_results = pd.DataFrame({
  'is_recid': y_outdf,
  'pred_0.2': pred_probs_02,
  'pred_0.1': pred_probs_01,
  'pred_0.05': pred_probs_005,
  'pred_0.01': pred_probs_001})
seldonian_results = pd.concat([X_outdf, X_sens, seldonian_results],
axis = 1)
```

Finally, the binary risk predictions can be obtained using a 0.5 probability threshold.

```
# define threshold
threshold = 0.5

# create risk columns
risk_02 = np.where(pred_probs_02 >= threshold, 1, 0)
risk_01 = np.where(pred_probs_01 >= threshold, 1, 0)
risk_005 = np.where(pred_probs_005 >= threshold, 1, 0)
risk_001 = np.where(pred_probs_001 >= threshold, 1, 0)
```

```
# add risk columns to data frame
seldonian_results['risk_0.2'] = risk_02
seldonian_results['risk_0.1'] = risk_01
seldonian_results['risk_0.05'] = risk_005
seldonian_results['risk_0.01'] = risk_001

# write the data frame to a CSV file
seldonian_results.to_csv('/home/dasienga24/Statistics-Senior-Honors-Thesis/'
'Data Sets/COMPAS/compas_seldonian_results_bw.csv', index=False)
```

Appendix F Generating the Simulation Parent Data Set

This appendix section displays the R code used to generate the parent simulation data set for Chapter 4.

```
set.seed(123)

# define a linear combination of predictors as desired
linear_combination = 5  - 0.2 * compas_sim$age +
  ifelse(compas_sim$prior_offense == 1, 0.5, 0)

# pass through an inverse-logit function
probs = exp(linear_combination) / (1 + exp(linear_combination))

# generate Bernoulli RVs for y
is_recid_sim = rbinom(nrow(compas_sim), 1, probs)

# join to original data frame
compas_sim_balanced_final <- cbind(compas_sim, is_recid_sim)

# induce balance
compas_b_y <- compas_sim_balanced_final %>%
  filter(race == "African-American" & is_recid_sim == 1)

compas_b_n <- compas_sim_balanced_final %>%
  filter(race == "African-American" & is_recid_sim == 0)

compas_w_y <- compas_sim_balanced_final %>%
  filter(race == "Caucasian" & is_recid_sim == 1)
```

```r
compas_w_n <- compas_sim_balanced_final %>%
  filter(race == "Caucasian" & is_recid_sim == 0)

compas_b_y_balanced <-
  compas_b_y[sample(nrow(compas_b_y), 1250, replace = TRUE), ]
compas_b_n_balanced <-
  compas_b_n[sample(nrow(compas_b_n), 1250, replace = TRUE), ]
compas_w_y_balanced <-
  compas_w_y[sample(nrow(compas_w_y), 1250, replace = TRUE), ]
compas_w_n_balanced <-
  compas_w_n[sample(nrow(compas_w_n), 1250, replace = TRUE), ]

compas_sim_balanced_final <- rbind(
  compas_b_y_balanced,
  compas_b_n_balanced,
  compas_w_y_balanced,
  compas_w_n_balanced
)
```

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.

Broward County Clerk's Office, Broward County Sherrif's Office, Florida Department of Corrections, & ProPublica. (2024). COMPAS Recidivism Risk Score Data [Data set]. ProPublica Data Store. Retrieved from https://www.propublica.org/datastore/dataset/compas-recidivism-risk-score-data-and-analysis

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

The Sentencing Project. (2018). Report to the united nations on racial disparities in the u.s. Criminal justice system. Retrieved from https://www.sentencingproject.org/reports/report-to-the-united-nations-on-racial-disparities-in-the-u-s-criminal-justice-system/

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.