ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

FairLib: A Toolkit for Bias Analysis and Mitigation in AI Systems

Tesi di laurea in:
Intelligent Systems Engineering

Relatore
Prof. Giovanni Ciatto

Candidato
Valerio Di Zio

Correlatori
Prof. Andrea Omicini
Prof.ssa Roberta Calegari

I Sessione di Laurea
Anno Accademico 2025-2026

Abstract

Automated decision-making systems are rapidly permeating socially sensitive domains such as finance, healthcare, justice, and autonomous mobility. While these data-driven solutions can increase efficiency, they can also perpetuate or amplify existing inequities whenever the underlying algorithms exhibit unfair behaviour. This thesis provides a systematic investigation of algorithmic fairness, clarifying the multiple, often competing, formal definitions of fairness adopted in the literature and mapping them to the practical risks of bias and discrimination that arise throughout the machine-learning pipeline.

After surveying the principal sources of bias—data imbalance, historical prejudice, model opacity, and feedback loops—the work reviews state-of-the-art mitigation strategies grouped into three families: pre-processing (data-repair and resampling techniques), in-processing (fairness-aware losses, constraints, and regularisers), and post-processing (prediction-adjustment and explanation tools). Building upon these foundations, the thesis introduces **FairLib**: a modular, open-source library *developed to address limitations observed in existing fairness toolkits* that unifies bias-diagnosis metrics and mitigation algorithms behind a consistent API. FairLib is designed to be model-agnostic, seamlessly integrating with popular ML frameworks, and to facilitate reproducible experimentation through configurable pipelines.

A preliminary evaluation on *canonical fairness benchmark datasets* indicates that selected FairLib pipelines can measurably reduce unfairness while leaving overall predictive accuracy broadly unchanged. Although limited to a modest set of benchmarks, these findings suggest that systematic fairness interventions are achievable without prohibitive performance trade-offs.

By coupling a critical analysis of fairness concepts with a practical, extensible toolkit, this thesis aims to foster greater transparency and accountability in artificial-intelligence systems and to assist practitioners in deploying models that respect fundamental principles of equity.

Dedica qui!

Contents

Chapter 1

Introduction

Artificial intelligence (AI) systems increasingly mediate decisions about credit, employment, healthcare, justice, and mobility. Their scale and opacity mean that even small statistical biases can translate into systematic, large-scale harm. The goal of *algorithmic fairness* is therefore to ensure that automated decisions do not introduce or amplify unjustified discrimination against protected groups [MMS+21]. Achieving that goal, however, remains challenging because unfairness can emerge at many points in the machine-learning (ML) pipeline, from data collection to model deployment, and because technical, legal, and ethical notions of "fair" often conflict in practice [SG21].

1.1 From Bias to Mitigation: Scope of This Report

1. **Diagnose the problem:** We survey the landscape of algorithmic unfairness, definitions of fairness, sources of bias (representation, measurement, aggregation, and historical), and the **feedback loops** that can entrench disparities. We also review state-of-the-art mitigation strategies that operate at three key stages of the ML lifecycle: *pre-processing* (data repair and reweighting), *in-processing* (fairness-constrained learning), and *post-processing* (output adjustment).

2. **Offer a practical solution:** In this study we present **FairLib**, a modular Python library that consolidates a broad spectrum of fairness metrics and bias-mitigation algorithms within a single, production-oriented interface. By elevating bias detection, remediation, and continuous auditing to integral components of the machine-learning development lifecycle, FairLib significantly reduces the practical barriers to deploying equitable intelligent systems.

1.2 Why Another Library? Limitations of the Status Quo

Although IBM's *AI Fairness 360* (AIF360) [BDH+18] and Microsoft's *Fairlearn* [BDE+20] have stimulated important research, they remain ill-suited for day-to-day engineering because they exhibit four persistent shortcomings:

- **Fragmented APIs and terminology:** Conceptually identical operations are exposed through incompatible class names, argument conventions, and data structures, forcing practitioners to maintain fragile "glue code" whenever they switch between metrics or mitigation algorithms.

- **Verbosity and operational complexity:** Even elementary tasks—such as computing a single fairness metric or launching a standard debiasing pass—require boilerplate scripts and nested wrapper objects, making the tool harder to learn and discouraging rapid experimentation.

- **Binary-only focus:** Most built-in metrics and mitigation routines assume *binary* sensitive attributes and *binary* classification targets; multi-class settings are either unsupported or require significant user re-implementation.

- **Integration barriers and rigid internal models:** Neither toolkit plugs cleanly into existing *PyTorch* or *scikit-learn* pipelines: mitigation methods are hard-wired to internally defined estimators that can be adjusted only through superficial hyper-parameters. A practitioner cannot, for example,

inject a domain-specific neural network as the base learner for a fairness algorithm without rewriting large portions of the codebase.

1.3 FairLib Objectives

FairLib is explicitly engineered so that each objective eliminates a corresponding limitation identified above:

- **Unifying fragmented APIs:** A coherent, scikit-learn–style interface (`fit`, `transform`, `predict`) and consistent terminology replace the incompatible conventions that plague existing libraries, removing the need for brittle integration code.
- **Reducing verbosity and complexity:** Group- and individual-level metrics share a vectorised NumPy core, so computing a metric or launching a debiasing routine becomes a one-liner, dramatically accelerating prototyping.
- **Extending beyond binary settings:** All fairness metrics and evaluation utilities natively support multi-class targets and both binary and categorical sensitive attributes, while debiasing algorithms offer varying levels of support for multi-class settings together addressing key gaps in existing fairness libraries.
- **Providing end-to-end debiasing:** Harmonised *pre-* and *in-processing* modules (with post-processing on the roadmap) let practitioners inject fairness at the point most compatible with data-governance and privacy constraints.
- **Enabling customisable trade-offs and models:** Every mitigation algorithm surfaces its fairness penalty as a hyperparameter and accepts a user-supplied estimator, which can range from linear models to bespoke neural networks, thereby supporting automatic exploration of the fairness–utility frontier without sacrificing architectural flexibility.
- **Ensuring seamless workflow integration:** Native PyTorch wrappers and drop-in scikit-learn compatibility allow FairLib components to slot directly

into production notebooks, model registries, and CI/CD pipelines, overcoming the rigid, "model-locked" designs of prior toolkits.

Chapter 2

Background

2.1 Introduction to AI Fairness

The use of machine learning algorithms has permeated every aspect of our lives: from movie recommendations to hiring decisions and even to high-stakes areas such as loan approvals or risk assessments in the judicial system. Although these systems offer significant advantages—such as processing vast amounts of information without fatigue—they are also susceptible to biases that can lead to unfair decisions. As machine learning systems continue to be deployed in increasingly sensitive domains, ensuring fairness becomes a critical concern for developers, policymakers, and society at large [MMS+21].

The concept of fairness in AI is multifaceted and involves technical, legal, and ethical considerations. While technical solutions can address some aspects of bias, a comprehensive approach requires understanding the social contexts in which these systems operate and the historical inequalities they might perpetuate or amplify. This report provides an overview of fairness in AI, including its definitions, origins of bias, mitigation strategies, and applications across various domains.

2.1.1 What is Fairness?

Fairness refers to the principle that every individual or group should be treated justly and impartially. In decision-making, this means that processes, whether human or automated, should not introduce unjustified discrimination based on

inherent or acquired characteristics such as gender, ethnicity, or socioeconomic status [MNDWLS24].

The notion of fairness is deeply rooted in philosophical and legal traditions, with concepts such as distributive justice, procedural fairness, and equal opportunity providing frameworks for understanding what constitutes fair treatment [KKBK21]. In the context of automated decision systems, fairness extends these principles to ensure that algorithms do not perpetuate or amplify existing social biases [BCG21].

For example, in a recruitment system, fairness requires that candidates be evaluated solely on their skills and qualifications, free from biases related to gender or ethnicity [Che23]. Similarly, in credit scoring, a fair system would assess creditworthiness based on relevant financial factors rather than characteristics protected by anti-discrimination laws [GGR24].

It's worth noting that fairness is context-dependent, and what constitutes fair treatment may vary across different domains and cultures. Nevertheless, certain fundamental principles, such as non-discrimination and equal treatment of similar cases, remain consistent across contexts [KKBK21].

2.1.2 The Importance of Fairness in AI Systems

Ensuring fairness in artificial intelligence is crucial, especially when these systems make decisions that directly affect people's lives [CDG23, YW24]. Unfair AI systems can perpetuate and even exacerbate existing social inequalities, affecting individuals' access to opportunities, resources, and services.

A canonical example is the COMPAS system, used in U.S. courts to assess the risk of recidivism. Studies have shown that COMPAS tends to produce higher false-positive rates for African-American defendants compared to Caucasian defendants under similar conditions. This means that Black defendants were more likely to be incorrectly labeled as high-risk, potentially leading to harsher sentencing or denial of parole [YW24].

Beyond the justice system, fairness concerns arise in numerous domains:

- **Employment:** Resume screening algorithms might inadvertently favor candidates from certain demographic groups or educational backgrounds [KBA24].

- **Healthcare:** Diagnostic tools calibrated on data from predominantly one demographic group may be less accurate for others[NHS+24].
- **Financial Services:** Credit scoring algorithms might systematically disadvantage certain communities, reinforcing historical patterns of exclusion [Fer23].
- **Education:** Automated evaluation systems might penalize students whose learning styles or language patterns differ from the norm [KBA24].

The consequences of unfair AI systems extend beyond individual harms to societal impacts. When automated systems systematically disadvantage certain groups, they can erode trust in technology, reinforce stereotypes, and contribute to social polarization. Instead, fair AI systems have the potential to promote inclusiveness, reduce discrimination and create more equitable outcomes [Fer23].

2.1.3 Origins of Bias and the Feedback Loop

Bias in AI can stem from multiple sources, each requiring different approaches for mitigation:

- **Bias in the Data:** If the training data is skewed or unrepresentative of the true population, the algorithm will learn and reproduce these biases, sometimes even amplifying them. For instance, facial recognition systems trained predominantly on light-skinned faces perform worse on darker-skinned individuals [BG18].
- **Bias in the Problem Formulation:** How we define the problem and choose variables can introduce bias. For example, using proxies like ZIP codes instead of more direct measures can inadvertently incorporate demographic biases if neighborhoods are segregated.
- **Bias in the Algorithms:** Even with unbiased data, design choices—such as optimization functions, regularization techniques, or modeling assumptions—can introduce unfairness. For instance, prioritizing overall accuracy might lead to models that perform well on majority groups but poorly on minorities [NHS+24].

- **Bias in Evaluation:** If we evaluate models using metrics that do not account for fairness considerations, we might deploy systems that appear successful but discriminate in practice.

A particularly concerning phenomenon is the **feedback loop**, where biased predictions influence future data collection, which in turn reinforces the original bias [EFN+17]. For instance, predictive policing algorithms might direct more officers to neighborhoods already experiencing higher policing, leading to more arrests, which then reinforce the prediction that these areas have higher crime rates. This creates a self-fulfilling prophecy that can entrench and amplify existing disparities.

Historical biases present in society can also manifest in AI systems through the data they're trained on. Language models trained on internet text may inherit sexist or racist associations present in their training corpus. Word embeddings trained on standard text corpora reflect gender and racial stereotypes present in society [CBN17].

2.1.4 Types of Bias and Sources of Discrimination

Understanding the various types of bias that can affect AI systems is essential for developing effective mitigation strategies. A number of frameworks have been proposed to categorize these biases, including those affecting data, model design, and deployment contexts [MMS+21, SG21].

- **Representation Bias** arises when the dataset does not adequately reflect the diversity of the population. For instance, Buolamwini and Gebru found that commercial facial recognition systems perform worst on darker-skinned women, illustrating severe underrepresentation in benchmark datasets [BG18, SG21].

- **Measurement Bias** occurs when proxies are used in place of direct measurements. For example, arrest records used in predictive policing introduce racial disparities due to historically biased law enforcement practices [EFN+17, SG21].

- **Aggregation Bias**, **Evaluation Bias**, **User Interaction Bias**, and **Temporal Bias** are discussed as distinct sources of harm that emerge when models are designed, validated, or deployed without accounting for subgroup-specific differences or time-based shifts. These were formally characterized in lifecycle frameworks proposed by Suresh and Guttag [SG21].

- **Historical Bias** reflects the reproduction of long-standing social inequalities. Language models, for instance, have been shown to encode and replicate gender and racial stereotypes learned from text corpora [CBN17].

- **Algorithmic Bias** stems from model design decisions, including feature selection, optimization objectives, or regularization strategies, which may produce disparate outcomes even when the training data is balanced [MMS+21].

Discrimination in AI can be **direct**, involving explicit use of protected attributes, or **indirect**, where neutral criteria disproportionately impact certain groups. These forms can be explainable, meaning they can be justified by relevant non-protected attributes, or unexplainable, which is considered unfair or illegal. Frameworks for quantifying and mitigating such discrimination have been proposed to distinguish between the two [KZC12, CDG23].

Understanding these distinctions is crucial for ethical and legal assessments of AI. While certain disparities may reflect legitimate factors (e.g., qualifications), others are manifestations of unjustified bias that require targeted safeguards in both design and deployment phases.

2.1.5 Definitions of Fairness

One of the central challenges in developing fair AI systems is that there is no single, universally agreed-upon definition of fairness. Different notions of fairness capture different intuitions and may be more appropriate in different contexts [VR18] [MPB+21].

Some of the most commonly used formal definitions include:

- **Demographic Parity:** Requires that the probability of a positive outcome is the same across all groups, regardless of protected attributes. Mathemat-

ically, for a decision $\hat{Y}$ and a protected attribute A:

$$P(\hat{Y} = 1|A = a) = P(\hat{Y} = 1|A = b) \text{ for all values } a, b \text{ of } A$$

This definition aims for equal representation but may not be appropriate when the base rates genuinely differ across groups.

- **Equalized Odds:** Demands that, conditioned on the true outcome, both true positive and false positive rates are equal across groups [HPS16]:

$$P(\hat{Y} = 1|Y = y, A = a) = P(\hat{Y} = 1|Y = y, A = b) \\ \text{for } y \in \{0, 1\} \text{ and all values } a, b \text{ of } A$$

This ensures that the algorithm's errors are distributed fairly across groups.

- **Equal Opportunity:** A relaxation of equalized odds that focuses solely on ensuring that individuals who truly belong to the positive class have an equal chance of being correctly classified:

$$P(\hat{Y} = 1|Y = 1, A = a) = P(\hat{Y} = 1|Y = 1, A = b) \text{ for all values } a, b \text{ of } A$$

This is particularly relevant in contexts like hiring, where we want qualified candidates to have equal chances regardless of their demographic group.

- **Predictive Parity:** Requires that the precision of the classifier is the same across all groups:

$$P(Y = 1|\hat{Y} = 1, A = a) = P(Y = 1|\hat{Y} = 1, A = b) \text{ for all values } a, b \text{ of } A$$

This ensures that a positive prediction has the same meaning regardless of group membership.

Importantly, these definitions can conflict with each other [KMR16]. This creates inherent trade-offs that must be navigated based on the specific context and ethical priorities.

The choice of fairness definition should be guided by the specific application domain, the nature of the task, the potential harms of different types of errors,

and the ethical values being prioritized. For instance, in medical diagnosis, we might prioritize equality of false negative rates to ensure that serious conditions aren't missed for any group, while in criminal justice, we might prioritize equality of false positive rates to avoid wrongful punishment.

2.1.6 Trade-offs Between Fairness and Other Objectives

An important consideration in implementing fair AI systems is the potential trade-off between fairness and other objectives such as accuracy, efficiency, or interpretability.

The most widely discussed trade-off is between fairness and accuracy. Imposing fairness constraints typically reduces a model's overall predictive performance, as measured by conventional metrics like accuracy or F1 score. This creates a Pareto frontier where improvements in fairness come at the cost of reduced accuracy, and vice versa. The extent of this trade-off varies depending on the specific fairness definition, the dataset characteristics, and the learning algorithm used.

Beyond accuracy, fairness may also trade off against:

- **Interpretability:** More complex models that incorporate fairness constraints may be harder to interpret, reducing transparency and making it difficult to identify sources of bias;

- **Computational Efficiency:** Many fairness-aware algorithms require additional computational resources for training and inference, potentially limiting their applicability in resource-constrained environments;

- **Privacy:** Ensuring fairness often requires collecting and analyzing sensitive demographic data, which may conflict with privacy objectives or regulations like GDPR;

- **Individual Utility:** Group fairness measures may sometimes reduce utility for specific individuals who would have received positive outcomes under an unfair but more accurate model;

- **Short-term vs. Long-term Fairness:** Interventions that promote fairness in the short term may have unintended consequences that reduce fairness in the long term, or vice versa.

Achieving fairness in AI systems isn't just a technical challenge — it involves making value-based decisions about which goals matter most in a given context. These decisions should include input from people with different backgrounds, especially those who might be affected by the system, and should be guided by ethical principles, legal standards, and domain-specific knowledge.

It's also important to remember that these trade-offs aren't fixed. They depend on things like data quality, model complexity, and how the problem is defined. Better data collection, more advanced algorithms, and clearer definitions of fairness can help ease — though not completely solve — these tensions [KMR16, CR20].

2.2 Methods for Mitigating Bias

Researchers have developed numerous approaches to address bias in AI systems, which can be categorized based on the stage of the machine learning pipeline where they're applied:

2.2.1 Pre-processing techniques

These methods focus on transforming the input data to remove discriminatory patterns before model training. Examples include:

- Reweighting or resampling data to balance representation across groups;
- Transforming features to remove correlations with protected attributes [KZC12];
- Learning fair representations that preserve task-relevant information while obscuring protected attributes [MMS+21];
- Data augmentation to generate synthetic samples for underrepresented groups [MMS+21].

2.2.2 In-processing techniques

These approaches incorporate fairness considerations directly into the learning algorithm, often by modifying the objective function or adding constraints. Examples include:

- Adversarial debiasing, which uses an adversarial approach to remove information about protected attributes from the learned representations [MMS+21];
- Fair classification with constraints, which explicitly incorporates fairness criteria as constraints in the optimization problem [CDG23];
- Fair reinforcement learning, which modifies reward functions to account for fairness considerations [MMS+21].

2.2.3 Post-processing techniques

These methods adjust the output of already trained models to ensure fairness. Examples include:

- Threshold optimization, which applies different decision thresholds for different groups to equalize error rates;
- Calibration techniques that ensure predictions have the same meaning across groups;
- Reject option classification, which identifies and manually handles cases in the critical region between positive and negative classifications [KZC12].

2.2.4 Comparative Analysis of Mitigation Approaches

Each of these methods presents trade-offs. **Pre-processing** techniques offer flexibility and compatibility with any model but may not fully eliminate bias. **In-processing** techniques often yield better fairness-accuracy trade-offs but require access to model internals. **Post-processing** techniques are typically easier to implement but can be less robust [MMS+21, SG21].

The choice of mitigation strategy should be guided by practical considerations such as:

- Access to protected attributes during training and deployment;
- Computational resources available;
- Regulatory requirements regarding the use of protected attributes;
- The specific fairness definition being targeted.

Importantly, bias mitigation is not a one-time fix. Effective fairness in AI requires ongoing auditing, evaluation, and adjustment as societal values, population characteristics, and contexts evolve [SG21].

Recent work emphasizes the importance of domain-specific fairness frameworks. For instance, fairness in healthcare may prioritize equitable health outcomes over statistical parity, whereas fairness in education or finance may involve different risk-benefit trade-offs [MMS+21].

2.3 Fairness in Different Application Domains

The implementation of fairness principles varies significantly across different domains, reflecting the diverse nature of AI applications:

- **Criminal Justice:** Beyond the COMPAS example, many risk assessment tools are used to make decisions about bail, sentencing, and parole. Fairness is especially important here because these decisions have serious consequences for people's lives and happen within a system that has a history of discrimination. Research shows that even factors that seem neutral, like past arrests, can reflect racial bias caused by unequal policing [BHJ+17, EFN+17]. Fairness in this area means finding the right balance between keeping the public safe and protecting individual rights, while also being careful not to repeat injustices from the past.
- **Healthcare:** AI is playing a growing role in healthcare, from diagnosing illnesses to suggesting treatments and managing how resources are distributed. One study [OPVM19] found that a commonly used algorithm gave lower risk

scores to Black patients than to White patients with similar health conditions, which led to fewer referrals for extra care. Ensuring fairness in healthcare requires more than just accounting for demographics — it also means recognizing variations in how diseases appear, how different groups respond to treatment, and the unequal access to medical services.

- **Finance:** Credit scoring, loan approval, insurance pricing, and other financial algorithms can significantly impact individual economic opportunities. Studies have found evidence of disparate impacts in mortgage lending, with certain demographic groups receiving higher interest rates or being denied loans more frequently, even after controlling for relevant financial factors [BMSW22]. Fairness in financial contexts must balance risk assessment with ensuring equal access to financial services and preventing the perpetuation of historical economic disparities.

- **Education:** AI systems are used for admissions decisions, student assessment, personalized learning, and resource allocation in educational settings. Automated grading systems, for instance, may penalize certain writing styles or language patterns associated with particular cultural backgrounds. Fairness in education requires considering diverse learning styles, cultural contexts, and educational backgrounds while ensuring that AI systems support rather than hinder educational equity [MMS+21].

- **Employment:** AI-powered resume screening and hiring tools can reflect and reproduce workplace discrimination if trained on biased historical hiring data. For instance, algorithms may deprioritize resumes with minority-associated names or nontraditional educational paths [Che23]. Fair employment AI demands careful review of what constitutes "merit" and ensures equal opportunity.

- **Public Services:** Government systems increasingly use AI for eligibility screening, benefits administration, and fraud detection. These services impact vulnerable populations most directly. Fairness in public services is crucial to prevent administrative exclusion or harm, and must be designed with high accountability standards [SG21].

These diverse applications highlight the need for domain-specific approaches to fairness that consider the unique ethical challenges, stakeholder perspectives, and regulatory frameworks relevant to each context. Moreover, they emphasize that technical solutions alone are insufficient—organizational practices, institutional policies, and legal frameworks must also evolve to support fair AI deployment.

2.4 Fairness Algorithms and Metrics

The state-of-the-art in algorithmic fairness is built upon several foundational pillars that have evolved over time. These pillars form a comprehensive framework for addressing bias in machine learning systems through a systematic approach. First, researchers have developed various metrics to identify and quantify bias, enabling precise measurement of different fairness notions across demographic groups. Second, a rich ecosystem of mitigation algorithms has emerged, categorized by their position in the machine learning pipeline: pre-processing techniques that transform training data before model development; in-processing methods that incorporate fairness directly into the learning algorithms; and post-processing approaches that adjust model outputs to ensure fair predictions. Together, these components provide practitioners with a diverse toolkit to address fairness concerns across different contexts and applications, balancing the often competing goals of accuracy and equity. The following sections examine each of these pillars in greater detail, highlighting key approaches and their theoretical foundations.

2.4.1 Fairness Metrics

Fairness metrics are essential for quantifying bias in machine learning models. They provide a means to evaluate how well a model adheres to various fairness definitions, enabling practitioners to identify and address potential disparities. Commonly used metrics include:

Statistical Parity Difference (SPD)

Measures whether positive prediction rates are balanced across different demographic groups. Formally:

$$\text{SPD} = P(\hat{Y} = 1 \mid A = a) - P(\hat{Y} = 1 \mid A = b) \tag{2.1}$$

A value close to 0 indicates that the model assigns positive outcomes at equal rates to both groups. SPD is useful for detecting allocative bias—whether one group is favored in outcome distribution [DHP+11].

Disparate Impact (DI)

Quantifies the ratio of positive prediction rates between different demographic groups.

$$\text{DI} = \frac{P(\hat{Y} = 1 \mid A = a)}{P(\hat{Y} = 1 \mid A = b)} \tag{2.2}$$

Values below 0.8 (or above 1.25) are typically seen as indicative of potential disparate impact, according to the 80% rule. DI is widely used in legal and regulatory contexts as a clear and interpretable indicator [FFM+15].

Equal Opportunity Difference

Evaluates whether true positive rates are equal across protected groups.

$$\text{EOD} = P(\hat{Y} = 1 \mid Y = 1, A = a) - P(\hat{Y} = 1 \mid Y = 1, A = b) \tag{2.3}$$

A value near 0 implies that among the individuals who should receive a positive outcome ($Y = 1$), each group is equally likely to be correctly identified. EOD is crucial for ensuring equal quality of service [HPS16].

2.4.2 Pre-processing methods

A Pre-processing algorithm transform the training data to remove bias before model training. These techniques act directly on the data—by transforming features, learning new representations, or rebalancing weights—to reduce the influ-

ence of protected attributes and improve fairness regardless of the downstream model. Commonly used pre-processing methods includes:

Learning Fair Representations (LFR)

This technique learns a compressed data representation that is informative for prediction, reconstructs the input, and obscures group membership. A simplified view of the objective is:

$$L_{\text{fair}} = \text{prediction loss} + \text{reconstruction loss} + \text{fairness penalty} \tag{2.4}$$

More formally, the training minimizes a combined loss:

$$L_{\text{LFR}} = \lambda_y \, \ell(\hat{y}(z), y) + \lambda_x \, \|x - \hat{x}(z)\|^2 + \lambda_a \, D(z, A) \tag{2.5}$$

where:

- $z = f(x)$ is the latent representation,
- $\hat{y}(z)$ is the prediction output,
- $\hat{x}(z)$ is the reconstructed input,
- $D(z, A)$ penalizes statistical dependence between the representation and the protected attribute.

This enables training fair models on top of a transformed space where sensitive information is less present [ZWS+13].

Disparate Impact Remover

This method transforms features to reduce their dependency on the protected attribute. At a high level, the transformation pushes each feature toward a "fair" version that is equally distributed across groups:

$$x' = (1 - \alpha) \cdot x + \alpha \cdot x_{\text{fair}} \tag{2.6}$$

The formal transformation applies quantile mapping using **Cumulative Distribution Functions**:

$$x' = (1 - \alpha)\, x + \alpha\, F^{-1}(F_{x|A}(x)) \tag{2.7}$$

where $F_{x|A}$ is the **CDF** of the feature within group A, and F^{-1} is the global inverse **CDF**. The parameter $\alpha \in [0, 1]$controls how strongly features are adjusted. This technique reduces disparate impact by making feature distributions more similar across groups [FFM+15].

Reweighing

Modifies the importance of each training sample based on how frequent its label-group combination is. The goal is to simulate a balanced dataset where the label distribution is independent of group membership. Each sample with label y and group a receives weight:

$$w(y, a) = \frac{P(Y = y)}{P(Y = y \mid A = a)} \tag{2.8}$$

This rebalancing ensures that minority or underrepresented combinations receive more influence during training, encouraging models to make fairer decisions [KC12].

2.4.3 In-processing methods

An In-processing algorithm modifies the learning algorithm to incorporate fairness constraints during model training. These methods can be more effective than pre-processing, as they directly influence the model's decision-making process. Common in-processing techniques include:

FaUCI

Augments any stochastic-gradient–based learner with a generic penalised loss that can host any group-fairness metric (statistical parity difference, equalised odds, disparate impact, ...) and works with binary, categorical, or continuous sensitive attributes.

$$\min_{\boldsymbol{\theta}} \; \mathcal{L}\text{task}(\boldsymbol{\theta}) \; + \; \lambda\, \Phi\big(\widehat{f\boldsymbol{\theta}},\, A\big), \tag{2.9}$$

where

- $\mathcal{L}_{\text{task}}$ is the ordinary prediction loss (cross-entropy, MSE, ...),
- A is the protected attribute,
- Φ is the user-chosen fairness metric expressed as a differentiable surrogate,
- $\lambda > 0$ trades off accuracy and fairness.

Because Φ is a plug-in term, the same implementation can enforce several metrics simply by swapping Φ [MCCO24].

Adversarial Debiasing

Train a predictor and an adversary simultaneously: the predictor tries to forecast the label Y; the adversary tries to recover the protected attribute A from the predictor's internal representation (or logits). If the adversary fails, the representation is (approximately) independent of A.

$$\min_{\boldsymbol{\theta}} \Big[\mathcal{L}\text{task}(\boldsymbol{\theta}) \; - \; \lambda \max \boldsymbol{\phi} \mathcal{L}_{\text{adv}}\big(\boldsymbol{\theta}, \boldsymbol{\phi}\big)\Big], \tag{2.10}$$

where

- $\boldsymbol{\phi}$ are the adversary parameters;
- $\mathcal{L}$adv is a cross-entropy loss on predicting A.

At equilibrium the classifier is both accurate (small $\mathcal{L}$task) and uninformative about A (large adversary loss) [ZLM18].

Prejudice Remover

Make the prediction $\widehat{Y}$ statistically independent of the protected attribute by adding a mutual-information regulariser to the learner's objective.

$$\min_{\boldsymbol{\theta}} \; \mathcal{L}_{\text{task}}(\boldsymbol{\theta}) \; + \; \eta \, I\big(\widehat{Y}; A\big), \tag{2.11}$$

where

- $I(\widehat{Y}; A)$ is the mutual information between the model's output and the sensitive attribute (estimated with differentiable approximations);

- η controls the strength of the fairness penalty.

Driving $I(\widehat{Y}; A)$ toward zero removes indirect prejudice that might leak through correlated features [KAAS12].

Meta-Fair Classifier:

Solve classification and fairness as a single constrained optimisation problem. The user specifies (i) a target fairness metric $C(\boldsymbol{\theta})$ (e.g. statistical-rate gap, false-discovery-rate ratio) and (ii) an acceptable tolerance τ. **Primal form.**

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{task}}(\boldsymbol{\theta}) \\ &\text{s.t. } |C(\boldsymbol{\theta})| \leq \tau. \end{aligned} \tag{2.12}$$

Lagrangian reduction. In practice the constraint is dualised,

$$\min_{\boldsymbol{\theta}} \Big[\mathcal{L}_{\text{task}}(\boldsymbol{\theta}) + \lambda\, |C(\boldsymbol{\theta})|\Big], \tag{2.13}$$

and solved with an exponentiated-gradient meta-algorithm that adapts λ during learning, offering provable convergence guarantees for a large family of convex surrogates [CHKV20].

2.4.4 Post-processing methods

After a model has been fully trained, post-processing acts only on its outputs (scores or hard labels). This means you can retrofit fairness guarantees without touching the training data or the learning algorithm—very handy when you must comply with new regulations but cannot retrain. Below are three classic methods, each with a short plain-language description followed by the minimal mathematics that drives it.

Equalized Odds

Modifies predictions to equalize error rates across protected groups by applying group-specific thresholding or probabilistic flipping to model outputs. It ensures that both the **false positive rate (FPR)** and **false negative rate (FNR)** are equal across groups:

$$\Pr(\hat{Y} = 1 \mid Y = 0, A = a) = \Pr(\hat{Y} = 1 \mid Y = 0, A = a') \quad \text{(FPR equality)} \tag{2.14}$$

$$\Pr(\hat{Y} = 0 \mid Y = 1, A = a) = \Pr(\hat{Y} = 0 \mid Y = 1, A = a') \quad \text{(FNR equality)} \tag{2.15}$$

This is often achieved by solving a linear program that adjusts predictions post hoc while minimising accuracy loss [HPS16].

Calibrated Equalized Odds

Adjusts predictions while maintaining calibration—that is, the predicted probability $\hat{P}(Y = 1|X)$ must match the true frequency:

$$\Pr(Y = 1 \mid \hat{P} = p, A = a) = p \quad \text{for all groups } a. \tag{2.16}$$

It seeks a transformation of scores that retains this calibration while bringing **FPR** and **FNR** differences between groups below a small threshold ε. This allows a trade-off between fairness and probabilistic interpretability of outputs [PRW+17].

Reject Option Classification

Introduces a rejection option for uncertain predictions that might reflect bias. Specifically, in a band around the decision boundary (typically $p \in [0.5-\theta, 0.5+\theta]$), predictions are flipped in favour of the disadvantaged group:

$$\widetilde{Y} = \begin{cases} 1 & \text{if } A \text{ is disadvantaged and } |p - 0.5| \leq \theta, \\ 0 & \text{if } A \text{ is advantaged and } |p - 0.5| \leq \theta, \\ \text{original prediction} & \text{otherwise.} \end{cases} \tag{2.17}$$

This improves statistical parity by reducing unfair decisions near the boundary where the model is least confident [KKZ12].

2.5 Fairness Toolkits and Libraries Overview

IBM's *AI Fairness 360* (AIF360) and Microsoft's *Fairlearn* are two prominent open-source toolkits designed to operationalise fairness techniques in machine learning. Both provide extensive libraries of fairness metrics and bias-mitigation algorithms: AIF360 offers several detection metrics and nine pre-, in-, and post-processing mitigations [BDH+18], whereas Fairlearn complements a dashboard for diagnostic visualisation with reduction- and post-processing-based mitigations for both classification and regression [BDE+20]. Despite this breadth, recent empirical studies reveal substantial usability barriers [DNL+22]. Further hurdles arise from API and documentation design: inconsistent terminology (e.g., `CorrelationRemover` in Fairlearn versus `DisparateImpactRemover` in AIF360) and limited explanatory material oblige many users to copy code from toy examples rather than adapt algorithms systematically [DNL+22]. Finally, the libraries' research-centric architecture complicates integration into production pipelines, especially where access to protected attributes is constrained by privacy or compliance requirements [HWVD+19]. In short, while AIF360 and Fairlearn lower the entry barrier to technical fairness assessments, their current incarnations remain far from "plug-and-play." Addressing gaps in task coverage, guidance, and workflow integration is therefore essential if such tools are to fulfil their promise in real-world machine-learning practice.

2.6 Current Challenges in Operationalizing Fairness

In operationalizing algorithmic fairness, practitioners face multiple interrelated challenges. **First**, numerous formal definitions of fairness exist (e.g., demographic parity, equalized odds), and many of these criteria conflict or cannot be satisfied simultaneously [KMR16, XZ24]. This forces stakeholders to decide which defini-

tion best aligns with their context and values. **Second**, pursuing fairness often involves trade-offs with model accuracy or other performance measures, since fairness constraints can degrade predictive accuracy [CDPF+17]. **Third**, bias in data presents a major obstacle: historical datasets may reflect societal prejudices, and models can inadvertently use proxy features correlated with protected attributes, leading to residual unfairness even after interventions [BS16]. **Fourth**, fairness solutions must be tailored to each domain's legal and social context; for example, anti-discrimination laws in some sectors restrict the use of sensitive attributes and mandate specific fairness criteria, necessitating domain-specific approaches [VB17]. **Finally**, even when technical solutions are available, deploying them in real-world systems is difficult. Open-source fairness toolkits (e.g., AI Fairness 360 [BDH+18]) provide many metrics and mitigation algorithms, but practitioners often struggle with their usability and integration into existing workflows [HWVD+19, LS21]. These challenges highlight that achieving fairness in practice requires not only better algorithms, but also careful attention to context, data quality, and human-centered tool design.

Chapter 3

FairLib: Toolkit for Bias Analysis and Mitigation

FairLib is designed as a comprehensive, modular Python package for analyzing and mitigating bias in machine learning models, providing both theoretical foundations and practical tools for fairness-aware machine learning.

3.1 Core Architecture

The library follows a layered architectural pattern with clear separation of concerns, enabling flexible composition of fairness techniques while maintaining consistency across different algorithmic approaches. The architecture is built around five core components that work synergistically to provide a complete fairness-aware machine learning ecosystem.

3.1.1 Enhanced Data Foundation Layer

At the foundation lies the **FairLib DataFrame**, an extension of Pandas DataFrame that serves as the primary data container with native fairness awareness. This enhanced data structure provides explicit declaration and validation of protected attributes with automatic consistency checking, ensuring that sensitive information is properly tracked throughout the data processing pipeline. The DataFrame maintains centralized management of prediction targets with support for multi-

target scenarios, allowing researchers to work with complex prediction tasks while preserving fairness metadata.

The FairLib DataFrame provides native support for calculating fairness metrics directly on the dataset without requiring external dependencies, streamlining the evaluation workflow.

3.1.2 Unified Metrics Framework

The metrics layer provides a comprehensive framework for standardized evaluation of algorithmic fairness. This framework implements individual fairness metrics including **Statistical Parity Difference (SPD)**, **Disparate Impact (DI)**, and **Equality of Opportunity (EOO)**, each designed to capture different aspects of algorithmic bias. The framework maintains a consistent interface across all metrics, enabling seamless metric switching and comparison without requiring code restructuring.

The metrics system supports flexible output formats, providing both numerical results for automated analysis and detailed breakdowns by sensitive attribute groups for interpretability. These metrics are designed for dual use as standalone evaluation tools and as integrated components within algorithmic frameworks, particularly for in-processing techniques that incorporate fairness constraints directly into the optimization process.

3.1.3 Bias Mitigation Processing Layer

The processing layer encompasses two complementary paradigms for bias mitigation, each following consistent interface patterns that facilitate interoperability and ease of use.

The **pre-processing module** focuses on data transformation algorithms that remove bias before model training begins. This module implements a common `Preprocessor` base class with standardized `fit`, `transform` and `fit_transform` methodology, ensuring consistent behavior across different algorithms. The module includes Learning Fair Representations (LFR), Disparate Impact Remover, and Reweighing algorithms, each addressing different aspects of data bias. These

algorithms maintain data integrity while ensuring fairness properties in the transformed datasets, allowing practitioners to address bias at the data level before any model training occurs.

The **in-processing module** provides model training algorithms that incorporate fairness constraints during the learning process itself. This module implements a unified `Processor` interface with consistent `fit` and `predict` methods, mirroring familiar machine learning patterns. The module includes **Fairness Under Constrained Injection (FaUCI)**, **Adversarial Debiasing**, and **Prejudice Remover** algorithms, each offering different approaches to integrating fairness objectives into the optimization process. These algorithms seamlessly blend fairness considerations with predictive performance, enabling practitioners to find optimal trade-offs between accuracy and fairness.

3.1.4 Framework Integration Layer

The integration layer provides native support for popular machine learning ecosystems, ensuring that FairLib can be adopted within existing workflows without requiring fundamental architectural changes. PyTorch integration allows deep learning models to be wrapped and enhanced with fairness capabilities while preserving their original interfaces, enabling researchers to apply fairness techniques to complex neural architectures. Scikit-learn compatibility ensures that standard fit/predict/transform patterns enable drop-in replacement within existing ML pipelines, reducing the barrier to adoption.

3.1.5 Consistency and Interoperability

Cross-cutting architectural principles ensure system coherence across all components. The architecture enforces uniform interface patterns where all algorithms follow consistent method signatures and parameter conventions, reducing the learning curve for practitioners working with multiple techniques. Metadata preservation ensures that fairness-relevant information, including sensitive attributes and target specifications, is automatically maintained across all operations, preventing the loss of critical information during complex processing pipelines.

This architectural design enables users to seamlessly transition between different fairness approaches, combine multiple techniques, and integrate fairness considerations into existing machine learning workflows without requiring fundamental changes to their development practices.

3.2 Implementation Details

This section describes the core implementation details of the FairLib library's main components: Dataframe, Metrics, Pre-processing, and In-processing algorithms.

3.2.1 Metadata–Enriched DataFrame

Fairness assessments and mitigation strategies depend critically on two orthogonal pieces of information: (i) the decision variable y (*target*) and (ii) one or more protected or sensitive attributes S. In conventional pipelines, these identifiers are typically passed as arguments to each function invocation, a practice that is both verbose and error-prone. To streamline this process, metadata is embedded directly into the standard `pandas.DataFrame`.

Design choice. Although defining a new subclass of `DataFrame` might appear to be a clean and modular solution, it proves to be impractical in real-world scenarios. Many widely used libraries, such as `scikit-learn`, perform internal operations that downcast custom subclasses to the base `DataFrame`, thereby stripping away any additional functionality or metadata. To preserve compatibility while extending functionality, the approach adopted here relies on *monkey patching* [Car19]: descriptors are attached at runtime to the global `pandas.DataFrame` class, allowing the enriched interface to coexist seamlessly with existing tooling.

a) **Metadata descriptors.** The attributes `targets` and `sensitive` are implemented as descriptors that read from and write to keys in `df.attrs`. Because `attrs` is serialised by pandas I/O backends, the annotations persist across copies, slices, concatenations, and round-trip storage.

b) **Utility transforms.** Methods such as `unpack()` are provided to operate on the data while preserving and propagating the associated metadata, even when structural changes occur in the underlying columns.

c) **Metric facades.** Parameterless methods such as `disparate_impact()` and `statistical_parity_difference()` automatically retrieve y and S from the stored descriptors and delegate the computation to standard routines defined in `metrics`.

Implementation. Each descriptor is implemented as a specialised `property` that manipulates `df.attrs`. Metric wrappers include validation logic and invoke the appropriate computational backend using the metadata declared on the frame.

Fairness Metrics

The *metrics* module embodies a rigorously structured architectural strategy aimed at reconciling computational performance with usability and extensibility. Central to its design is a clear separation of concerns that distinguishes the mathematical core of fairness metric computation from their operational integration within the overarching data analysis framework.

The adoption of a dual-layer architecture is motivated by the heterogeneity of the intended user base and use cases. On the one hand, data scientists require efficient and portable computational primitives that integrate seamlessly with `NumPy`-centric analytical workflows. On the other hand, end-users and applied practitioners benefit from higher-level abstractions that interface intuitively with custom data containers. To this end, core metric functions such as `disparate_impact`, `statistical_parity_difference`, and `equality_of_opportunity` are implemented as independent, framework-agnostic operations over `NumPy` arrays. These low-level primitives are encapsulated within class-based wrappers (`DisparateImpact`, `StatisticalParityDifference`, etc.), which inherit from a common `Metric` base class, thereby enabling polymorphic usage and seamless integration with the DataFrame abstraction layer.

A particularly novel aspect of the design is the extension mechanism, which allows metric functionalities to be dynamically attached to `DataFrame` instances via

a descriptor-based pattern, implemented through `DataFrameExtensionFunction`. In practice, this design allows users to invoke metrics via an object-oriented syntax (e.g., df.statistical_parity_difference()), which enhances discoverability and ease of use. This syntactic convenience significantly enhances the discoverability and accessibility of fairness evaluation tools, aligning with established paradigms in exploratory data analysis and interactive computing.

The inherent multi-dimensionality of fairness evaluation outcomes necessitated the introduction of dedicated data structures. In particular, the `DomainDict` class provides a semantically rich mapping layer that supports flexible querying across sensitive attributes, prediction targets, and their value combinations. This enables domain-aware operations such as selective retrieval by attribute names, partial assignments, or intersectional groupings, which are essential for nuanced fairness analyses across diverse population strata.

Robust input validation and exception handling are implemented as first-class concerns within the library. The `check_and_setup` utility performs comprehensive precondition checks to ensure the dimensional alignment of sensitive attributes and outcome variables, issuing informative warnings in edge cases such as low group cardinality or missing categories. Furthermore, boundary conditions—such as metrics computed on empty demographic groups—are gracefully handled using `NaN` or infinite values, preserving the semantic consistency of results while avoiding runtime interruptions.

Finally, the metrics module supports multiple output modalities via the `as_dict` parameter, which enables users to toggle between matrix-based representations optimized for numerical pipelines and dictionary-based formats that enhance readability and facilitate downstream processing (e.g., visualization or reporting). This dual-mode output capability reflects a deliberate design decision to serve both programmatic and interpretive aspects of fairness analysis within responsible machine learning workflows.

Statistical Parity Difference

Statistical Parity Difference represents a fundamental group fairness metric that quantifies the disparity in favorable outcome probabilities between privileged and

unprivileged demographic groups. The theoretical foundation of this metric rests on the principle of demographic parity, which stipulates that a fair algorithmic system should produce equal acceptance rates across all protected groups, irrespective of the underlying ground truth distribution.

Mathematically, Statistical Parity Difference is formulated as:

$$\text{SPD} = P(\hat{Y} = 1|S = \text{privileged}) - P(\hat{Y} = 1|S = \text{unprivileged}) \tag{3.1}$$

where $\hat{Y}$ denotes the predicted outcome, S represents the sensitive attribute, and the probabilities are computed over the respective demographic subgroups.

The implementation in FairLib follows a dual-layer architectural pattern, providing both a low-level functional interface and a high-level object-oriented wrapper. The core computational logic iterates through all unique combinations of target and sensitive attribute values, calculating group-specific acceptance rates through array masking operations. For each sensitive group s, the algorithm computes the privileged rate as $\frac{|i:\hat{y}_i=t \wedge s_i=s|}{|i:s_i=s|}$ and the unprivileged rate as $\frac{|i:\hat{y}_i=t \wedge s_i\neq s|}{|i:s_i\neq s|}$, where t represents the target value. The implementation incorporates robust error handling for edge cases, including scenarios with empty subgroups, where infinite values are assigned to maintain mathematical consistency. The metric returns values in the range $[-1, 1]$, with zero indicating perfect fairness and values within $[-0.1, 0.1]$ generally considered acceptable in practical applications.

Disparate Impact

Disparate Impact constitutes a ratio-based fairness metric that emerged from legal frameworks, particularly the "80% rule" established in US anti-discrimination jurisprudence. This metric addresses the detection of indirect discrimination by measuring the relative likelihood of favorable outcomes between demographic groups, providing a multiplicative rather than additive perspective on fairness violations.

The mathematical formulation of Disparate Impact is expressed as:

$$\text{DI} = \frac{P(\hat{Y} = 1|S = \text{unprivileged})}{P(\hat{Y} = 1|S = \text{privileged})} \tag{3.2}$$

This ratio-based approach offers intuitive interpretation: values below 1.0 indicate

systematic disadvantage for the unprivileged group, while values above 1.0 suggest the opposite bias pattern.

The computational implementation mirrors the Statistical Parity Difference algorithm in its structural approach, utilizing the same group-wise probability estimation methodology. However, the critical distinction lies in the final aggregation step, where the unprivileged rate is divided by the privileged rate rather than subtracted. The implementation demonstrates careful attention to numerical stability, handling division-by-zero scenarios through appropriate infinite value assignments. The algorithm maintains consistency with legal standards by producing values in the range $[0, \infty)$, where perfect fairness corresponds to 1.0, and the legally acceptable threshold of 0.8 serves as a practical lower bound. Moreover, the FairLib implementation supports more complex scenarios, including multi-class targets and multiple sensitive attributes. It performs a comprehensive combinatorial evaluation across all target and attribute combinations, organizing the results in a structured DomainDict format. This design facilitates intersectional fairness analysis across diverse demographic categories.

3.2.2 Equality of Opportunity

Equality of Opportunity represents a more nuanced approach to algorithmic fairness that focuses specifically on ensuring equitable treatment among qualified individuals across demographic groups. This metric, grounded in the philosophical principle of meritocratic fairness, addresses scenarios where the primary concern is preventing discrimination against individuals who legitimately deserve favorable outcomes.

The theoretical framework centers on the concept of True Positive Rate (TPR) equalization, mathematically expressed as:

$$\text{EOO} = \text{TPR}_{\text{privileged}} - \text{TPR}_{\text{unprivileged}} \tag{3.3}$$

where $\text{TPR}_g = P(\hat{Y} = 1 | Y = 1, S = g)$ represents the true positive rate for demographic group g. This formulation ensures that qualified individuals (those with $Y = 1$) have equal probabilities of receiving positive predictions regardless of their protected attribute status.

The implementation architecture in FairLib demonstrates sophisticated handling of the three-way relationship between true labels, predictions, and sensitive attributes. The algorithm employs boolean masking operations to identify qualified individuals within each demographic group, subsequently computing group-specific true positive rates through vectorized operations. The algorithm creates a boolean mask for the privileged group (e.g., $sensitive\_column == privileged\_value$) and its complement for the unprivileged group ($sensitive\_column! = privileged\_value$). It then intersects these masks with the positive ground-truth mask ($target\_column == positive\_target$) to isolate qualified individuals in each group

The true positive rates are calculated as the ratio of correctly predicted positive cases to total positive cases within each group, with appropriate handling of empty sets through infinite value assignment. Unlike previous metrics, Equality of Opportunity requires an additional input—the ground truth labels (true outcomes)—making it suitable for post-hoc fairness evaluation after model training The metric produces values in the range $[-1, 1]$, where zero indicates perfect equality of opportunity, and the implementation supports flexible positive class specification through the positive_target parameter, accommodating diverse labeling conventions across different application domains.

3.2.3 Pre-Processing Algorithms

The pre-processing module of FairLib implements a suite of algorithms designed to mitigate bias in datasets before model training. These algorithms transform the input data to reduce the influence of sensitive attributes, ensuring that subsequent learning processes are less affected by inherent biases.

The preprocessing module uses a consistent interface architecture (inspired by scikit-learn) designed specifically to work with FairLib's enhanced DataFrame structure. The core interface consists of three primary methods: `fit()`, `transform()`, and `fit_transform()`. The `fit_transform()` method serves as the primary interface that all preprocessing algorithms must implement, combining parameter learning and data transformation in a single operation. For algorithms requiring separate fitting and transformation phases, the `fit()` method learns parameters from the training data without modifying it, while `transform()` applies the learned

transformation to new data. This separation enables efficient preprocessing of multiple datasets using the same learned parameters, which is particularly useful in train–test scenarios.

All methods are designed to work exclusively with FairLib's custom DataFrame, which extends the standard pandas DataFrame by preserving metadata for target and sensitive attributes. In particular, the DataFrame's `unpack()` method automatically extracts the feature matrix, target labels, and sensitive attribute indices, while built-in validation ensures consistent data structure across algorithms.

Reweighing Implementation

The Reweighing algorithm implementation follows the statistical approach proposed by Kamiran and Calders [KZC12], where instance weights are computed to achieve statistical parity across different demographic groups. The core implementation centers around a static method `_reweighing`, which calculates four distinct weight values corresponding to the combinations of privileged/unprivileged group status and favorable/unfavorable outcome.

The algorithm first validates the input DataFrame structure and identifies the target column. It then creates boolean masks for favorable and unfavorable outcomes using a private utility function `get_favorable_unfavorable_masks`. For each combination of sensitive attribute value and target outcome, a weight is computed using the ratio formula

$$w \;=\; \frac{n_{\text{outcome}} \times n_{\text{group}}}{n_{\text{total}} \times n_{\text{joint}}}\,,$$

where n_{outcome} is the total number of instances with the specified outcome (favorable or unfavorable) in the entire dataset, n_{group} is the number of instances in the given demographic group (privileged or unprivileged), n_{total} is the total number of instances in the dataset, and n_{joint} is the number of instances that belong to both that group and that outcome category. The implementation also includes a specialized variant called `ReweighingWithMean`. This variant handles cases with multiple sensitive attributes by computing weights for each attribute independently and then averaging them, providing a more nuanced approach for intersectional fairness scenarios. The final transformed DataFrame includes a new `weights` col-

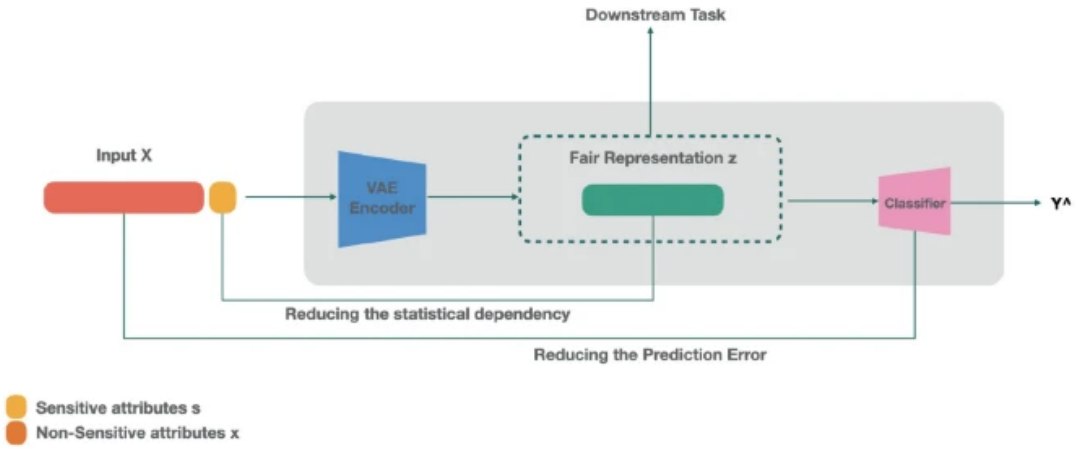

Figure 3.1: Architecture of the Learning Fair Representations (LFR) algorithm. The encoder compresses input features into a fair representation and the classifier predicts the target label from the fair representation [TYJY+24].

umn that can be directly used with scikit-learn classifiers via the `sample_weight` parameter during training.

Learning Fair Representations Implementation

The Learning Fair Representations (LFR) algorithm is implemented as a deep neural network consisting of three interconnected components—an encoder, a decoder, and a classifier—constructed with PyTorch. The encoder network transforms the input features into a lower-dimensional "fair" representation, *as illustrated in* Figure 3.1, using a three-layer fully connected architecture (input → 64 → 32 → latent_dim) with ReLU activations; this 64–32 configuration is simply the library's default and can be replaced by defining custom `Encoder`, `Decoder`, and `Classifier` modules that respect the required layer dimensions, making the entire LFR pipeline fully customisable. The decoder then attempts to reconstruct the original feature space from this representation using the symmetric architecture (latent_dim → 32 → 64 → output_dim), which can likewise be adapted to alternative depths or widths.

Finally, a classifier component predicts the target label from the fair representation using a two-layer network that culminates in a sigmoid output unit for

binary classification; like the encoder and decoder, this classifier can be freely re-implemented to suit specific tasks.

The training process optimizes a multi-objective loss function

$$L_{\text{total}} \;=\; \alpha_z\, L_z \;+\; \alpha_x\, L_x \;+\; \alpha_y\, L_y\,,$$

where L_z is a fairness loss defined as the sum of squared differences between the mean representations of the protected and unprotected groups, L_x is a reconstruction loss (measured as the mean squared error between the original features and their reconstructed counterparts), and L_y is the binary cross-entropy loss for predicting the target. All three networks (encoder, decoder, and classifier) are trained jointly using the Adam optimizer, and input features are standardized via scikit-learn's `StandardScaler` prior to training. During each forward pass, the encoder produces a fair representation, which the decoder uses to reconstruct the inputs and the classifier uses to predict the outcome. This joint training procedure ensures that the learned representations preserve predictive utility while minimizing sensitive attribute information. In particular, the fairness loss term L_z serves as a statistical parity regularizer, encouraging the encoded representations to be independent of the sensitive attribute.

Disparate Impact Remover Implementation

The Disparate Impact Remover algorithm implements the fairness-aware "data repair" technique proposed by Feldman et al. [FFM+15]. This technique transforms feature distributions to achieve statistical parity across demographic groups through quantile-based distribution alignment.

In the implementation, the first step is to compute the empirical cumulative distribution function (CDF) for each feature within each sensitive attribute group. A helper function `_make_cdf` is used for this purpose; it returns a callable CDF for a given feature by sorting the feature values and enabling binary search lookup of quantile positions. During the fitting phase, the algorithm then constructs a quantile "map" for each feature as follows: it groups the data by sensitive attribute, establishes a common grid of quantile values (e.g., using `np.linspace` from 0 to 1), and computes the median feature value across all groups at each quantile level.

These median values across quantiles form a reference distribution for the feature that represents a merged (fair) version of the original group-specific distributions.

In the transformation phase, each feature value x is adjusted via this reference distribution. For a given data point, the algorithm first determines the quantile rank of x within the CDF of its own sensitive group. It then uses linear interpolation (via `np.interp`) to map that quantile to the corresponding value in the reference (median) distribution. The resulting value x_{median} is the fully "repaired" version of x (i.e., the value x would have in the median distribution). Finally, the algorithm computes the output as a weighted combination of the original and repaired values:

$$x_{\text{repaired}} \;=\; (1-\lambda)\, x_{\text{original}} \;+\; \lambda\, x_{\text{median}}$$

Here λ is the repair level parameter that controls the trade-off between fairness and data fidelity.

This approach makes the marginal distributions of each feature much more similar across different demographic groups, thereby reducing disparate impact, while still preserving some of the original data characteristics. The repair level λ allows practitioners to balance between achieving complete distributional fairness ($\lambda = 1$, using only the median-aligned value) and preserving the original feature values ($\lambda = 0$, no adjustment). Additionally, the implementation preserves metadata consistency by keeping the same column names and retaining sensitive attribute information in the transformed DataFrame.

3.2.4 In-Processing Algorithms

The in-processing module of FairLib implements advanced fairness-aware machine learning algorithms by integrating fairness constraints directly into model training. This approach enables the simultaneous optimization of predictive accuracy and fairness objectives, allowing practitioners to achieve better trade-offs between these often competing goals. The implementation follows a modular design pattern, and all algorithms inherit from a common `Processor` base class that defines standard `fit()` and `predict()` methods. This abstraction ensures a consistent interface across different fairness algorithms and compatibility with FairLib's DataFrame-based data handling system, making it easy to interchange in-processing mitigation

methods within the library.

Adversarial Debiasing Implementation

Building on the original Adversarial Debiasing framework of [ZLM18], we adopt the architectural refinement proposed by [BCZC17] in which the *adversary consumes an intermediate hidden representation* rather than the predictor's final logits. This shift from an *output-based* to a *representation-based* adversarial signal encourages the predictor to remove sensitive information *earlier* in the network and has been shown to yield stronger fairness guarantees while retaining predictive power.

The algorithm is implemented as a dual-network architecture in PyTorch, consisting of a predictor network and an adversarial network connected via a custom gradient-reversal layer. At the heart of the implementation is the `GradientReversal` function, a custom autograd function that forwards inputs unchanged in the forward pass but in the backward pass multiplies the gradients by $-\lambda_{\text{adv}}$, thereby reversing and scaling them by the factor λ_{adv}.

The predictor network is a three-layer feed-forward model: each hidden layer applies batch normalization (`BatchNorm1d`) before a linear transformation, followed by a ReLU activation and dropout (default drop rate 0.3) for regularization. Crucially, it returns both the task logits and the hidden representation $\mathbf{h}$ that serves as input to the adversary. Both the predictor and the adversary can be replaced with custom `nn.Module`s, provided they share a compatible interface: (i) the predictor must expose a representation whose dimensionality matches the adversary's input size, and (ii) both modules must return logits suitable for their respective loss functions.

An example of this architecture is shown in Figure 3.2.

This flexibility allows practitioners to experiment with deeper, convolutional, or transformer-based architectures without changing the training loop. The adversarial network mirrors the predictor's architecture (including batch normalization and dropout) but employs LeakyReLU activations in its hidden layers. During training, an alternating optimization strategy is employed: in each iteration, the adversary is first updated for a fixed number of steps (`adv_steps`) to better predict the sensitive attribute from $\mathbf{h}$ (with predictor parameters frozen), and then

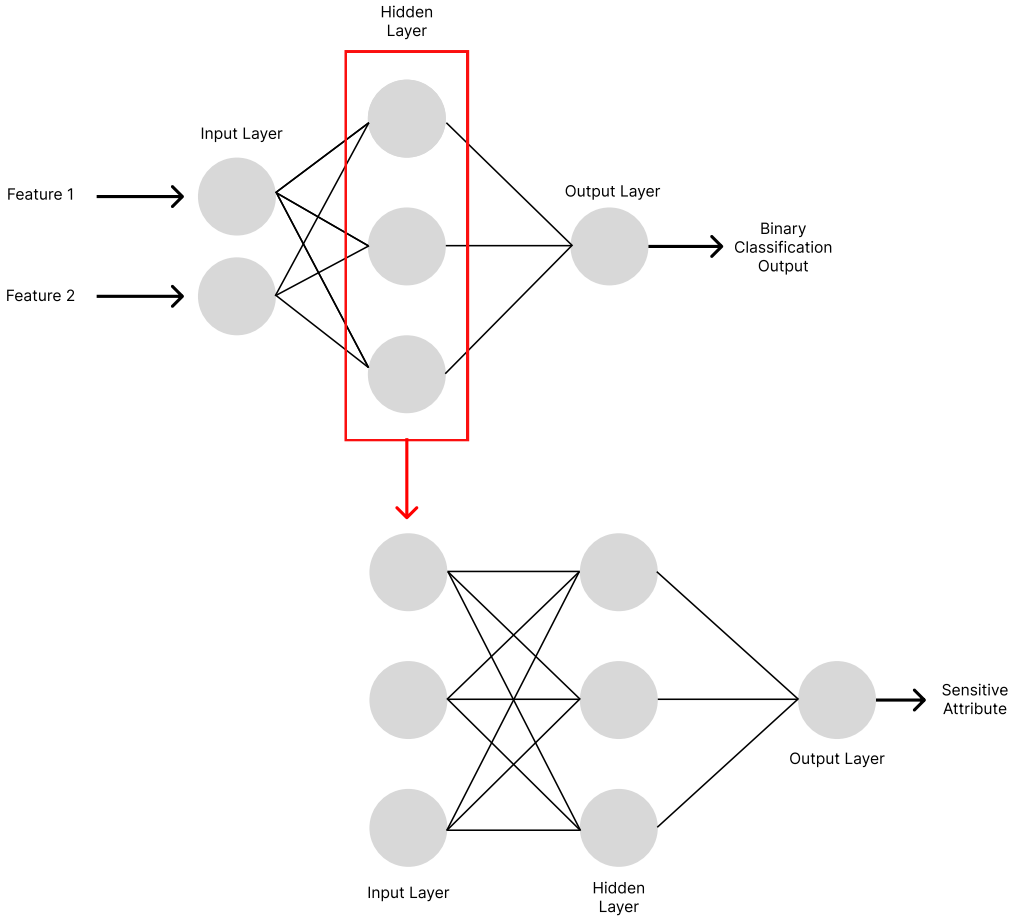

Figure 3.2: Adversarial Debiasing architecture: a predictor network learns to make predictions while an adversarial network attempts to predict the sensitive attribute from the predictor's *internal* representation, following the hidden-layer strategy.

the predictor is updated (via the gradient-reversal mechanism) to minimize its primary prediction loss while *maximizing* the adversary's loss. This effectively forces the predictor to learn representations that confound the adversary, thereby promoting fairness at the representation level. To stabilize training, gradient clipping (max-norm 1.0) is applied to both networks, and optimization is performed with AdamW (including weight decay). Hyperparameters such as learning rates, batch size, the number of adversary steps, and the critical λ_{adv} parameter controlling the fairness–accuracy trade-off remain fully configurable.

FaUCI Implementation

The **FaUCI** (Fairness Under Constrained Injection) algorithm incorporates fairness metrics directly into the loss function as regularization terms. It is implemented through a modular loss architecture that seamlessly integrates with standard PyTorch training workflows. In particular, a hierarchical loss design is used: a base loss (e.g. cross-entropy for classification) is encapsulated by a `BaseLoss` class, and a `PenalizedLoss` subclass extends this by adding a fairness regularization term. The total loss at each training step is computed as a weighted sum of the base loss and the fairness penalty:

$$L_{\text{total}} \;=\; (1 - w)\, L_{\text{base}} \;+\; w\, L_{\text{fairness}}, \tag{3.4}$$

where w is the regularization weight that controls the trade-off between accuracy and fairness. The fairness penalty leverages a pluggable metric system (implemented in `_torch_metrics.py`) supporting metrics such as Statistical Parity Difference (SPD) and Disparate Impact (DI). These metrics are computed on each mini-batch during training to provide an on-the-fly estimate of unfairness. For example, the SPD metric is calculated as the absolute difference in positive outcome probability between the protected and unprotected groups, while the DI metric is the ratio of these probabilities. Both metrics are efficiently computed using the model's predictions for the current mini-batch, with a small epsilon (e.g. 10^{-7}) added to probability estimates to ensure numerical stability (avoiding division by zero or undefined log values). The `FaUCI` class serves as the main processor for this algorithm, wrapping an arbitrary PyTorch model and managing the

training loop via a flexible interface. It allows users to specify custom optimizers, base loss functions, and fairness metrics (along with the regularization weight w). During training, the sensitive attribute values for each batch are passed into the loss computation (via a property injection mechanism) so that the fairness metric can be evaluated for that batch's predictions. The implementation supports a range of hyperparameters including the number of epochs, batch size, and the regularization weight w, making FaUCI a versatile framework for exploring different fairness–accuracy trade-offs.

Prejudice Remover Implementation

The Prejudice Remover algorithm is implemented via a mutual-information-based regularization framework, which explicitly penalizes the model for encoding statistical dependence between its predictions and the sensitive attribute. The core of this approach is a `PrejudiceRemoverLoss` that augments the standard prediction loss with a mutual information (MI) penalty term:

$$L_{\text{total}} = L_{\text{base}} \; + \; \eta \cdot MI(\hat{Y}, S) \tag{3.5}$$

where η is a hyperparameter controlling the strength of the fairness regularization. Here $MI(\hat{Y}, S)$ measures the mutual information between the model's output $\hat{Y}$ and the sensitive attribute S, which is zero if the predictions are independent of S. In practice, this mutual information is estimated on each mini-batch by computing the joint distribution $P(\hat{Y}, S)$ and the marginal distributions $P(\hat{Y})$ and $P(S)$ from the batch's predictions and labels. The implementation calculates group-conditional probabilities (such as $P(\hat{Y} = 1 \mid S = 0)$ and $P(\hat{Y} = 1 \mid S = 1)$) by averaging the model's predicted probabilities for the positive class within each sensitive group in the batch. Using these, it obtains estimates of $P(\hat{Y} = 1, S = 0)$, $P(\hat{Y} = 1, S = 1)$, etc., for all four combinations of binary outcome and sensitive value. To prevent numerical issues (e.g. $\log(0)$), a small epsilon (10^{-8}) is added to each probability estimate. The mutual information penalty is then computed as the Kullback–Leibler divergence between the joint and product-of-marginals

distributions:

$$MI(\hat{Y}, S) \approx \sum_{y\in\{0,1\}} \sum_{s\in\{0,1\}} P(\hat{Y} = y,\ S = s)\ \log \frac{P(\hat{Y} = y,\ S = s)}{P(\hat{Y} = y)\, P(S = s)} \tag{3.6}$$

which corresponds to summing over the four possible (y, s) combinations in the binary case. The `PrejudiceRemover` class integrates this loss into a PyTorch training pipeline, allowing flexible model architectures via dependency injection and automatically converting FairLib's DataFrame inputs into tensors for training. The training routine ensures proper gradient flow through the mutual information term (so that the model's predictions are influenced by the penalty), and it supports both binary cross-entropy and mean squared error as the base loss (depending on the prediction task). To help practitioners monitor progress, the implementation includes extensive logging of metrics, tracking the model's predictive performance as well as fairness indicators (such as the estimated MI or group outcome differences) at each epoch. This mutual-information regularization approach (originally proposed by Kamishima et al. [KAAS12]) encourages the model to produce fairer predictions by explicitly discouraging any measurable dependency between $\hat{Y}$ and S.

Chapter 4

Evaluation and Results

4.1 Introduction

This chapter presents an empirical evaluation of the proposed Python library for algorithmic fairness. We analyse its behaviour along four complementary axes: *verbosity of the API*, *ease of use*, *seamless integration within existing machine–learning pipelines*, and *quantitative impact on fairness metrics*. The goal is to provide a comprehensive view of the trade–offs introduced by the library and to validate its practical utility.

4.2 Verbosity Analysis

In this section we quantify the amount of code required to express common fairness workflows using the library such as computing fairness metrics, applying preprocessing algorithms, and integrating in-processing techniques.

4.2.1 Metrics

The `FairLib` library provides a streamlined interface for computing a wide variety of fairness metrics, allowing for efficient integration into experimental workflows. In particular, it supports the evaluation of group fairness criteria with minimal coding effort. The code snippet 4.1 illustrates how to compute key metrics such

as Statistical Parity Difference, Disparate Impact, and Equality of Opportunity, which are commonly used to assess the presence of bias in classification models.

```
import fairlib as fl

dataset = load_example_dataset()
fairlib_dataframe = fl.DataFrame(dataset)

# Setting the target feature and sensitive attributes
fairlib_dataframe.targets = 'income'
fairlib_dataframe.sensitive = ['sex', 'race']

# Calculating fairness metrics
fairlib_dataframe.statistical_parity_difference()
fairlib_dataframe.disparate_impact()
fairlib_dataframe.equality_of_opportunity()
```

Listing 4.1: Example Python script for computing fairness metrics

Only twelve lines-eight if we omit comments-are needed to progress from raw tabular data to a structured fairness report. Each metric call returns a dictionary. The library automatically infers the target and sensitive attributes from the DataFrame metadata, eliminating the need for explicit parameter passing. This significantly reduces cognitive load and potential errors in specifying these identifiers.

4.2.2 Pre-processing Algorithms

The library's pre-processing algorithms are designed to be easily applied to datasets with minimal boilerplate code. The code snippet 4.2 demonstrates how to apply the Reweighing algorithm, which adjusts instance weights to mitigate bias in classification tasks. The example shows how to instantiate the algorithm, fit it to the data, and transform the dataset in just a few lines of code.

```
import fairlib as fl

dataset = load_example_dataset()
fairlib_dataframe = fl.DataFrame(dataset)
```

```
5
6  # Setting the target feature and sensitive attributes
7  fairlib_dataframe.targets = 'income'
8  fairlib_dataframe.sensitive = ['sex', 'race']
9
10 reweighing = fl.Reweighing()
11 reweighed_df = reweighing.fit_transform(fairlib_dataframe)
12 # The reweighed DataFrame now contains adjusted instance
       weights to mitigate bias in classification tasks.
```

Listing 4.2: Example Python script for applying the Reweighing pre-processing algorithm

Support for **multiple sensitive attributes** enables intersectional fairness analysis within the library's framework. Instance weights are automatically computed and appended to the dataset, allowing for seamless integration with scikit-learn classifiers via the `sample_weight` parameter.

The Reweighing algorithm is particularly useful for addressing disparities in classification outcomes by adjusting the weights of instances based on their sensitive attributes. The library's design allows users to easily switch between different pre-processing techniques, such as Learning Fair Representations or Disparate Impact Remover, with minimal code changes.

4.2.3 In-processing Algorithms

The in-processing algorithms in FairLib are designed to be easily integrated into existing machine learning workflows. The code snippet 4.3 illustrates how to apply the Adversarial Debiasing algorithm, which trains a model to make predictions while simultaneously minimizing bias with respect to sensitive attributes.

```
1  import fairlib as fl
2  from sklearn.model_selection import train_test_split
3
4  EPOCHS = ...
5
6  X = load_feature_dataset()
7  y = load_target_dataset()
```

```
8
9  X_train, X_test, y_train, y_test = train_test_split(
10     X, y, test_size=0.3, random_state=42
11 )
12
13 fair_model = fl.AdversarialDebiasing(
14     input_dim=X_train.shape[1],
15     hidden_dim=8,
16     output_dim=1,
17     sensitive_dim=1,
18     lambda_adv=1, # Fairness intervention
19 )
20
21 fair_model.fit(X_train, y_train, num_epochs=EPOCHS)
22 y_pred = fair_model.predict(X_test)
```

Listing 4.3: Example Python script for applying the Adversarial Debiasing in-processing algorithm

The example shows how to instantiate the Adversarial Debiasing processor, fit it to the training data, and make predictions with just a few lines of code. The library's design allows users to easily switch between different in-processing techniques, such as FaUCI or Prejudice Remover, with minimal code changes.

4.3 Integration into Existing Pipelines

This section explores how the library can be embedded into pre–existing scikit–learn / PyTorch pipelines with minimal refactoring. We discuss interoperability with data preprocessing stages, hyperparameter optimisation loops, and model persistence mechanisms.

4.3.1 Preprocessing Integration

The library's preprocessing algorithms can be seamlessly integrated into existing data pipelines. The code snippet 4.4 shows a basic example of binary classification, without the application of any fairness algorithms.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

X = load_features_dataset()
y = load_target_dataset()

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)

classifier = LogisticRegression()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuratezza sul test set: {accuracy:.4f}")
```

Listing 4.4: Example Python script of a classification process without application of fairness

In contrast, the code snippet 4.5 demonstrates how to apply the LFR pre-processing algorithm to the same dataset, adjusting instance weights based on sensitive attributes.

```python
import fairlib as fl
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

X = load_features_dataset()
y = load_target_dataset()

X.sensitive = 'gender'  # Example sensitive attribute
EPOCHS = ...
BATCH_SIZE = ...

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
```

```
14
15 layers_shape = X_train.shape[1]
16
17 lfr = fl.LFR(
18     input_dim=layers_shape,
19     latent_dim=8,
20     output_dim=layers_shape,
21     alpha_z=1.0,
22     alpha_x=1.0,
23     alpha_y=1.0,
24 )
25
26 lfr.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE)
27
28 X_train_transformed = lfr.transform(X_train)
29 X_test_transformed = lfr.transform(X_test)
30
31 classifier = LogisticRegression()
32 classifier.fit(X_train_transformed, y_train)
33
34 y_pred = classifier.predict(X_test_transformed)
35 accuracy = accuracy_score(y_test, y_pred)
36
37 print(f"Accuratezza sul test set: {accuracy:.4f}")
```

Listing 4.5: Example Python script of a classification process with application of fairness

This example illustrates how the library's preprocessing algorithms can be easily integrated into existing data pipelines, allowing users to apply fairness techniques without significant changes to their codebase. The preprocessing step is performed before model training, ensuring that the model learns from a fair representation of the data.

4.3.2 In-processing Integration

The library's in-processing algorithms can be integrated into existing model training workflows with minimal code changes. The code snippet 4.6 shows a basic

example of training a PyTorch model without applying any fairness techniques.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
import torch.optim as optim

EPOCHS = ...

X = load_features_dataset()  # torch.tensor (N, D)
y = load_target_dataset()    # torch.tensor (N,)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

class SimpleNN(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear = nn.Linear(input_dim, 2)

    def forward(self, x):
        return self.linear(x)

model = SimpleNN(X.shape[1])
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

for _ in range(EPOCHS):
    optimizer.zero_grad()
    output = model(X_train)
    loss = loss_fn(output, y_train)
    loss.backward()
    optimizer.step()

with torch.no_grad():
    preds = torch.argmax(model(X_test), dim=1)
    acc = accuracy_score(y_test.numpy(), preds.numpy())
    print(f"Accuratezza sul test set: {acc:.4f}")
```

Listing 4.6: Example Python script of a PyTorch model training process without application of fairness

In contrast, the code snippet 4.7 demonstrates how to apply FaUCI in-processing algorithm during model training.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
import torch.optim as optim

import fairlib as fl

EPOCHS = ...
BATCH_SIZE = ...

X = load_features_dataset()  # torch.tensor (N, D)
y = load_target_dataset()    # torch.tensor (N,)

X.sensitive = 'gender'

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)


class SimpleNN(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear = nn.Linear(input_dim, 2)

    def forward(self, x):
        return self.linear(x)


model = SimpleNN(X.shape[1])

```

```
33
34 fauci_model = fl.Fauci(
35     torchModel=model,
36     optimizer=optim.Adam(
37         model.parameters(
38             model.parameters(),
39             lr=0.001
40             )
41         ),
42     loss=nn.CrossEntropyLoss(),
43     fairness_regularization="spd", # or "di" or others
       supported by _torch_metrics.get
44     regularization_weight=0.5, # Example weight, adjust as
       needed
45 )
46
47 fauci_model.fit(X, y, epochs=EPOCHS, batch_size=BATCH_SIZE)
48
49 with torch.no_grad():
50     preds = torch.argmax(fauci_model.predict(X_test), dim=1)
51     acc = accuracy_score(y_test.numpy(), preds.numpy())
52     print(f"Accuratezza sul test set: {acc:.4f}")
```

Listing 4.7: Example Python script of a PyTorch model training process with application of fairness

This example illustrates how the library's in-processing algorithms can be easily integrated into existing model training workflows, allowing users to apply fairness techniques without significant changes to their codebase. The in-processing step is performed during model training, ensuring that the model learns to make fair predictions while optimizing for accuracy.

4.4 Fairness Metrics: Before and After Mitigation

4.4.1 Experimental Setup

We describe the datasets, target variables, protected attributes, and evaluation protocol employed to measure fairness both prior to and after applying the library's pre–processing and in–processing algorithms. For this evaluation, we use the Adult Income dataset [adu], which contains demographic information about individuals and whether they earn more than \$50,000 per year. The sensitive attribute is `sex`, and the target variable is `income` (binary classification: income $\leq$ \$50,000 vs. income $>$ \$50,000). The dataset is split into training and test sets, with the training set used for model fitting and the test set for evaluation.

The evaluation protocol, for **pre-processing algorithms**, consists of the following steps:

1. Compute fairness metrics (Statistical Parity Difference, Disparate Impact) on the original dataset.
2. Apply pre-processing algorithms (Reweighing, Learning Fair Representations, Disparate Impact Remover) to the training set.
3. Train a classifier (logistic regression) on the transformed training set.
4. Compute fairness metrics on the test set after applying the pre-processing algorithms.

For **in-processing algorithms**, the evaluation protocol consists of:

1. Compute fairness metrics (Statistical Parity Difference, Disparate Impact) on the original dataset.
2. Train a classifier (logistic regression) with in-processing algorithms (Adversarial Debiasing, FaUCI, Prejudice Remover) integrated into the training loop.
3. Compute fairness metrics on the test set after training with in-processing algorithms.

4.4.2 Results Summary

A concise presentation of key numerical results, focusing on disparity measures (e.g., Statistical Parity Difference, Disparate Impact) and accuracy.

Pre-processing Algorithms Results

The results of applying pre-processing algorithms to the Adult Income dataset are summarised in 3 figures. Figure 4.1 shows the accuracy of the baseline model and the models trained after applying pre-processing algorithms. The accuracy is computed on the test set of the Adult Income dataset. Figure 4.2 shows the Statistical Parity Difference (SPD) for the baseline model and the models trained after applying pre-processing algorithms. The SPD is computed on the test set of the Adult Income dataset. Figure 4.3 shows the Disparate Impact (DI) for the baseline model and the models trained after applying pre-processing algorithms. The DI is computed on the test set of the Adult Income dataset.

From the results obtained by applying the algorithms in the pre-processing module, we observe that all algorithms successfully reduce the Statistical Parity Difference (SPD) and Disparate Impact (DI) metrics compared to the baseline model.

In-processing Algorithms Results

The results of applying in-processing algorithms to the Adult Income dataset are summarised in 3 figures. Figure **??** shows the accuracy of the baseline model and the models trained after applying in-processing algorithms. The accuracy is computed on the test set of the Adult Income dataset. Figure **??** shows the Statistical Parity Difference (SPD) for the baseline model and the models trained after applying in-processing algorithms. The SPD is computed on the test set of the Adult Income dataset. Figure **??** shows the Disparate Impact (DI) for the baseline model and the models trained after applying in-processing algorithms. The DI is computed on the test set of the Adult Income dataset.

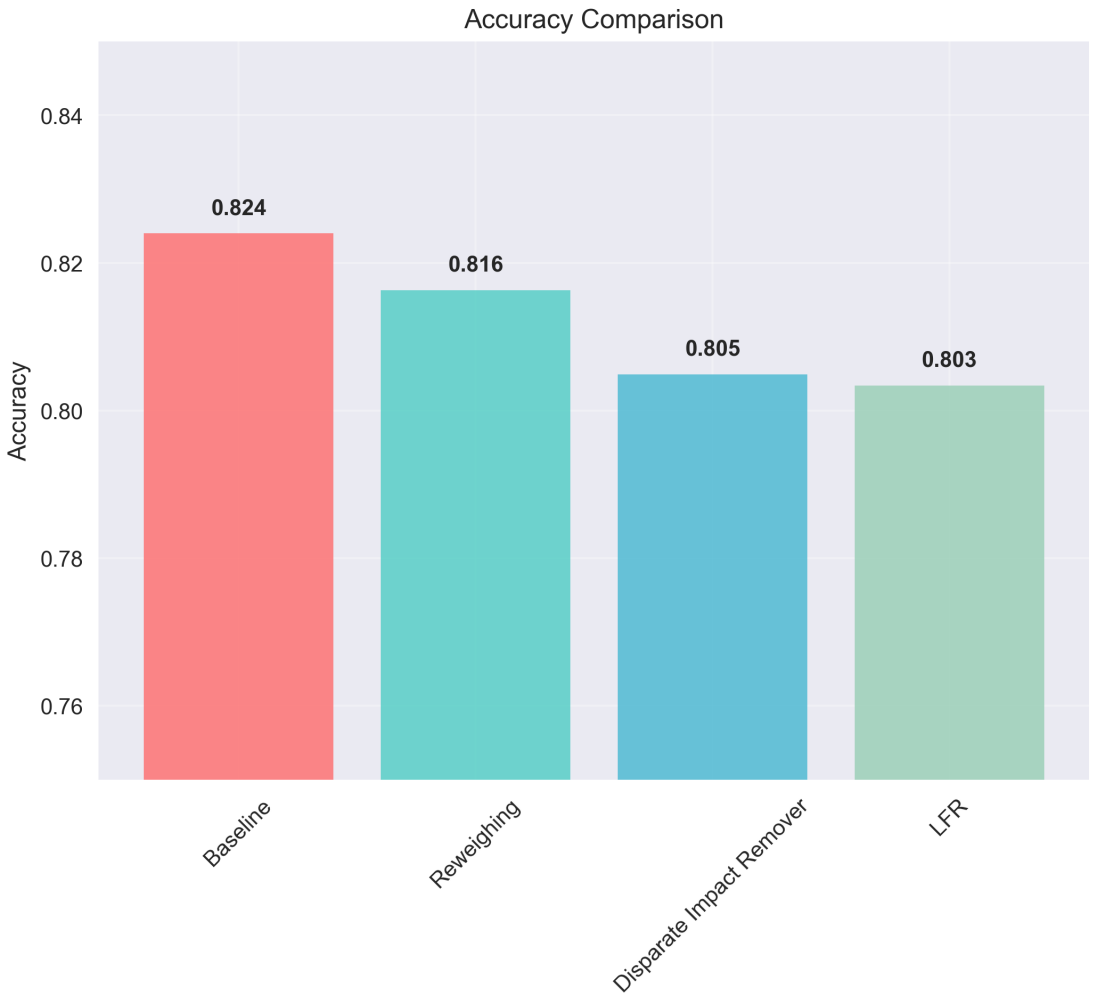

Figure 4.1: Accuracy score for the baseline model and the models trained after applying pre-processing algorithms. The accuracy is computed on the test set of the Adult Income dataset.

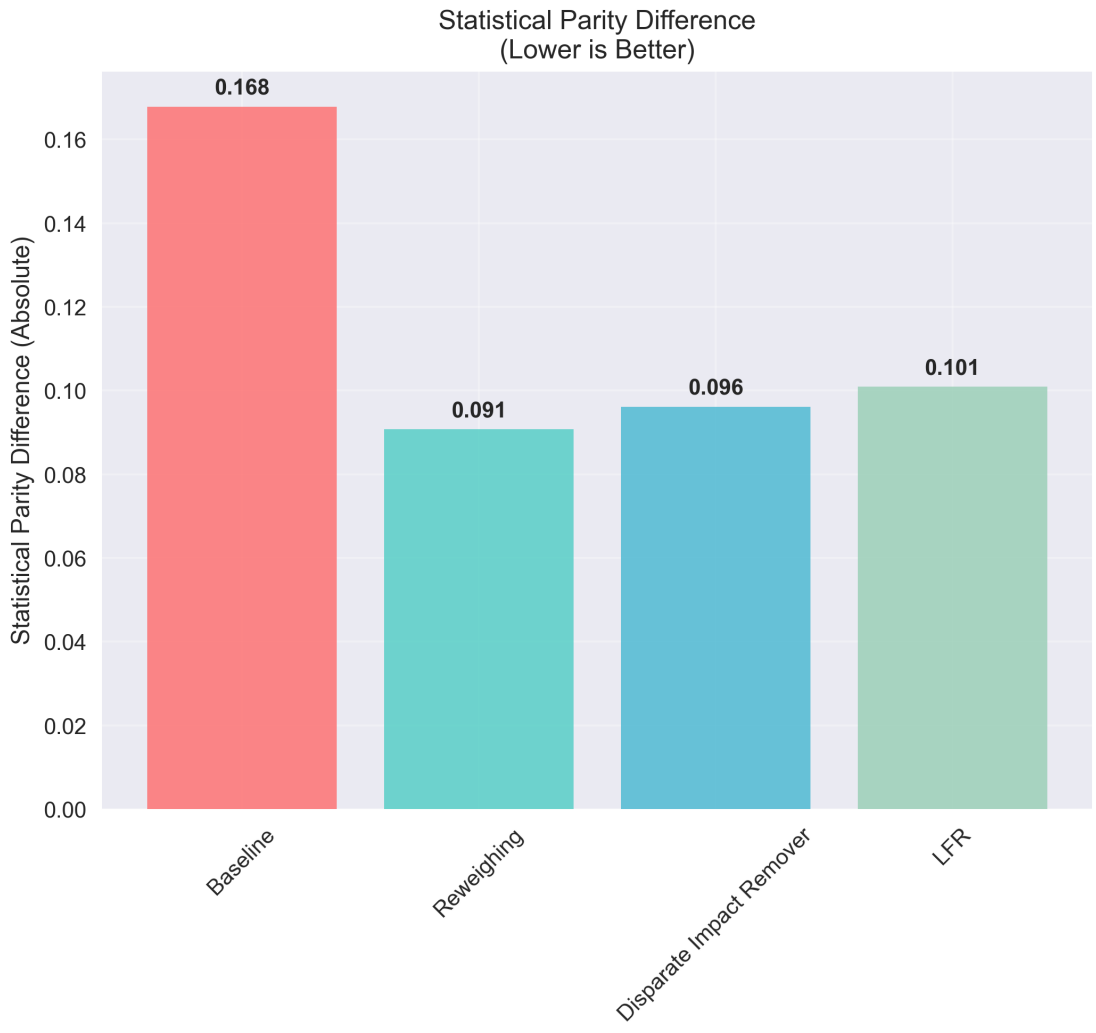

Figure 4.2: Statistical Parity Difference (SPD) for the baseline model and the models trained after applying pre-processing algorithms. The SPD is computed on the test set of the Adult Income dataset.

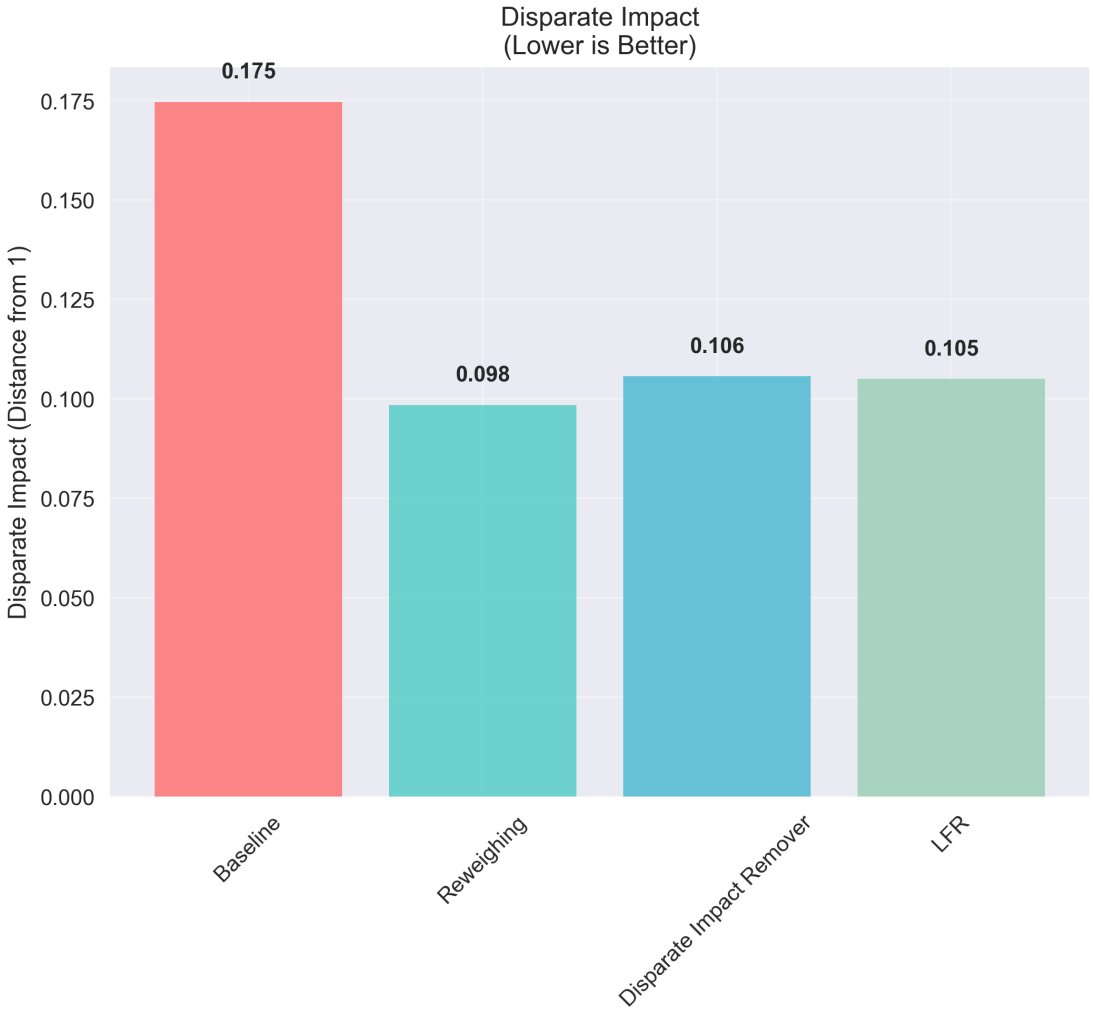

Figure 4.3: Disparate Impact (DI) for the baseline model and the models trained after applying pre-processing algorithms. The DI is computed on the test set of the Adult Income dataset.

4.4.3 Discussion

The results demonstrate that the library's pre-processing and in-processing algorithms effectively reduce bias in classification tasks while maintaining competitive accuracy. The Statistical Parity Difference (SPD) and Disparate Impact (DI) metrics show significant improvements compared to the baseline model, indicating that the algorithms successfully mitigate disparities in outcomes across sensitive groups.

4.5 Conclusion

This chapter has provided a comprehensive evaluation of the FairLib library, focusing on its usability, integration capabilities, and effectiveness in mitigating bias in machine learning models. The results demonstrate that FairLib offers a user-friendly interface for implementing fairness algorithms, with minimal code required to achieve significant improvements in fairness metrics. The library's design allows for seamless integration into existing machine learning workflows, making it a valuable tool for practitioners seeking to address algorithmic bias. The empirical results show that both pre-processing and in-processing algorithms can effectively reduce disparities in classification outcomes while maintaining competitive accuracy. The library's modular architecture and consistent interface make it easy to experiment with different fairness techniques, enabling users to tailor their approaches to specific datasets and fairness objectives.

Bibliography

[adu]

[BCG21] Teresa Bono, Karen Croxson, and Adam Giles. Algorithmic fairness in credit scoring. *Oxford Review of Economic Policy*, 37(3):585–617, 09 2021.

[BCZC17] Alex Beutel, Jilin Chen, Zhe Zhao, and Ed H. Chi. Data decisions and theoretical implications when adversarially learning fair representations, 2017.

[BDE+20] Sarah Bird, Miro Dudík, Richard Edgar, Brandon Horn, Roman Lutz, Vanessa Milan, Mehrnoosh Sameki, Hanna Wallach, and Kathleen Walker. Fairlearn: A toolkit for assessing and improving fairness in ai. Technical Report MSR-TR-2020-32, Microsoft, May 2020.

[BDH+18] Rachel K. E. Bellamy, Kuntal Dey, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, Seema Nagar, Karthikeyan Natesan Ramamurthy, John Richards, Diptikalyan Saha, Prasanna Sattigeri, Moninder Singh, Kush R. Varshney, and Yunfeng Zhang. Ai fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias, 2018.

[BG18] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In Sorelle A. Friedler and Christo Wilson, editors, *Proceedings of the*

1st Conference on Fairness, Accountability and Transparency, volume 81 of *Proceedings of Machine Learning Research*, pages 77–91. PMLR, February 2018.

[BHJ+17] Richard Berk, Hoda Heidari, Shahin Jabbari, Michael Kearns, and Aaron Roth. Fairness in criminal justice risk assessments: The state of the art, 2017.

[BMSW22] Robert Bartlett, Adair Morse, Richard Stanton, and Nancy Wallace. Consumer-lending discrimination in the fintech era. *Journal of Financial Economics*, 143(1):30–56, 2022.

[BS16] Solon Barocas and Andrew D. Selbst. Big data's disparate impact. *California Law Review*, 104:671, 2016.

[Car19] Aquiles Carattino. Monkey patching and its consequences, 2019. Accessed 2025-06-19.

[CBN17] Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, April 2017.

[CDG23] Sam Corbett-Davies and Sharad Goel. The measure and mismeasure of fairness. *Journal of Machine Learning Research*, 24:1–117, 2023.

[CDPF+17] Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. Algorithmic decision making and the cost of fairness. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 797–806, New York, NY, USA, 2017. Association for Computing Machinery.

[CGCMO23] Roberta Calegari, Gabriel G. Castañé, Michela Milano, and Barry O'Sullivan. Assessing and enforcing fairness in the ai lifecycle. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages

6554–6562. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Survey Track.

[Che23] Zhisheng Chen. Ethics and discrimination in artificial intelligence-enabled recruitment practices. *Humanities and Social Sciences Communications*, 10(1):567, 2023.

[CHKV20] L. Elisa Celis, Lingxiao Huang, Vijay Keswani, and Nisheeth K. Vishnoi. Classification with fairness constraints: A meta-algorithm with provable guarantees, 2020.

[CR20] Alexandra Chouldechova and Aaron Roth. A snapshot of the frontiers of fairness in machine learning. *Commun. ACM*, 63(5):82–89, April 2020.

[DHP+11] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Rich Zemel. Fairness through awareness, 2011.

[DNL+22] Wesley Hanwen Deng, Manish Nagireddy, Michelle Seng Ah Lee, Jatinder Singh, Zhiwei Steven Wu, Kenneth Holstein, and Haiyi Zhu. Exploring how machine learning practitioners (try to) use fairness toolkits. In *2022 ACM Conference on Fairness Accountability and Transparency*, FAccT '22, page 473–484. ACM, June 2022.

[EFN+17] Danielle Ensign, Sorelle A. Friedler, Scott Neville, Carlos Scheidegger, and Suresh Venkatasubramanian. Runaway feedback loops in predictive policing, 2017.

[Fer23] Emilio Ferrara. Fairness and bias in artificial intelligence: A brief survey of sources, impacts, and mitigation strategies. *Sci*, 6(1):3, December 2023.

[FFM+15] Michael Feldman, Sorelle Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact, 2015.

[GGR24] Ana Cristina Bicharra Garcia, Marcio Gomes Pinto Garcia, and Roberto Rigobon. Algorithmic discrimination in the credit domain: what do we know about it? *AI & SOCIETY*, 39(4):2059–2098, 2024.

[HPS16] Moritz Hardt, Eric Price, and Nathan Srebro. Equality of opportunity in supervised learning, 2016.

[HWVD+19] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé, Miro Dudik, and Hanna Wallach. Improving fairness in machine learning systems: What do industry practitioners need? In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–16. ACM, May 2019.

[KAAS12] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. Fairness-aware classifier with prejudice remover regularizer. pages 35–50, 09 2012.

[KBA24] Tahsin Alamgir Kheya, Mohamed Reda Bouadjenek, and Sunil Aryal. The pursuit of fairness in artificial intelligence models: A survey, 2024.

[KC12] F. Kamiran and T.G.K. Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33(1):1–33, 2012.

[KKBK21] Matthias Kuppler, Christoph Kern, Ruben L. Bach, and Frauke Kreuter. Distributive justice and fairness metrics in automated decision-making: How much overlap is there?, 2021.

[KKZ12] Faisal Kamiran, Asim Karim, and Xiangliang Zhang. Decision theory for discrimination-aware classification. In *2012 IEEE 12th International Conference on Data Mining*, pages 924–929, 2012.

[KMR16] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores, 2016.

[KNP+25] Dewant Katare, David Solans Noguero, Souneil Park, Nicolas Kourtellis, Marijn Janssen, and Aaron Yi Ding. Analyzing and mitigating bias for vulnerable road users by addressing class imbalance in datasets. *IEEE Open Journal of Intelligent Transportation Systems*, 6:590–604, 2025.

[KZC12] Faisal Kamiran, Indrė Zliobaitė, and Toon Calders. Quantifying explainable discrimination and removing illegal discrimination in automated decision making. *Knowledge and Information Systems*, 1:in press, 06 2012.

[LCZ+24] Xinyue Li, Zhenpeng Chen, Jie M. Zhang, Federica Sarro, Ying Zhang, and Xuanzhe Liu. Bias behind the wheel: Fairness testing of autonomous driving systems, 2024.

[LS21] Michelle Seng Ah Lee and Jat Singh. The landscape and gaps in open source fairness toolkits. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery.

[MCCO24] Matteo Magnini, Giovanni Ciatto, Roberta Calegari, and Andrea Omicini. Enforcing fairness via constraint injection with FaUCI. In Roberta Calegari, Virginia Dignum, and Barry O'Sullivan, editors, *AEQUITAS 2024: Fairness and Bias in AI*, volume 3808 of *CEUR Workshop Proceedings*, pages 8:1–8:13. CEUR-WS, October 2024. Proceedings of the 2nd Workshop on Fairness and Bias in AI co-located with 27th European Conference on Artificial Intelligence (ECAI 2024).

[MMS+21] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys*, 54(6):1–35, 2021.

[MNDWLS24] Marie Mirsch Née Decker, Laila Wegner, and Carmen Leicht-Scholten. Procedural fairness in algorithmic decision-making: the

role of public engagement. *Ethics and Information Technology*, 27, 11 2024.

[MPB+21] Shira Mitchell, Eric Potash, Solon Barocas, Alexander D'Amour, and Kristian Lum. Algorithmic fairness: Choices, assumptions, and definitions. *Annual Review of Statistics and Its Application*, 8:141–163, 2021.

[MWK23] Aboli Marathe, Rahee Walambe, and Ketan Kotecha. In rain or shine: Understanding and overcoming dataset bias for improving robustness against weather corruptions for autonomous vehicles, 2023.

[NHS+24] Khalida Walid Nathim, Nada Abdulkareem Hameed, Saja Abdulfattah Salih, Nada Adnan Taher, Hayder Mahmood Salman, and Dmytro Chornomordenko. Ethical ai with balancing bias mitigation and fairness in machine learning models. In *2024 36th Conference of Open Innovations Association (FRUCT)*, pages 797–807, 2024.

[OPVM19] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019.

[PRW+17] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. On fairness and calibration. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[SG21] Harini Suresh and John Guttag. A framework for understanding sources of harm throughout the machine learning life cycle. In *Equity and Access in Algorithms, Mechanisms, and Optimization*, EAAMO '21, page 1–9. ACM, October 2021.

[SGA19] Ismail Abiodun Sulaimon, Ahmed Ghoneim, and Mubarak Alrashoud. A new reinforcement learning-based framework for unbiased autonomous software systems. In *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, pages 1–6, 2019.

[TYJY+24] Aida Tayebi, Mehdi Yazdani-Jahromi, Ali Khodabandeh Yalabadi, Niloofar Yousefi, and Ozlem Ozmen Garibay. Learning fair representations: Mitigating statistical dependencies. In Helmut Degen and Stavroula Ntoa, editors, *Artificial Intelligence in HCI*, pages 105–115, Cham, 2024. Springer Nature Switzerland.

[VB17] Michael Veale and Reuben Binns. Fairer machine learning in the real world: Mitigating discrimination without collecting sensitive data. *Big Data & Society*, 4(2):2053951717743530, 2017.

[VR18] Sahil Verma and Julia Rubin. Fairness definitions explained. In *Proceedings of the International Workshop on Software Fairness*, FairWare '18, page 1–7, New York, NY, USA, 2018. Association for Computing Machinery.

[XZ24] Ruicheng Xian and Han Zhao. A unified post-processing framework for group fairness in classification, 2024.

[YSE+23] Jenny Yang, Andrew Soltan, David Eyre, Yang Yang, and David Clifton. An adversarial training framework for mitigating algorithmic biases in clinical machine learning. *NPJ digital medicine*, 6:55, 03 2023.

[YW24] Chih-Cheng Rex Yuan and Bow-Yaw Wang. Ensuring fairness with transparent auditing of quantitative bias in ai systems, 2024.

[ZHL+23] Jie Zhu, Mengsha Hu, Xueyao Liang, Amy Zhang, Ruoming Jin, and Rui Liu. Fairness-sensitive policy-gradient reinforcement learning for reducing bias in robotic assistance, 2023.

[ZLM18] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning, 2018.

[ZWS+13] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 325–333, Atlanta, Georgia, USA, June 2013. PMLR.

Acknowledgements

Optional. Max 1 page.