Training Data Debugging for the Fairness of Machine Learning Software

Yanhui Li
State Key Laboratory for Novel Software Technology, Nanjing University, China
yanhuili@nju.edu.cn

Linghan Meng*
State Key Laboratory for Novel Software Technology, Nanjing University, China
menglinghan@smail.nju.edu.cn

Lin Chen†
State Key Laboratory for Novel Software Technology, Nanjing University, China
lchen@nju.edu.cn

Li Yu
State Key Laboratory for Novel Software Technology, Nanjing University, China
yuli@smail.nju.edu.cn

Di Wu
Momenta, Suzhou, China
wudi@momenta.ai

Yuming Zhou
State Key Laboratory for Novel Software Technology, Nanjing University, China
zhouyuming@nju.edu.cn

Baowen Xu
State Key Laboratory for Novel Software Technology, Nanjing University, China
bwxu@nju.edu.cn

ABSTRACT

With the widespread application of machine learning (ML) software, especially in high-risk tasks, the concern about their unfairness has been raised towards both developers and users of ML software. The unfairness of ML software indicates the software behavior affected by the sensitive features (e.g., sex), which leads to biased and illegal decisions and has become a worthy problem for the whole software engineering community.

According to the "data-driven" programming paradigm of ML software, we consider the root cause of the unfairness as biased features in training data. Inspired by software debugging, we propose a novel method, **L**inear-regression based **T**raining **D**ata **D**ebugging (LTDD), to **debug** feature values in training data, i.e., (a) identify which features and which parts of them are biased, and (b) exclude the biased parts of such features to recover as much valuable and unbiased information as possible to build fair ML software. We conduct an extensive study on nine data sets and three classifiers to evaluate the effect of our method LTDD compared with four baseline methods. Experimental results show that (a) LTDD can better improve the fairness of ML software with less or comparable damage to the performance, and (b) LTDD is more actionable for fairness improvement in realistic scenarios.

*Linghan Meng is the co-first author.
†Lin Chen is the corresponding author.

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9221-1/22/05…$15.00
https://doi.org/10.1145/3510003.3510091

KEYWORDS

Debugging, Fairness, ML Software, Training Data

ACM Reference Format:
Yanhui Li, Linghan Meng, Lin Chen, Li Yu, Di Wu, Yuming Zhou, and Baowen Xu. 2022. Training Data Debugging for the Fairness of Machine Learning Software. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3510003.3510091

1 INTRODUCTION

Machine learning (ML) software has penetrated many aspects of our daily lives, whose results are employed in high-stake applications to make decisions or predictions. For example, people have applied ML software to identify credit risks [2], to prove loan applications [4], to predict heart disease [1], and even to estimate reoffending probabilities [10]. Along with the widespread usage of ML software, the concern about its fairness has also been raised towards developers, users, and regulators of ML software. The unfairness of ML software usually indicates the discriminatory treatment of different groups divided by sensitive features [40], e.g., sex. According to biased decisions made by ML software, one group (e.g., male) may have a particular advantage, i.e., with more opportunity to obtain "favorable" decisions[1], called privileged, over the other group (e.g., female), called unprivileged. Under such circumstances, the biased treatment not only undermines the objectivity of ML software but also violates some anti-discrimination laws [14].

The fairness of ML software has been considered a worthy software engineering (SE) research problem [46], where fairness has come to be seen as a core non-functional quality property [47] of ML software. Brun and Meliou [20] reported a new kind of software defect, "fairness defect", which is related to software behavior in

[1]To illustrate, for loan applications, "approval" is the favorable decision, and "rejection" is the unfavorable decision.

a biased manner and needs to be tackled. Chakraborty et al. [25] argued that, when discovering unfair problems with ML software, it is the job of software engineers to reduce discrimination. Biswas and Rajan [18] conducted an empirical evaluation of fairness on 40 real-world ML software under a comprehensive set of fairness indicators. They observed that performance optimization techniques might lead to unfairness.

Following the data-driven programming paradigm [47], ML software obtains its decision logic from training data [13]. The behavior of ML software to a large extent is determined by the quality of training data [44], i.e., biased training data may lead to the trained ML software with statistical discrimination. Researchers have tried to explain the effect of training data on the fairness of ML software from the following angles:

- *The size of feature sets.* Zhang and Harman [46] compared the results of ML software with different sizes of feature sets and reported that enlarging the size of feature sets in training data would increase ML software fairness.
- *Biased labels and imbalanced distributions.* Chakraborty et al. [23] assumed that label information might contain bias and proposed a novel method, Fairway, to remove training samples with biased labels. Recently, they [22] postulated root causes of unfairness as a combination of biased labels and imbalanced internal distributions and proposed Fair-Smote to remove biased labels and rebalance internal distributions.

This paper deviates from previous studies, as it adopts a different angle: the root cause of the unfairness could be **biased features** in training data. Here, we define biased features as *features that are highly related to the sensitive feature in training data.* In the development process of ML software, developers have noticed that biased features might lead to unfair behaviors of ML software. As reported in Amazon fairness issues [8], when developers constructed Amazon's same-day delivery service, they observed that some features in the training data are highly related to the race (the sensitive feature that is already excluded in the training process) of customers, e.g., the zipcode feature could obviously deduce whether the delivery address is in white or non-white communities. As a result, even though the delivery service runs without obtaining the race information, it still prefers the delivery address with the zipcode in white communities, and causes unfairness between white and non-white customers. This kind of fairness issue about biased features has been considered fairness bugs [20], which asks for novel methods to debug (formally find and resolve) biased features. However, to our knowledge, there are no previous studies about ML software debugging techniques focusing on biased features.

To obtain unbiased features, we try to **debug** feature values in training data, i.e., (a) identify which features and which parts of them are biased, and (b) exclude the biased parts of such features to recover as much useful and unbiased information as possible. To achieve this goal, we propose a novel method, **L**inear-regression based **T**raining **D**ata **D**ebugging (**LTDD**), which employs the linear regression model [50] to identify the biased features (see details in Section 3). On the whole, our method comprises the following three steps: (a) adopting Wald test [30] with t-distribution to check whether the features contain significant bias, (b) estimating the biased parts of training features as explained variances [49] in the original feature values by the sensitive feature, and (c) removing the biased parts of training features by subtracting the estimated values from original feature values to construct fair ML software.

To evaluate our method LTDD, we conduct an extensive empirical study on nine tasks [1–3, 5–7, 9, 10] with common sensitive features (sex, race, and age) under three widely used ML classifiers, Logistic Regression, Naive Bayes, and SVM. To assess the performance of LTDD, we introduce four methods as the baselines: two state-of-the-art fairness methods Fair-Smote [22] at FSE'2021 and Fairway [23] at FSE'2020, and two widely studied pre-processing methods Reweighing [31] and Disparate Impact Remover [27]. Our experimental results support the claim that LTDD performs well: compared with the baselines, LTDD can largely improve the fairness of ML software, with less or comparable damage to its performance.

Our study makes the following two contributions:

- Strategy. This paper proposes a novel fairness method, Linear-regression based Training Data Debugging (LTDD), as an efficient strategy to alleviate ML software's unfairness by identifying biased features and removing their biased parts in training data.
- Study. This paper contains an extensive empirical study on nine tasks under three ML classifiers. The results of our experiment show that LTDD can largely improve the fairness of ML software, with less or comparable damage to its performance.

The rest of this paper is structured as follows. We introduce background in Section 2 and present a detailed description of our method LTDD in Section 3. Section 4 presents the experimental settings, including studied datasets, baseline methods, prediction settings, and used classifiers. Sections 5 and 6 give the experimental results and important further discussions. Finally, Section 7 describes threats to validity, and Section 8 concludes our paper.

2 BACKGROUND

2.1 Preliminaries

We introduce some concepts and symbols about the fairness of ML software here, which facilitate the readers to understand the following parts.

ML Software. For binary classification tasks[2], ML Software $\mathcal{S}_{\mathcal{ML}}$ would be considered as a function mapping the domain feature vectors $\mathbf{x} = [x_1, x_2, \ldots, x_d] \in \mathbb{R}^d$ into class labels $y \in \{0, 1\}$, i.e., $\mathcal{S}_{\mathcal{ML}} : \mathbb{R}^d \rightarrow \{0, 1\}$. Normally, for a new input $\mathbf{x}$, we use y to denote the actual label, while $\hat{y} = \mathcal{S}_{\mathcal{ML}}(\mathbf{x})$ indicates the predicted label by ML Software.

Sensitive Feature. The sensitive feature is a feature x_s in the domain feature vector $\mathbf{x}$, which divides samples into two categories (called privileged and unprivileged), in which the benefits received differ considerably. For example, sex is one of the common sensitive features. As observed in [2, 9, 23], the male group is often regarded as the privileged group with more favorable labels predicted by ML Software.

Group Fairness. ML software has group fairness, if the sensitive feature does not affect the group probability of decision outcomes

[2]Here, we focus on the ML Software for binary classification tasks. Our definition can be easily generalized to multiple classification tasks.

[46], i.e., privileged groups and unprivileged groups have (almost) equal probabilities of being predicted as favorable labels. Group fairness is widely studied [23, 47] and guaranteed by legal regulations on fairness [27, 28]. In this study, we focus on group fairness.

2.2 Related Work

We introduce the related work from three aspects: fairness testing and evaluation, bias removal algorithm, and data debugging.

2.2.1 Fairness Testing and Evaluation. Fairness testing hopes to find instances of bias in the data set and evaluate the fairness of the model's prediction results. AEQUITAS [42] finds instances of discrimination in the dataset and generates more instances to retrain further. It applies two search strategies global search based on random technology and local search conducted by disturbing the instances found in global search. Affected by AEQUITAS, Aggarwal et al. [11] proposed a black-box technique for fairness testing based on global and local search combined with symbolic execution and local interpretable models. Zhang et al. [48] proposed a white-box testing technique to find and generate test cases, which aims at complex DNN models and introduces gradient-based methods into global and local search.

2.2.2 Bias Removal Algorithm. Bias removal algorithms can be divided into three categories: pre-processing, in-processing, and post-processing.

(a) The pre-processing algorithm affects the training of the model by modifying the data to achieve fairness. As the pre-processing algorithm is the most related work to this study, we detailedly summarize pre-processing approaches in the following three folds:

- datapoint modification which deletes, generates, or assigns weights to datapoints of training data ([22, 23, 31]);
- feature selection which leverages feature construction to choose more suitable features that lead to both high accuracy and fairness [37, 46];
- feature modification which modifies features as a debiasing method. Disparate Impact Remover (DIR) is the most related feature modification method [27] and has also been employed as one of four baselines (see Section 4.2). DIR focuses on the predictability of sensitive features from non-sensitive features (non-sensitive→sensitive), which is an opposite strategy of our method to evaluate the bias.

Compared with related work, our method employs linear regression to estimate the biased association of non-sensitive features from sensitive features (i.e., sensitive→non-sensitive, see Section 3), which is a novel angle to assess and remove the bias.

(b) The in-processing method is to modify the model to make the prediction result of the model fairer. Zhang et al. [45] proposed a method to improve the model through negative feedback. It combines an adversarial model to obtain the difference between sensitive features and non-sensitive features to reduce bias.

(c) The post-processing method achieves prediction fairness by modifying the prediction results of the model. Hardt et al. [29] proposed a post-processing method based on equal opportunity difference, which achieves the goal of fairness by transferring the cost of bad classification from unprivileged groups to decision makers.

2.2.3 Data Debugging. The goal of data debugging is to locate and modify the data that causes program errors. Chad and Ronald [39] introduced program chipping to simplify inputs that cause program errors, which can automatically delete or cut off certain parts of the inputs. To debug the input data, Lukas et al. [32] introduced an algorithm called ddmax, which maximizes the subset of input that the program can still process to repair the input data. They argued that ddmax is the first general technology to repair faulty inputs automatically. Wu et al. [43] proposed a complaint-driven data debugging system named Rain which allows users to complain about the intermediate or final output of the query and returns the smallest set of training subset that can solve the bug after deletion.

3 OUR APPROACH

In this section, we present a detailed description of our approach. First, we show a motivation example. Next, we employ linear regression to model biased features. Finally, we present our method Linear-regression based Training Data Debugging (LTDD).

3.1 Motivation Example

In Section 1, we quoted a report [8] about Amazon fairness issues to show that, when biased features appear in the training data, ML software might discriminate between privileged and unprivileged groups. Here, we propose a more detailed motivation example to show why and how features are biased, i.e., why features are considered related to sensitive features and how they cause ML software unfairness. This example is also helpful to demonstrate the following steps in Sections 3.2 and 3.3.

Figure 1(a) presents a snippet taken from the COMPAS dataset [10] which is used to evaluate the possibility of a criminal defendant reoffending. The snippet contains six samples with the sensitive feature "Race" and the non-sensitive feature "Age". Here we consider the snippet[3] as a small but representative motivation example to show the training dataset with biased features. As can be seen in Figure 1(a), we have the following observations:

- Bias between race groups is observed in training data. There are obviously different rates of favorable labels (no-reoffend) between privileged (white) and unprivileged (non-white) groups, e.g., all three white people are labeled as no-reoffend (the favorable decision). In contrast, only one out of three non-white is labeled as no-reoffend.
- *Why* "Age" contains race bias? There is a high association between "Race" and "Age": the "Age" values of the privileged group (three white people) are relatively much *larger* than that of the unprivileged group (three non-white people).
- *How* "Age" causes ML software biased, even without knowing "Race" information? Since larger "Age" values (> 33) imply white people, the race bias may be introduced by the "Age" values when training ML software.

3.2 Linear Regression Modeling

As discussed above, some features in the training data might be biased, leading to unfairness in ML software. To obtain unbiased

[3]For brevity, we focus on the six samples here and discuss the biased association between the sensitive feature "Race" and the non-sensitive feature "Age" in this snippet. In Section 6.2, we will extend the scope to the whole set of COMPAS dataset.

Group	Domain features			Label
	Sensitive	Non-sensitive		
	Race	Age	...	
Privileged	White	36	...	no-reoffend
	White	38	...	no-reoffend
	White	34	...	no-reoffend
Unprivileged	Non-white	30	...	no-reoffend
	Non-white	32	...	reoffend
	Non-white	28	...	reoffend

(a) Association between Age and Race.

Employ linear regression equation
$\text{Age} = a + b \cdot \text{Race} + \mu$
to estimates $\hat{a} = 30$ and $\hat{b} = 6$
$36 = a + b \cdot 1 + \mu$
$38 = a + b \cdot 1 + \mu$
$34 = a + b \cdot 1 + \mu$
$30 = a + b \cdot 0 + \mu$
$32 = a + b \cdot 0 + \mu$
$28 = a + b \cdot 0 + \mu$

(b) Calculate bias parts

Remove biased parts from Age:		
$\hat{\text{Age}}$	=	$\hat{a} + \hat{b} \cdot \text{Race}$
Age^u	=	$\text{Age} - \hat{\text{Age}}$
0	=	$36 - (30 + 6 \cdot 1)$
2	=	$38 - (30 + 6 \cdot 1)$
-2	=	$34 - (30 + 6 \cdot 1)$
0	=	$30 - (30 + 6 \cdot 0)$
2	=	$32 - (30 + 6 \cdot 0)$
-2	=	$28 - (30 + 6 \cdot 0)$

(c) Remove biased parts

Figure 1: An example of biased features (Age) which are highly related to sensitive features (Race).

features, we have to debug feature values in training data, i.e., (a) identify which features and which parts of them are biased, and (b) remove the biased parts to recover as much helpful and unbiased information as possible.

The main idea of our training data debugging is to apply the linear regression equation [50] to analyze the association between non-sensitive features and sensitive features, which is a simple but effective method applied in many SE research areas, such as defect prediction [49] and software effort estimation [12]. Specifically, we build a linear regression model for the non-sensitive features x_n on the sensitive features x_s:

$$x_n = a + b \cdot x_s + \mu \tag{1}$$

where the symbols a, b, and μ denote the population regression intercept, slope, and residual, respectively. We employ Wald test [30] with t-distribution to check whether the null hypothesis (that the slope of the linear regression model is zero) holds. When the p-values ≥ 0.05, we ignore the current non-sensitive feature and move to the next; otherwise, we consider that x_n contains *significant* bias and conduct the following steps to estimate and exclude bias.

By calculating on values of x_n and x_s in training data of ML software, we obtain the estimates $\hat{a}$ and $\hat{b}$ for a and b, respectively. Based on the above estimates, we have the following equation to measure the association between x_n and x_s:

$$\hat{x_n} = \hat{a} + \hat{b} \cdot x_s \tag{2}$$

where $\hat{x_n}$ could be considered as the predicted value generated from Equation 1. Intuitively, $\hat{x_n}$ is the explained variance in the original non-sensitive features x_n by the sensitive feature x_s. We consider $\hat{x_n}$ as the *biased* part of x_n, remove $\hat{x_n}$ from x_n, and denote the remaining unbiased part as x_n^u:

$$x_n^u = x_n - \hat{x_n} \tag{3}$$

Fig. 1(b) and Fig. 1(c) show the detail of calculating the sample estimates and the remaining values based on the six samples in Fig. 1(a). In detail, after translating categorical variables white/non-white into numerical variables 1/0, we have the following linear regression equations:

$$\left[36, 38, 34, 30, 32, 28\right]^T = a + b \cdot \left[1, 1, 1, 0, 0, 0\right]^T + \mu \tag{4}$$

We find that p-value < 0.05 and obtain the estimates $\hat{a} = 30$ and $\hat{b} = 6$. Finally, we obtain the unbiased age value Age^u by subtracting the predicted values $\hat{\text{Age}} = 30 + 6 \cdot \text{Race}$: $\text{Age}^u = \text{Age} - \hat{\text{Age}}$. As can be seen in Fig. 1(c), the distributions of "Age" values under privileged and unprivileged groups become independent of "Race" after removing the biased parts, i.e., the revised "Age" values in two groups are the same.

3.3 Our Method

Based on the linear regression model, we propose our algorithm Linear-regression based Training Data Debugging (LTDD), shown in Algorithm 1.

Given the training dataset $\mathcal{S}_{tr} = \{\langle \mathbf{x}_1, y_1 \rangle, \cdots, \langle \mathbf{x}_n, y_n \rangle\}$, where for any j ($1 \leq j \leq n$), $\mathbf{x}_j = [x_1^j, \ldots, x_d^j]$ is a d-dimension vector to denote d feature values, x_d^j is assumed to be the sensitive feature value and the other values $x_1^j, \cdots, x_{d-1}^j$ are non-sensitive feature values, and $y_j \in \{0, 1\}$, our method LTDD comprises the following three steps:

(1) *Identify the biased features and estimate the biased parts of them.* For each non-sensitive feature x_i, we evaluate the association between the sensitive features x_d and x_i in the training dataset. It is worth noting that, since the association between some non-sensitive features and the sensitive feature may be trivial, we employ Wald test (line 7) with t-distribution to check whether the null hypothesis (that the slope $\hat{b}$ of the linear regression model is zero) holds. Specifically, we introduce the p-value of Wald test to avoid unnecessary removing steps, i.e., consider "p-value< 0.05" (line 8) as a precondition. If "p-value< 0.05" holds, we calculate the estimates $\hat{a_i}$ and $\hat{b_i}$ of the linear regression model (line 9), which are sorted in $\mathcal{E}_a$ and $\mathcal{E}_b$ (line 10).

(2) *Exclude the bias parts from training samples.* In this step, for any training sample $\langle \mathbf{x}_j, y_j \rangle$, we conduct the following two operators to eliminate bias: deleting the sensitive feature (line 12) and revising the non-sensitive feature values by removing the association (line 13-14). After that, we build a (fair) ML software $\mathcal{S}_{\mathcal{ML}}$ based on the revised and unbiased training dataset (line 15), i.e., $\mathcal{S}_{\mathcal{ML}} : \mathbb{R}^{d-1} \rightarrow \{0, 1\}$.

(3) *Apply the same revision on the testing samples.* As we revised the dimension and distribution of training data, we apply the same revision on the testing sample $\mathbf{x}_{te}$ to fit ML software (line 16-18), i.e., delete the sensitive feature and revise the other attributes by the estimates $\mathcal{E}_a[i]$ and $\mathcal{E}_b[i]$ calculated on the training samples. Finally, we use $\mathcal{S}_{\mathcal{ML}}$ to predict the label of $\mathbf{x}_{te}$ (line 19).

Algorithm 1: **L**inear-regression based **T**raning **D**ata **D**ebugging LTDD($\mathcal{S}_{tr}, \mathbf{x}^{te}$)

Input: The training dataset $\mathcal{S}_{tr} = \{\langle \mathbf{x}_1, y_1\rangle, \cdots, \langle \mathbf{x}_n, y_n\rangle\}$, where $\mathbf{x}_j = [x_1^j, \ldots, x_d^j]$ is a d-dimension vector to denote the d attribute values, x_d^j is the sensitive feature value and the other $x_1^j, \cdots, x_{d-1}^j$ are non-sensitive feature values, $y_j \in \{0, 1\}$, and the testing sample $\mathbf{x}_{te} = [x_1^{te}, \ldots, x_d^{te}]$.

Output: a ML software $\mathcal{S}_{\mathcal{ML}}$ and the predicted label $\mathcal{S}_{\mathcal{ML}}(\mathbf{x}^{te})$ for $\mathbf{x}^{te}$.

initialize $(d-1)$-dimension array $\mathcal{E}_a[1:d-1]$ with $\mathcal{E}_a[i] = 0$, which is used to store the estimation result of intercept $\hat{a_i}$;
initialize $(d-1)$-dimension array $\mathcal{E}_b[1:d-1]$ with $\mathcal{E}_b[i] = 0$, which is used to sort the estimation result of slope $\hat{b_i}$;
construct the column vector V_d of the sensitive feature values from $\mathcal{S}_{tr}$: $V_d = [x_d^1, \cdots, x_d^n]^T$;
for $i \in \{1, \cdots, d-1\}$ **do**
    construct the column vector V_i of the current non-sensitive feature values: $V_i = [x_i^1, \cdots, x_i^n]^T$;
    apply the linear regression model on V_i: $V_i = a_i + b_i \cdot V_d + \mu$;
    conduct Wald test with t-distribution to get the p-value;
    if p-value< 0.05 **then**
        estimate $\hat{a_i}$ and $\hat{b_i}$ for a_i and b_i;
        insert $\hat{a_i}$ and $\hat{b_i}$ into $\mathcal{E}_a$ and $\mathcal{E}_b$: $\mathcal{E}_a[i] = \hat{a_i}$, $\mathcal{E}_b[i] = \hat{b_i}$;
for $\langle \mathbf{x}_j, y_j\rangle \in \mathcal{S}_{tr}$ **do**
    remove the sensitive feature from $\mathbf{x}_j$: $\mathbf{x}_j = \mathbf{x}_j[1:d-1]$;
    for $i \in \{1, \cdots, d-1\}$ **do**
        remove the biased part based on the estimation: $x_i^j = x_i^j - (\mathcal{E}_a[i] + \mathcal{E}_b[i] \times x_d^j)$;
train ML software $\mathcal{S}_{\mathcal{ML}}$ from the revised $(d-1)$-dimension training data;
remove the sensitive feature from $\mathbf{x}_{te}$: $\mathbf{x}_{te} = \mathbf{x}_{te}[1:d-1]$;
for $i \in \{1, \cdots, d-1\}$ **do**
    apply the same revision on the testing sample $\mathbf{x}_{te}$: $x_i^{te} = x_i^{te} - (\mathcal{E}_a[i] + \mathcal{E}_b[i] \times x_d^{te})$;
return $\mathcal{S}_{\mathcal{ML}}$ and $\mathcal{S}_{\mathcal{ML}}(\mathbf{x}_{te})$;

4 EXPERIMENT SETUPS

In this section, we will introduce the experiment setups. Code and data are publicly available online and reusable (see Section 8.1).

4.1 Studied Datasets

We employ nine datasets to evaluate the performance of our method, all of which have been widely used in previous software fairness studies [23, 46].

Adult [3]. This data set contains 48,842 samples with 14 features. The goal of the data set is to determine whether a person's annual income can be larger than 50k. This dataset has two sensitive features sex and race.

COMPAS [10]. COMPAS is the abbreviation of Correctional Offender Management Profiling for Alternative Sanctions, which is a commercial algorithm for evaluating the possibility of a criminal defendant committing a crime again. The dataset contains the variables used by the COMPAS algorithm to score the defendant and the judgment results within two years. There are over 7000 rows in this dataset, with two sensitive features sex and race.

Table 1: The datasets used in this experiment

Dataset	#Feature	Size	Sensitive feature
Adult	14	48,842	sex/race
COMPAS	28	7,214	sex/race
Default	24	30,000	sex
German	20	1,000	sex
Heart	14	297	age
Bank	16	45,211	age
Student	33	1,044	sex
MEPS15	139	15,830	race
MEPS16	139	15,675	race

Default of Credit Card Clients (Default for short) [9]. The purpose of this dataset is to determine whether customers will default on payment through customers' information. It contains 30,000 rows and 24 features, including one sensitive feature sex.

German Credit (German for short) [2]. This data set contains 1000 rows and 20 features, where each person is divided into good or bad credit risk according to feature values. The sensitive feature in this dataset is sex.

Heart Disease (Heart for short) [1]. This dataset judges whether the patient has heart disease based on the collected information. The data set includes 76 features, where age is the sensitive feature in this dataset.

Bank Marketing (Bank for short) [5]. The goal of this dataset is to predict whether the client will subscribe to a term deposit. It contains 45211 pieces of data, and its sensitive feature is age.

Student Performance (Student for short) [6]. This data set analyzes student performance in secondary education. It contains 1044 pieces of data with the sensitive feature sex.

Medical Expenditure Panel Survey (MEPS for short) [7]. It collects data about the health services used by respondents, the cost and frequency of services, and demographics. We use the survey data for the calendar years 2015 and 2016, named MEPS15 and MEPS16, respectively. They contain more than 15,000 rows, and the sensitive feature is race.

Since there are two sensitive features (sex and race) in two datasets Adult and COMPAS, we convert the 9 datasets into 11 scenarios by considering one sensitive feature in one scenario, i.e., the Adult (COMPAS) dataset appears twice as Adult_sex and Adult_race (COMPAS_sex, COMPAS_race).

4.2 Baseline Methods

As described in Section 3, our method LTDD is a pre-processing method, i.e., debugging training data before the construction of ML software to reduce the bias. We introduce four fairness methods as the baselines: two state-of-the-art methods Fairway at FSE'2020 and Fair-Smote at FSE'2021, and two widely used pre-processing methods Reweighing and Disparate Impact Remover (DIR) for comparison.

- Fair-Smote [22]: Fair-Smote is a pre-processing method that uses the modified SMOTE method to make the distribution of

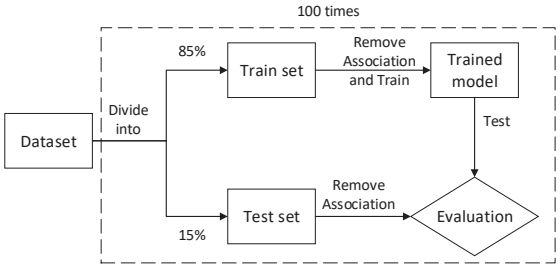


Figure 2: Prediction settings for our experiment.

sensitive features in the data set consistent and then deletes biased data through situation testing.

- Fairway [23]: Fairway is a hybrid algorithm that combines pre-processing and in-processing methods. It trains the models separately based on samples with different sensitive features to remove biased data points. Then it uses the Flash technique [35] for multi-objective optimization, including model performance indicators and fairness indicators.
- Reweighing [31]: Reweighing is a pre-processing method that calculates a weight value for each data point based on the expected probability and the observed probability, to help the unprivileged class have a greater chance of obtaining favorable prediction results.
- Disparate Impact Remover (DIR) [27]: this method is a pre-processing technology, whose primary goal is to eliminate Disparate Impact and increase fairness between groups by modifying feature values except for the sensitive features.

Our code for implementing LTDD and all baselines is based on Python 3.7.6, where Reweighing and DIR are implemented using AIF360 [16] (version 0.3.0).

4.3 Prediction Settings and Classifiers

Following the training and testing process in Fairway [23], we randomly divide the studied dataset into two parts: 85% is used to preprocess and train[4], and the remaining 15% is used as the test set. Since our studied methods have randomness, as shown in Figure 2, we repeat our experiment 100 times to reduce this influence.

Like the previous researches on fairness testing, we introduce logistic regression model [23, 24] as a main classifier to test fairness in this experiment. To test the effect of our method under other classifiers, we also employ Naive Bayes and SVM, which are also widely used classifiers in SE research [38, 41]. When we implement all classifiers, we employ the default settings of the Sklearn module.

4.4 Fairness and Performance Metrics

As reported in [17], the increasing of software fairness may damage the performance, which is called the accuracy-fairness tradeoff. Our experiment focuses on both fairness and performance metrics to check the relationship between fairness and performance changed after applying fairness methods to ML software.

[4]In Fairway [23], 70% are for training and 15% are for validation. For other methods without validation, we employ the total 85% (70%+15%) for training.

Fairness Metrics. Disparate Impact (DI) and Statistical Parity Difference (SPD) are the main indicators we use to measure fairness. Given the studied ML software $\mathcal{S}_{\mathcal{ML}}$, the sample $\mathbf{x}$, the label y, the predicted $\hat{y} = \mathcal{S}_{\mathcal{ML}}(\mathbf{x})$ of $\mathbf{x}$, and the sensitive feature x_s, we define Disparate Impact and Statistical Parity Difference as follows:

Disparate Impact (DI) [27]. It indicates the ratio of the probabilities of favorable results obtained by the unprivileged ($x_s = 0$) and privileged ($x_s = 1$) classes.

$$\text{DI} = p(\hat{y} = 1|x_s = 0)/p(\hat{y} = 1|x_s = 1)$$

Statistical Parity Difference (SPD) [21]. It is similar to DI, but it represents the difference between the unprivileged and privileged classes to obtain favorable results.

$$\text{SPD} = p(\hat{y} = 1|x_s = 0) - p(\hat{y} = 1|x_s = 1)$$

When reporting DI, we calculate its absolute distance from 1, i.e., $|1 - \text{DI}|$ [19]. While for SPD, we report the absolute value result $|\text{SPD}|$ [23].

We take DI and SPD as the main fairness metrics due to the following reasons. (a) DI and SPD are designed to indicate the difference between the decision distributions of privileged and unprivileged individuals, which is specifically prohibited by anti-discrimination laws [27]; (b) Based on their calculation, DI and SPD are independent of the label information about the decisions (i.e., they need only the prediction $\hat{y} = \mathcal{S}_{\mathcal{ML}}(\mathbf{x})$ rather than the label y), which is suspected of containing bias [23]. Besides, in Section 6.3, we will present the results under other fairness metrics, including AOD and EOD [15].

Performance Metrics: we introduce three performance metrics to measure the performance of the classifiers.

Accuracy (ACC for short). It measures the ratio of correct predictions on total data:

$$\text{ACC} = (|TP| + |TN|)/|Total|$$

where TP denotes the true positive samples, TN denotes the true negative samples, and $Total$ denotes the total samples.

Recall. Recall represents the probability of being predicted as positive samples in samples that are actually positive:

$$\text{Recall} = |TP|/(|TP| + |FN|)$$

where FN denotes the false negative samples.

False Alarm. It is also called the false positive rate:

$$\text{False Alarm} = |FP|/(|FP| + |TN|)$$

where FP denotes the false positive samples.

When calculating these above indicators (including both fairness and performance), we employ Python AIF360 module [16] which integrates all these indicators.

4.5 Analysis Method

To verify whether the difference between the metric values obtained by our method and the baselines is statistically significant, we use the Wilcoxon rank sum test [26]. If the difference between the two sets of results is significant, then the p-value obtained by the test should be less than 0.05.

Then, to measure the effect size of the two sets of results, we use Cliff's delta δ [36]. If the value of $|\delta|$ is less than 0.147, the difference between the two sets is negligible; if it is greater than 0.147 but

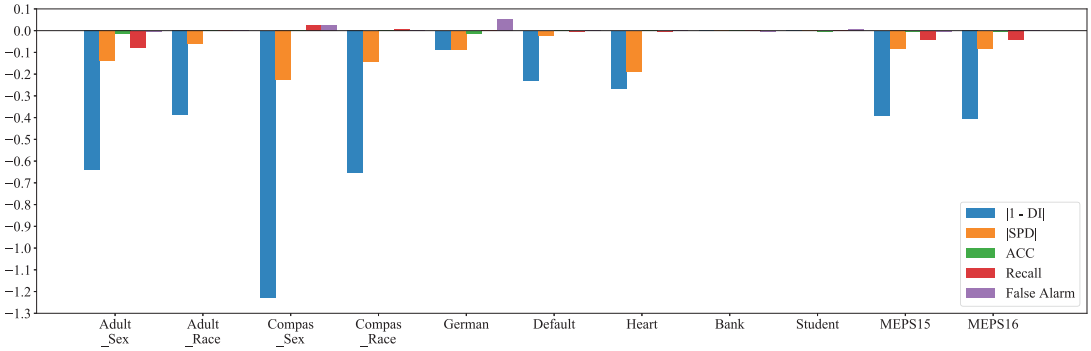


Figure 3: Comparison results of our approach with the original ML software under five indicators. For DI and SPD, more *negative* differences mean more fairness improved.

less than 0.330, then the difference is relatively small; if it is greater than 0.330 but less than 0.474, then there is a medium difference; if it is greater than 0.474, then the difference between the two is considered large.

According to p-value and δ value, we mark "W/T/L" [33, 34] for two sets of compared results. "W" means our result wins, where the corresponding p-value is less than 0.05, and the δ value is greater than 0.147. "L" means our results lose, where the p-value is less than 0.05, and the δ value is smaller than -0.147. Otherwise, the situation is "T", which means the two sets of results are tied.

5 EXPERIMENT RESULTS

In this section, we present the results of our three RQs, along with their motivations, approaches, and findings.

RQ1: How well does our algorithm improve the fairness of ML software?

Motivation and Approach. We first need to check whether our method can eliminate the discrimination that exists in training data. Meanwhile, we hope that our method can damage the performance of ML software as little as possible. The main classifier used in this RQ is the logistic regression model. To eliminate the influence of randomness, we repeat the experiment 100 times and report the average values. We compare the results of our method and baselines under five indicators: (a) DI and SPD as fairness indicators, and (b) ACC, Recall, and False Alarm as performance indicators.

Results. In Figure 3, we report the change of our methods applying to original ML software. The blue and orange bars in the figure represent the changes in $|1 - DI|$ and $|SPD|$, respectively. Based on the calculation of $|1 - DI|$ and $|SPD|$, the smaller the value of $|1 - DI|$ and $|SPD|$, the fairer the prediction result of the model. Therefore, the more **negative** the difference observed, the more our method improves the fairness of the standard classifier. It can be seen that our method LTDD significantly improves the fairness of the standard classifier in most scenarios. Especially, on Compas dataset, LTDD reduces the value of $|1 - DI|$ by 1.2.

Besides, the other three bars represent the changes of ACC, Recall, and False Alarm. As can be seen, our method has minimal impact on the performance of ML software in most scenarios. Even under the two data sets COMPAS and German, our method has surprisingly improved the Recall of the original ML software.

> Answer to RQ1: Our method can greatly improve the fairness of the original ML software and slightly damage its performance.

RQ2: How well does LTDD perform compared with the state-of-the-art fairness algorithms?

Motivation and Approach. We compare our method LTDD with the state-of-the-art algorithms to see if our method can perform better than these algorithms. We hope that our method can achieve that, compared with baselines, (a) it can better improve the fairness of ML software; (b) the negative impact of our algorithm on performance would be less or comparable. Similar to RQ1, this RQ applies the logistic regression model as the main classifier and repeats the experiment 100 times to eliminate the influence of randomness.

Results for fairness. The comparison results under fairness metrics are shown in Table 2. The gray background in the table indicates that our method has won the result of the baseline (the Wilcoxon p-value is less than 0.05, and the Cliff's delta δ is greater than 0.147). The black background indicates that our method loses to the baseline (the Wilcoxon p-value is less than 0.05, and the Cliff's delta δ is less than -0.147). The white background indicates a tie. It can be seen that: (1) Our method is superior to other baselines in most cases in terms of average values (**less** values means more fairness). (2) Our method has won the baselines in 37 out of 55 cases under DI and 38 out of 55 cases under SPD. (3) Compared with the best baseline Fair-Smote, LTDD has comparable performance under DI (3 wins and 3 losses), while it performs better under SPD (5 wins and 3 losses). We conclude that our method significantly improves the fairness of ML software and surpasses the baselines in most cases.

Results for performance. We list the comparison results under performance indicators in Table 3. To compare the effects of different methods on the performance loss, we compare our method and baselines with the results of the *original* classifiers without applying fairness algorithms. Generally, all methods have losses to the performance of the original classifiers. In terms of ACC, Recall, and

Table 2: Fairness comparison of our method and baselines with logistical regression. *Less* values means more fairness.

Indicators	Methods	Datasets											W/T/L
		Adult _Race	Adult _Sex	Compas _Race	Compas _Sex	German	Default	Heart	Bank	MEPS15	MEPS16	Student	
|1 - DI|	Original	0.5894	0.8531	0.7929	1.3061	0.1151	0.3139	0.5525	0.0446	0.6590	0.6946	0.1705	9/2/0
	Reweighing	0.2744	0.4533	0.1244	0.1278	0.0286	0.0866	0.4358	0.0423	0.3525	0.3193	0.1387	6/5/0
	DIR	0.6837	0.8787	0.8261	1.3117	0.9144	0.2740	0.5645	0.0999	0.6665	0.7041	0.1635	10/1/0
	Fairway	0.5099	nan*	0.5639	1.6904	0.1359	0.3071	0.5204	0.0466	0.6576	0.6953	0.1903	9/2/0
	Fair-Smote	0.2184	0.2655	0.0801	0.0851	0.1445	0.0655	0.4276	0.0451	0.1089	0.1792	0.1811	3/5/3
	LTDD	0.2027	0.2136	0.1381	0.0790	0.0286	0.0850	0.2866	0.0463	0.2688	0.2867	0.1686	
|SPD|	Original	0.0899	0.1659	0.2037	0.2605	0.1141	0.0279	0.3289	0.0227	0.1166	0.1115	0.0714	9/2/0
	Reweighing	0.0400	0.0653	0.0535	0.0494	0.0269	0.0064	0.2397	0.0181	0.0434	0.0343	0.0583	6/3/2
	DIR	0.1383	0.2101	0.2061	0.2604	0.0396	0.0226	0.3483	0.0426	0.1180	0.1138	0.0679	10/1/0
	Fairway	0.0598	0.0018	0.1828	0.2956	0.1327	0.0254	0.3114	0.0198	0.1124	0.1134	0.0793	8/2/1
	Fair-Smote	0.0789	0.1005	0.0372	0.0399	0.0780	0.0211	0.2438	0.0208	0.0112	0.0188	0.0741	5/3/3
	LTDD	0.0293	0.0272	0.0616	0.0347	0.0270	0.0059	0.1416	0.0224	0.0309	0.0296	0.0708	

The background color indicates the comparison result between **LTDD** and **baseline methods**. The gray background indicates that our method wins the baseline, that is to say, the p-value is less than 0.05 and the δ is greater than 0.147; the black background indicates that our method loses to the baseline, that is to say, the p-value is less than 0.05 and the δ value is less than -0.147; a white background indicates a tie.
*The classifier predicts all results for the privileged class as 0.

Table 3: Performance comparison of our method and baselines with logistical regression.

Indicators	Methods	Datasets											W/T/L
		Adult _Race	Adult _Sex	Compas _Race	Compas _Sex	German	Default	Heart	Bank	MEPS15	MEPS16	Student	
ACC	Original	0.8217	0.8217	0.6383	0.6403	0.6995	0.8086	0.8233	0.7977	0.8648	0.8614	0.9321	
	Reweighing	0.8212	0.8136	0.6404	0.6430	0.6864	0.8079	0.8347	0.7745	0.8623	0.8560	0.9342	0/6/5
	DIR	0.8206	0.8217	0.6404	0.6415	0.3143	0.8058	0.8327	0.7769	0.8644	0.8600	0.9362	0/8/3
	Fairway	0.7850	0.7607	0.6375	0.6383	0.7021	0.8068	0.8227	0.7703	0.8639	0.8577	0.9324	0/6/5
	Fair-Smote	0.7659	0.7540	0.6364	0.6282	0.6073	0.7042	0.8318	0.7778	0.8547	0.8522	0.9323	0/3/8
	LTDD	0.8211	0.8059	0.6392	0.6391	0.6863	0.8079	0.8218	0.7993	0.8605	0.8569	0.9287	0/7/4
Recall	Original	0.4187	0.4187	0.5618	0.5652	0.9697	0.2293	0.7714	0.8162	0.4024	0.3599	0.9074	
	Reweighing	0.4177	0.3660	0.5820	0.5872	0.9754	0.2255	0.7802	0.7231	0.3708	0.3220	0.9124	2/4/5
	DIR	0.5058	0.4810	0.5594	0.5638	0.0596	0.2159	0.7904	0.7496	0.3988	0.3546	0.9121	2/6/3
	Fairway	0.2782	0.0028	0.5978	0.5873	0.9520	0.2126	0.7894	0.7179	0.3896	0.3538	0.9119	2/3/6
	Fair-Smote	0.7403	0.7068	0.6257	0.6203	0.5885	0.6012	0.7870	0.7559	0.3599	0.3262	0.9102	5/2/4
	LTDD	0.4186	0.3405	0.5670	0.5906	0.9737	0.2257	0.7672	0.8149	0.3616	0.3177	0.9055	1/6/4
False Alarm	Original	0.0516	0.0516	0.2983	0.2978	0.9207	0.0259	0.1321	0.2187	0.0390	0.0377	0.0447	
	Reweighing	0.0520	0.0457	0.3103	0.3111	0.9785	0.0258	0.1181	0.1792	0.0359	0.0359	0.0449	4/4/3
	DIR	0.0803	0.0713	0.2927	0.2952	0.0766	0.0254	0.1265	0.1983	0.0396	0.0381	0.0405	2/7/2
	Fairway	0.0553	0.0006	0.3310	0.3187	0.0480	0.0239	0.1482	0.1826	0.0382	0.0400	0.0478	4/3/4
	Fair-Smote	0.2261	0.2312	0.3503	0.3620	0.3462	0.2665	0.1290	0.2026	0.0427	0.0418	0.0463	2/2/7
	LTDD	0.0524	0.0479	0.3002	0.3209	0.9746	0.0259	0.1301	0.2146	0.0358	0.0346	0.0496	3/6/2

The background color indicates the comparison result between **fairness methods** and the **original method**. The gray background indicates that the fairness method wins the original method, that is to say, the p-value is less than 0.05 and the δ is greater than 0.147; the black background indicates that the fairness method loses to the original method, that is to say, the p-value is less than 0.05, and the δ is less than -0.147; a white background indicates a tie.

False Alarm, our method significantly damages the performance of the original model in 10 out of 33 cases, which is less than 3 out of 4 baselines. Besides, our method has only 2 cases with significant damage under False Alarm, which is the best among all fairness methods. It can be said that our method maintains the original performance in most cases, i.e., the negative impact of our algorithm on performance is less or comparable.

> Answer to RQ2: Our method performs better than baselines in the improvement of fairness indicators with the performance damage less than or comparable to baselines.

RQ2a: How well does our method compare with the state-of-the-art fairness algorithms under different classifiers?

Table 4: Fairness comparison between our method and baselines with three classifiers.

	vs. Reweighing		vs. DIR		vs. Fair-Smote		vs. Original	
	DI	SPD	DI	SPD	DI	SPD	DI	SPD
WIN	17	17	29	28	12	17	27	27
TIE	11	10	4	5	11	9	6	6
LOSS	5	6	0	0	10	7	0	0

Because Fairway's model optimization part is designed for logistic regression models, we do not compare with Fairway here.

Motivation and Approach. To verify the generalization of LTDD, we also conduct our experiments under two other classifiers NB and SVM. Similarly, we run the experiment 100 times and compare it with baselines under fairness and performance indicators.

Table 5: Comparison of performance indicators between original classifiers and fairness methods in 11 scenarios for 3 classifiers

Methods	Reweighing vs. original			DIR vs. original			Fair-Smote vs. original			LTDD vs. original		
Indicators	ACC	Recall	False Alarm	ACC	Recall	False Alarm	ACC	Recall	False Alarm	ACC	Recall	False Alarm
WIN	5	3	14	2	5	5	5	17	8	4	3	13
TIE	18	16	13	21	20	19	12	6	5	18	14	14
LOSS	10	14	6	10	8	9	16	10	20	10	16	6

Results for fairness. We show the comparison of fairness metrics in Table 4. We count the number of win, tie, and loss cases compared with the baselines and summarize the results of the three classifiers together. When the number of wins is greater than or equal to the number of ties plus losses, we mark this column with the gray background, which means that our method is significantly better than the baseline. It can be seen that in 7 of 8 scenarios, our method is significantly better than baselines. When compared with Fair-Smote, our results on DI are similar, but our number of wins is still slightly higher than Fair-smote.

Results for performance. We list the comparison results of the fairness indicators in Table 5. We compare the four methods with the results of the original classifier and count the number of W/T/L. It can be seen that our method has the least number of losses in terms of ACC and False Alarm compared with the original classifier: there are 10 and 6 cases that produce losses, respectively.

> Answer to RQ2a: With three different classifiers, our method still defeats baselines in most cases under fairness indicators and also has the least loss under ACC and False Alarm.

RQ3: Is our method actionable for fairness improvement in realistic scenarios?

Motivation and Approach. In previous RQs, we have compared the prediction results of our method and baselines, and the results have shown that our method can generate fairer predictions. In this RQ, we introduce a new indicator, *the rate of favorable decisions*, to measure the actionability of fairness methods. Our measurement is based on the following points: (a) in realistic scenarios, the rate of favorable decisions is limited due to the finite social resources, e.g., for the loan application, the rate of "approval" is restricted; (b) before applying fairness methods, original ML software has made favorable and unfavorable decisions, among which the original favorable rate could be considered as a benchmark; (c) after applying fairness methods, ML software might change the favorable rate. If the favorable rate essentially increases, it would also raise the need for social resources, even beyond the boundaries of available social resources, reducing actionability. In this RQ, we will compare our method with Fair-Smote under the new indicator.

Results. The result of the comparison is shown in Figure 4, where we employ two endpoints to indicate the favorable rates of the privileged and non-privileged classes and connect them to draw a line. The short horizontal stroke indicates the whole favorable rate. The figure's red, blue, and green points represent the original value, the result of Fair-smote, and that of LTDD, respectively.

It can be seen that in most (10 out of 11) cases, the way that LTDD improves fairness is to increase the favorable rate of non-privileged classes and to reduce the favorable rate of privileged classes, so that the whole favorable rate is very *close* to the original value. This means that our method does not need more social resources to achieve in most cases. On the contrary, Fair-Smote tries to increase the favorable rate of privileged and non-privileged classes, resulting in a significant increase of the whole favorable rate. For example, on the Adult data set, the whole favorable rate of Fair-Smote increases about 1.5 times. This means that the implementation of Fair-Smote needs much more social resources.

> Answer to RQ3: Our method improves the fairness indicators while ensuring that the whole favorable rate is close to the original value and does not need more social resources.

6 DISCUSSION

We further discuss the aims and results of our method LTDD in the following six points.

6.1 Distribution of biased features

As described in Sections 3.2 and 3.3, we judge whether features are biased according to the p-value generated by Wald test [30] of linear regression model. Here we calculate the distribution of biased features with p-value < 0.05.

Figure 5 shows the average proportion of biased features in 11 scenarios, where the blue bars indicate the proportion. We can see that in 8 out of the 11 scenes, the percents of biased features are over 60%. In the COMPAS data set, when the race is used as a sensitive feature, all other features are considered biased, which means that the high association with the sensitive feature is universal in the data set. Besides, we have also observed that the proportions of biased features in the three data sets, Bank, Student, and German, are the lowest three. According to the results of RQ1, the fairness optimization effect of these three data sets is also the worst among all data sets. We argue that the proportion of biased features may be related to the fairness improvement effect of LTDD.

6.2 A case study on the COMPAS dataset

This section will use an example to show how our method eliminates biased parts from biased features, narrowing the difference in these biased features between privileged and unprivileged classes.

As shown in Figure 6, this is the distribution of the "Age" feature for privileged (white) and unprivileged (non-white) categories on the COMPAS dataset. The orange dotted line represents the median value, the purple triangle represents the average value, and the green diamonds indicate the outliers.

Figure 6(a) is the original distribution of the "Age" feature. It can be seen that the mean and median values of "Age" of white

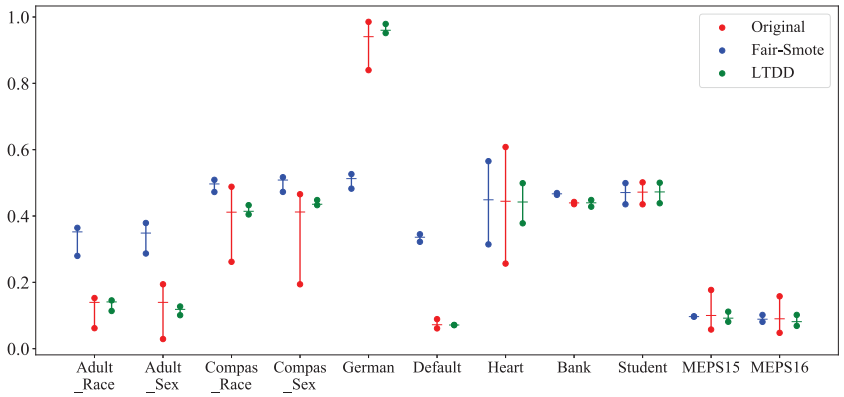


Figure 4: Comparison of our approach with Fair-Smote on favorable rates. The favorable rates of privileged and non-privileged classes are represented by two endpoints, while the whole favorable rate is represented by the short horizontal stroke.

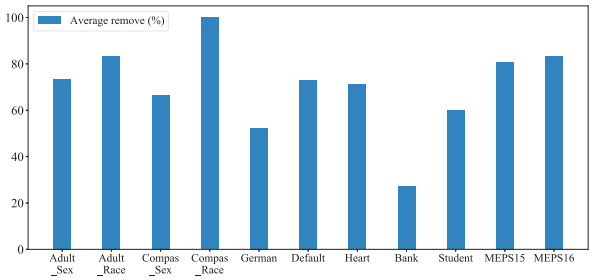
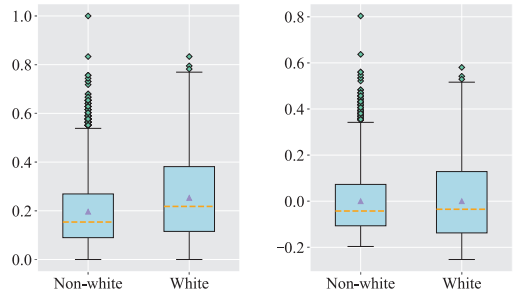


Figure 5: Distribution (%) of biased non-sensitive features.



(a) Before (b) After

Figure 6: An example to show how LTDD revises biased features on COMPAS dataset. Figures 6(a) and 6(b) show the original and revised Age distribution (by normalizing into $[0, 1]$) of Non-white and White on COMPAS dataset

people are obviously higher than that of non-white people. After removing the association between the "Age" feature and the "Race" feature, the distributions of "Age" between the two groups are shown in Figure 6(b). It can be seen that both sets of values have been "shifted", which makes the distribution between the two groups closer. From the mean and median, we can see that after removing biased parts, our method *narrows* the distribution difference between the privileged group and the unprivileged group.

Table 6: Comparison of AOD and EOD between our method and Fair-Smote in 11 scenarios for 3 classifiers

Models		LSR		NB		SVM	
		mean	W/T/L	mean	W/T/L	mean	W/T/L
AOD	Fair-Smote	0.0493	5/3/3	0.0535	3/7/1	0.1077	4/4/3
	LTDD	0.0429		0.0476		0.0991	
EOD	Fair-Smote	0.0709	4/4/3	0.0580	2/8/1	0.0645	2/4/5
	LTDD	0.0653		0.0573		0.0613	

6.3 Performance under other fairness indicators

As observed in previous studies [18], there may be conflicts between different fairness indicators. To verify the performance of our method under other fairness indicators, we compare our method with the state-of-the-art method Fair-Smote method under other two indicators AOD and EOD [15] with three classifiers.

The difference between the results of our method and Fair-Smote is shown in Table 6. As can be seen from the table, our method is slightly better than Fair-Smote in most scenarios. The average value of the two indicators of our method in the three classifiers is lower (lower means more fairness) than Fair-Smote. Except for the EOD indicator under SVM, the number of wins for our method is greater than the number of losses. Combined with the conclusion in RQ2, it can be seen that our method has a greater improvement under other fairness indicators compared to Fair-Smote.

6.4 Differences between DI and SPD

Intuitively, DI and SPD are two similar fairness metrics. In this subsection, we will conduct additional analysis to show the differences between DI and SPD.

(a) We give a case study to show the differences between DI and SPD. Given two groups A,B and their favorable rates under two fairness methods f_1,f_2: $r_A^1 = 0.32$ and $r_B^1 = 0.2$ for f_1, and $r_A^2 = 0.6$ and $r_B^2 = 0.4$ for f_2. According to the calculation of DI and SPD, we

obtain the following results:

$$\begin{aligned} \mathrm{DI}(f_1) &= r_A^1/r_B^1 = 1.6 \\ \mathrm{DI}(f_2) &= r_A^2/r_B^2 = 1.5 \\ \mathrm{SPD}(f_1) &= r_A^1 - r_B^1 = 0.12 \\ \mathrm{SPD}(f_2) &= r_A^1 - r_B^1 = 0.2 \end{aligned}$$

We would draw different conclusions from the comparison between f_1 and f_2 under DI and SPD: for DI, f_2 performs better than f_1 (more close to 1 means better); for SPD, f_1 performs better than f_2 (more close to 0 means better).

(b) We compute the Pearson correlation coefficient between the values of DI and SPD in 11 studied scenarios for five methods. We observe that the coefficient is very small in some scenarios, e.g., 0.08 for the German dataset, which means the values of DI and SPD do not have high correlations. Hence, one of them cannot be directly substituted for the other.

6.5 Performance with other association approaches?

This paper employs the linear regression model to obtain the association between features. It is unclear how other association approaches work. In this section, we conduct an experiment with another association approach, polynomial regression, which is an extension of simple linear regression:

$$x_n = a + b_1 \cdot x_s + b_2 \cdot x_s^2 + \cdots + b_d \cdot x_s^d$$

where x_s and x_n denote sensitive and non-sensitive features. To make the fitting result have a large difference from the linear regression, we set the degree (d) of the polynomial to 20 (linear regression could be considered as a specific polynomial regression when $d = 1$).

We compare the results of linear regression and polynomial regression under fairness indicators. We observe that in most cases, the results between the two methods are comparable, and the results of linear regression are significantly better/worse in 4/1 scenarios.

6.6 Insight of our method

Our method LTDD explores a simple but effective model, linear regression, to identify, estimate, and remove the high association between non-sensitive features and sensitive features as the biased parts of the non-sensitive features. The results of our experiments support the claim that LTDD can largely improve the fairness of ML software, with less or comparable damage to its performance. Our study may lead to three insights for the following studies in the fairness of ML software:

(a) The distribution and association of feature values in training data would be helpful to estimate and improve the fairness of ML software. We encourage following researchers to employ more effective methods to model the feature distribution and association.

(b) Although ML software is much different from traditional software, the research methods in traditional SE are also applicable to the related research topic of ML software and can achieve good performance. We encourage following researchers to apply more traditional SE methods and ideas to the research of ML software.

(c) For fairness research, our study provides a new practical perspective: we employ the rate of favorable decisions to measure the actionability of fairness methods, which indicates the actual cost of social resources for fairness methods. We hope that future researchers can take into account the actual cost while focusing on fairness.

7 THREATS TO VALIDITY

In this section, we discuss the threats to validity of our approach.

First of all, the choice of data set may be a threat. We employ 9 data sets widely used in fairness testing to evaluate our method. However, these data sets may not be sufficient, because the distribution and characteristics of different data sets are not the same. We will introduce more datasets to test our method in the future.

Second, the choice of classifiers may be a threat. In the experiment, we mainly use the logistic regression model for fairness testing, which is used in many fairness studies, but the influence of the method on different learners may be different. Although we also test the performance under the other two learners, there are still more complex learners worth experimenting with. In the future, we will test how our method performs in complex neural networks.

Finally, the fairness indicators we use may also be a threat. Previous studies have pointed out that there may be contradictions between different fairness indicators. There are so many fairness indicators proposed at present, and it is impossible to satisfy all fairness indicators [24]. Therefore, the indicators should be selected reasonably according to the actual scenario. In the paper, we mainly discuss the results under DI and SPD, and check the effect of our method under other indicators in the discussion.

8 CONCLUSION

The widespread usage of machine learning software has raised concerns about its fairness. More and more SE researchers have paid attention to the fairness of ML software. The training set is considered to be one of the reasons for the bias of ML software. This paper adopts a new perspective: the root cause of the unfairness could be biased features in training data.

To obtain features unbiased, we try to debug feature values in training data: identify which features and which parts of them are biased and exclude the biased parts of such features. To achieve this goal, we propose a novel method, **L**inear-regression based **T**raining **D**ata **D**ebugging (**LTDD**), which employs the linear regression model to identify the biased features. We introduce nine datasets to evaluate our method and compare our methods with four baselines. The results show that our method performs better in improving the fairness indicators, and damages the performance slightly. In the future, we hope to conduct research on more extensive datasets to explore different characteristics of the datasets.

8.1 Replication Package

We provide all datasets and source code used to conduct this study at **https://github.com/fairnesstest/LTDD**.

ACKNOWLEDGEMENTS

The work is supported by the National Natural Science Foundation of China (Grant No. 62172202, 61872177, 61772259, 62172205, 61832009, 61772263).

REFERENCES

[1] 1988. Heart Disease Data Set. https://archive.ics.uci.edu/ml/datasets/heart+disease.

[2] 1994. Statlog (German Credit Data) Data Set. https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data).

[3] 1996. Adult Data Set. https://archive.ics.uci.edu/ml/datasets/adult.

[4] 2011. The algorithm that beats your bank manager. https://www.forbes.com/sites/parmyolson/2011/03/15/the-algorithm-that-beats-your-bank-manager/#15da2651ae99.

[5] 2012. Bank Marketing Data Set. https://archive.ics.uci.edu/ml/datasets/Bank+Marketing.

[6] 2014. Student Performance Data Set. https://archive.ics.uci.edu/ml/datasets/Student+Performance.

[7] 2015. MEPS Data Set. https://meps.ahrq.gov/mepsweb.

[8] 2016. Amazon just showed us that unbiased algorithms can be inadvertently racist. https://www.businessinsider.com/how-algorithms-can-be-racist-2016-4.

[9] 2016. default of credit card clients Data Set. https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients.

[10] 2017. compas-analysis. https://github.com/propublica/compas-analysis.

[11] Aniya Aggarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. 2019. Black box fairness testing of machine learning models. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 625–635. https://doi.org/10.1145/3338906.3338937

[12] Sousuke Amasaki and Chris Lokan. 2015. On the effectiveness of weighted moving windows: Experiment on linear regression based software effort estimation. *Journal of Software: Evolution and Process* 27, 7 (2015), 488–507.

[13] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[14] Solon Barocas and Andrew D Selbst. 2016. Big data's disparate impact. *Calif. L. Rev.* 104 (2016), 671.

[15] R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilovi, S. Nagar, K. Natesan Ramamurthy, J. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang. 2019. AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. *IBM Journal of Research and Development* 63, 4/5 (2019), 4:1–4:15. https://doi.org/10.1147/JRD.2019.2942287

[16] 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 T. Richards, Diptikalyan Saha, Prasanna Sattigeri, Moninder Singh, Kush R. Varshney, and Yunfeng Zhang. 2018. AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias. *CoRR* abs/1810.01943 (2018). arXiv:1810.01943 http://arxiv.org/abs/1810.01943

[17] Richard Berk, Hoda Heidari, Shahin Jabbari, Matthew Joseph, Michael Kearns, Jamie Morgenstern, Seth Neel, and Aaron Roth. 2017. A convex framework for fair regression. *arXiv preprint arXiv:1706.02409* (2017).

[18] Sumon Biswas and Hridesh Rajan. 2020. Do the machine learning models on a crowd sourced platform exhibit bias? an empirical study on model fairness. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 642–653.

[19] Sumon Biswas and Hridesh Rajan. 2020. Do the machine learning models on a crowd sourced platform exhibit bias? an empirical study on model fairness. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 642–653. https://doi.org/10.1145/3368089.3409704

[20] Yuriy Brun and Alexandra Meliou. 2018. Software fairness. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 754–759.

[21] Toon Calders and Sicco Verwer. 2010. Three naive Bayes approaches for discrimination-free classification. *Data Min. Knowl. Discov.* 21, 2 (2010), 277–292. https://doi.org/10.1007/s10618-010-0190-x

[22] Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. 2021. Bias in machine learning software: why? how? what to do?. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 429–440. https://doi.org/10.1145/3468264.3468537

[23] Joymallya Chakraborty, Suvodeep Majumder, Zhe Yu, and Tim Menzies. 2020. Fairway: a way to build fair ML software. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 654–665. https://doi.org/10.1145/3368089.3409697

[24] Joymallya Chakraborty, Kewen Peng, and Tim Menzies. 2020. Making Fair ML Software using Trustworthy Explanation. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 1229–1233. https://doi.org/10.1145/3324884.3418932

[25] Joymallya Chakraborty, Tianpei Xia, Fahmid M Fahid, and Tim Menzies. 2019. Software engineering for fairness: A case study with hyperparameter optimization. *arXiv preprint arXiv:1905.05786* (2019).

[26] L. De Capitani and D. De Martini. 2011. On stochastic orderings of the Wilcoxon Rank Sum test statistic–With applications to reproducibility probability estimation testing. *Statistics & Probability Letters* 81, 8 (2011), 937–946. https://doi.org/10.1016/j.spl.2011.04.001

[27] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and Removing Disparate Impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams (Eds.). ACM, 259–268. https://doi.org/10.1145/2783258.2783311

[28] Joseph L Gastwirth. 1988. A clarification of some statistical issues in Watson v. Fort Worth Bank and Trust. *Jurimetrics J.* 29 (1988), 267.

[29] Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of Opportunity in Supervised Learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 3315–3323. https://proceedings.neurips.cc/paper/2016/hash/9d2682367c3935defcb1f9e247a97c0d-Abstract.html

[30] Georg Heinze and Michael Schemper. 2002. A solution to the problem of separation in logistic regression. *Statistics in medicine* 21, 16 (2002), 2409–2419.

[31] Faisal Kamiran and Toon Calders. 2011. Data preprocessing techniques for classification without discrimination. *Knowl. Inf. Syst.* 33, 1 (2011), 1–33. https://doi.org/10.1007/s10115-011-0463-8

[32] Lukas Kirschner, Ezekiel O. Soremekun, and Andreas Zeller. 2020. Debugging inputs. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 75–86. https://doi.org/10.1145/3377811.3380329

[33] Yibin Liu, Yanhui Li, Jianbo Guo, Yuming Zhou, and Baowen Xu. 2018. Connecting software metrics across versions to predict defects. In *25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018*, Rocco Oliveto, Massimiliano Di Penta, and David C. Shepherd (Eds.). IEEE Computer Society, 232–243. https://doi.org/10.1109/SANER.2018.8330212

[34] Linghan Meng, Yanhui Li, Lin Chen, Zhi Wang, Di Wu, Yuming Zhou, and Baowen Xu. 2021. Measuring Discrimination to Boost Comparative Testing for Multiple Deep Learning Models. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 385–396. https://doi.org/10.1109/ICSE43902.2021.00045

[35] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding faster configurations using flash. *IEEE Transactions on Software Engineering* 46, 7 (2018), 794–811.

[36] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, Jeff Skowronek, and Linda Devine. 2006. Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's d indices the most appropriate choices. In *annual meeting of the Southern Association for Institutional Research*. Citeseer, 1–51.

[37] Ricardo Salazar, Felix Neutatz, and Ziawasch Abedjan. 2021. Automated feature engineering for algorithmic fairness. *Proceedings of the VLDB Endowment* 14, 9 (2021), 1694–1702.

[38] Qinbao Song, Martin Shepperd, Michelle Cartwright, and Carolyn Mair. 2006. Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering* 32, 2 (2006), 69–82.

[39] Chad D. Sterling and Ronald A. Olsson. 2007. Automated bug isolation via program chipping. *Softw. Pract. Exp.* 37, 10 (2007), 1061–1086. https://doi.org/10.1002/spe.798

[40] Latanya Sweeney. 2013. Discrimination in online ad delivery. *Commun. ACM* 56, 5 (2013), 44–54.

[41] Chakkrit Tantithamthavorn, Ahmed E Hassan, and Kenichi Matsumoto. 2018. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Transactions on Software Engineering* 46, 11 (2018), 1200–1219.

[42] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 98–108. https://doi.org/10.1145/3238147.3238165

[43] Weiyuan Wu, Lampros Flokas, Eugene Wu, and Jiannan Wang. 2020. Complaint-driven Training Data Debugging for Query 2.0. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1317–1334. https://doi.org/10.1145/3318464.3389696

[44] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna P Gummadi. 2017. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th international conference on world wide web*. 1171–1180.

[45] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. 2018. Mitigating Unwanted Biases with Adversarial Learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, AIES 2018, New Orleans, LA, USA, February 02-03, 2018*, Jason Furman, Gary E. Marchant, Huw Price, and Francesca Rossi (Eds.). ACM, 335–340. https://doi.org/10.1145/3278721.3278779

[46] Jie M. Zhang and Mark Harman. 2021. "Ignorance and Prejudice" in Software Fairness. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1436–1447. https://doi.org/10.1109/ICSE43902.2021.00129

[47] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* (2020).

[48] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. 2020. White-box fairness testing through adversarial sampling. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 949–960. https://doi.org/10.1145/3377811.3380331

[49] Yuming Zhou, Hareton Leung, and Baowen Xu. 2009. Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. *IEEE Transactions on Software Engineering* 35, 5 (2009), 607–623.

[50] Yuming Zhou, Baowen Xu, Hareton Leung, and Lin Chen. 2014. An in-depth study of the potentially confounding effect of class size in fault prediction. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 1 (2014), 1–51.