Functional Dependencies to Mitigate Data Bias

(Discussion Paper)

Fabio Azzalini[1,2], Chiara Criscuolo[1] and Letizia Tanca[1]

[1]*Politecnico di Milano - Dipartimento di Elettronica, Informazione e Bioingegneria*
[2]*Human Technopole - Center for Analysis, Decisions and Society*

Abstract

Technologies based on data are frequently adopted in many sensitive environments to build models that support important and life-changing decisions. As a result, for an application to be ethically reliable, it should be associated with tools to discover and mitigate bias in data, in order to avoid (possibly unintentional) unethical behaviors and the associated consequences.

In this paper we propose a novel solution that, exploiting the notion of Functional Dependency and its variants - well-known data constraints - aims at enforcing fairness by discovering and solving discrimination in datasets. Our system first identifies the attributes of a dataset that encompass discrimination (e.g. gender, ethnicity or religion), generating a list of dependencies, then, based on this information, determines the smallest set of tuples that must be added or removed to mitigate such bias in the dataset.

Experimental results on two real-world datasets demonstrated that our approach can greatly improve the ethical quality of data sources.

Keywords

Fairness, Data Bias, Functional Dependencies

1. Introduction

In the recent years fairness has become an important topic of interest in the Data Science community. Indeed, computers and algorithms have made our lives efficient and easier, but among the prices we risk to pay is the possible presence of discrimination and unfairness in the decisions we make with their support. As a result, algorithmic decision systems should work on unbiased data to obtain fair results, since learning from historical data might mean to learn also traditional prejudices that are endemic in society, producing unethical decisions.

Last year we presented FAIR-DB [1] [2], a framework that, by discovering and analyzing a particular type of database integrity constraints can find unfair behaviors in a dataset. Specifically, the system employees approximate conditional functional dependencies (ACFDs) to recognize the cases where the values of a certain attributes (e.g. gender, ethnicity or religion) *frequently determines* the value of another one (such as range of the proposed salary or social state). In this paper we enhance the framework with an additional functionality to repair datasets found unfair by the discovery phase of FAIR-DB. The new module, *ACFD-Repair method*, is a procedure that, based on the discovered dependencies, determines the smallest set of tuples to be added or removed from the dataset to mitigate, or completely, remove the previously discovered bias.

SEBD 2022: The 30th Italian Symposium on Advanced Database Systems, June 19-22, 2022, Tirrenia (Pisa), Italy

fabio.azzalini@polimi.it (F. Azzalini); chiara.criscuolo@polimi.it (C. Criscuolo); letizia.tanca@polimi.it (L. Tanca)
0000-0003-0631-2120 (F. Azzalini); 0000-0003-2607-3171 (L. Tanca)

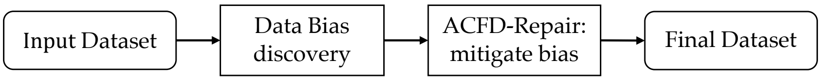

Figure 1: Framework workflow

ACFD	Support	Difference
(Sex = ‘Female’) → Income = ‘≤50K’	0.287	0.135
(Race = ‘Black’) → Income = ‘≤50K’	0.081	0.119
(Race = ‘Amer-Indian-Eskimo’) → Income = ‘≤50K’	0.008	0.129
(Native-Country = ‘NC-Hispanic’) → Income = ‘≤50K’	0.044	0.158

Table 1
A few user-selected dependencies from the U.S. Census Adult Dataset

2. Framework Overview

Before presenting the methodology, we first introduce some fundamental notions that will accompany us along our discussion. A *protected attribute* is a characteristic for which non-discrimination should be established, like age, race, sex, etc [3]; the *target variable* is the feature of a dataset about which the user wants to gain a deeper understanding, for example the income, or a boolean label that indicates whether a loan is authorized or not, etc.

Figure 1 shows the framework of the enhanced version of FAIR-DB.

We now give a brief presentation of the *Data Bias discovery* step. For a deeper understanding of this phase, we refer the reader to the following papers: [1], [2]. The *Data Bias discovery* module operates according to the following steps: we import the data and perform (if needed) data cleaning and integration, we extract the ACFDs from the dataset and discard the ones that do not contain protected attributes and the target variable, then, to select the ACFDs that show unfair behaviours for each dependency, we compute some metrics capturing the “ethical level” of the dependency, to facilitate user interaction we rank the ACFDs in descending order of importance; finally the user selects the ACFDs she perceives as the most problematic and then, the system computes some metrics that summarize the level of unfairness of the dataset.

The core of the discovery phase is the *Difference*, a novel metric that indicates how much *‘unethical’* the behaviour highlighted by a dependency is. The higher this metric, the stronger the alert for an unfair behavior.

For each dependency ϕ, we define the *Difference* metric of ϕ as the difference between the dependency confidence and the confidence of the dependency computed without the protected attributes of the Left-Hand-Side (LHS) of the ACFD. Given a dependency in the form ϕ:$(X \rightarrow Y, t_p)$, let $Z = (X - \{\mathit{ProtectedAttributes}\})$, that is the LHS of the ACFD without its *protected attributes*, and ϕ': $(Z \rightarrow Y, t_p)$. We define the *Difference* as:

$$\text{Difference}(\phi) = \text{Confidence}(\phi) - \text{Confidence}(\phi')$$

The *Difference* metric gives us an idea of how much the values of the protected attributes influence the value of Y. In order to assess the unfair behavior of a dependency, we also take into consideration its support, that indicates the *pervasiveness* of the ACFD; unethical dependencies with high support will impact many tuples, and thus will be more important.

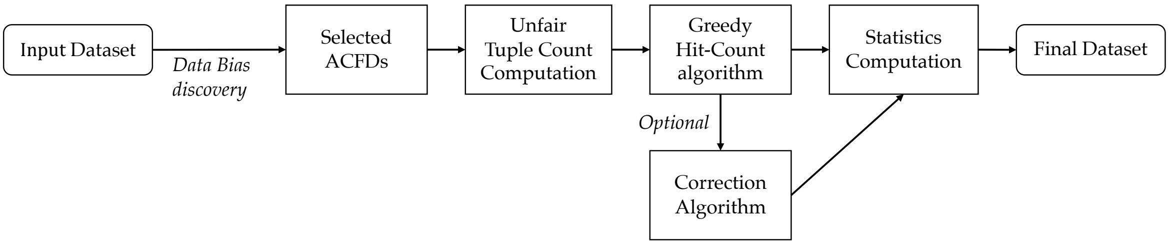

Figure 2: ACFD-Repair pipeline

Table 1 reports a few user-selected ACFDs mined from the U.S. Census Adult Dataset[1]. An analysis of the dependencies shows that the groups more discriminated are: 'Female', 'Black', 'NC-Hispanic' and 'Amer-Indian-Eskimo'.

3. ACFD-Repair

The final objective of the repair phase is to create a fair dataset where the unethical behaviors are greatly mitigated or completely removed. The procedure has to conform to a fundamental requirement, i.e., since the repaired dataset has to be used later on in a data science pipeline, the number of modifications, consisting of deletion or addition of tuples, has to be minimized in order to protect the original distributions of values. Note also that, given the variety of decision algorithms, the repair procedure must work also for attributes with non-binary values. Figure 2 presents the *ACFD-Repair* methodology: we now give a detailed description of each phase using the U.S. Census Adult dataset as running example.

3.1. Unfair Tuple Count Computation

We propose to try to reduce, or eliminate, unfairness, starting from the list of ACFDs computed in the *Data Bias discovery* step. The repair is performed by bringing the value of the Difference of each user-selected dependency below a minimum threshold δ; this can be achieved in two ways: *(i)* removing the tuples matching the dependency so that, after removal, the Difference value of the dependency will become lower than δ; *(ii)* adding the tuples that, combined with the elements that match the dependency, if added, will lower its initial Difference value below δ.

The Unfair Tuple Count computation step is therefore responsible for finding, for each dependency, the number of tuples that has to be added or removed. For each ACFD ϕ, we define its Unfair Tuple Count as $F(\phi) = (\alpha,\beta)$ where α represents the number of tuples that should be added, and β the number of tuples that should be removed to repair the discrimination behaviour shown by the dependency.

To explain how the Unfair Tuple Count value of each dependency is computed, we make use of the dependency ϕ_1 : *(Sex = 'Female', Workclass = 'Private')* → *Income = '≤50K'*. The Difference of ϕ_1 can be computed as: $\text{Diff}(\phi_1) = \text{Conf}(\phi_1) - \text{Conf}(\phi_1')$, where ϕ_1': *(Workclass = 'Private')* → *(Income = '≤50K')*. Rewriting the confidence as a ratio of supports we get the following formula:

$$\text{Diff}(\phi_1) = \frac{\text{Sup}(\phi_1)}{\text{Sup}(\text{LHS}(\phi_1))} - \frac{\text{Sup}(\phi_1')}{\text{Sup}(\text{LHS}(\phi_1'))} = \frac{a}{b} - \frac{c}{d}.$$

[1] https://archive.ics.uci.edu/ml/datasets/Adult

As already mentioned, to repair ϕ_1 we can either add or remove tuples: let's first focus on the former. To remove the unfair behaviour highlighted by ϕ_1: *(Sex = 'Female', Workclass = 'Private') → Income = '≤50K'*, intuitively, we can either add males that earn less than 50K \$/year and work in the private sector or add females that earn more than 50K \$/year and work in the private sector. Therefore we have to add to the dataset tuples that satisfy the following two dependencies:

- ϕ_2: *(Sex = 'Male', Workclass = 'Private') → Income = '≤50K'*
- ϕ_3: *(Sex = 'Female', Workclass = 'Private') → Income = '>50K'*

The two dependencies above define the tuples that, if added to the initial dataset, will lower the initial Difference value below the threshold δ; ϕ_2 and ϕ_3, being the "antagonists" of ϕ_1, form the *Opposite Set* (OS) of ϕ_1. Specifically, ϕ_2 has been obtained by changing, one at the time, the values of the protected attributes: we name the set of such ACFDs *Protected attribute Opposite Set* (POS). In the example we have only one, binary protected attribute, therefore the POS will contain only the dependency ϕ_2; in case the protected attribute is non-binary, or there is more than one protected attribute, the POS would include more than one dependency. On the other hand, ϕ_3 has been obtained by reversing the value of the target class. This ACFD is in the *Target attribute Opposite Set* (TOS), that, since the target attribute is required to be binary, will always contain only one dependency.

In case we have multiple discriminated minorities, adding tuples that satisfy the dependencies contained in the POS could result in enhancing discrimination. If we consider ϕ: *(Race= 'Black') → Income = '≤50K'*, the POS contains the ACFDs that involve all the other values of the protected attribute *'Race'*: 'White', 'Asian-Pac-Islander', 'Other' and 'Amer-Indian-Eskimo'; therefore we would add tuples referring to 'Amer-Indian-Eskimo' people that earn less than 50K \$/year to the dataset, thus aggravating their already discriminatory condition.

Finally, given what we have just presented and the fact that we want to repair the dataset by improving the condition of discriminated groups, we decide to add, for each ACFD, only the tuples that satisfy the dependency contained in each respective TOS.

We now show how to compute the number of tuples to be added to the dataset for the dependencies belonging to the TOS: we start by analyzing ϕ_3, obtained by reversing the value of the target attribute. Adding k tuples that satisfy ϕ_3 will impact the Difference of ϕ_1 in the following way:

$$\text{Diff}(\phi_1) = \frac{a}{b+k} - \frac{c}{d+k}$$

Now, to repair ϕ_1, we have to impose that Diff(ϕ_1) be lower than δ, and, solving the inequality, we find the number of tuples that need be added by solving for k.

Now that we demonstrated how to find the number of tuples to be added in order to repair a dependency, we show how to find the number of tuples that need be removed to accomplish the same task. To repair an ACFD by removing tuples, we simply need to remove tuples that satisfy the dependency. To compute the number of tuples to be deleted from the dataset, we start by noticing that removing from the dataset k tuples that satisfy ϕ_1 will modify the Difference of ϕ_1 in the following way:

$$\text{Diff}(\phi_1) = \frac{a-k}{b-k} - \frac{c-k}{d-k}$$

Now, to repair ϕ_1, we have to impose that Diff(ϕ_1) has to be lower than δ, and, similarly to what we have done in the previous two cases, find the number of tuples that should be removed by solving for k.

ACFD	Tuples to add	Tuples to remove
(Sex = 'Female') → Income = '≤50K'	1141	4883
(Race= 'Black') → Income = '≤50K'	185	852
(Race = 'Amer-Indian-Eskimo') → Income = '≤50K'	21	96
(Native-Country = 'NC-Hispanic') → Income = ≤50K	162	734

Table 2
A few user-selected dependencies from the U.S. Census Dataset with the corresponding values of *(α, β)*

Example 1. *For the repair phase, we continue using the U.S. Census Dataset. Table 1 report some examples of selected ACFDs to repair the dataset. Table 2 reports, for each selected ACFD, the number of tuples that should be added according to the TOS computation, or removed to compensate the Difference metric.*

3.2. Greedy Hit-Count algorithm

A basic solution to repair the dataset could simply consist in adding, for each negative ACFD, the tuples that satisfy its TOS, using, for each tuple, its Unfair Tuple Count as multiplicity. Unfortunately, this is not a valid option, because it would result in adding too many tuples to the dataset, which would not respect the key requirement to limit the modifications of the original dataset.

To add tuples to the initial dataset in an optimized way, we use a modified version of the *Greedy Hit-Count algorithm* algorithm presented in [4]. From each dependency in a TOS, we generate a *pattern*, which is an array whose dimension is equal to the number of attributes in the dataset and where each cell represents the value of a specific attribute. Given a dataset D with d categorical attributes and an ACFD ϕ, a pattern P is a vector of size d generated from ϕ, where $P[k]$ is either X (meaning that its value is unspecified) or the value of the corresponding attribute in ϕ. The pattern computation step is useful for two reasons: *(i)* to transform an ACFD into a tuple and decide the value of the free attributes (i.e. the non-instantiated attributes in the dependency); and *(ii)* to add tuples to the dataset in an optimized way, exploiting the Greedy Hit-Count algorithm.

The *Greedy Hit-Count algorithm* takes as input the set of patterns computed at the previous step and returns the set of tuples needed to repair the dataset. The idea behind this algorithm is that a value combination (i.e. a generated tuple) can cover multiple patterns simultaneously, allowing us to add fewer tuples than the basic solution, and thus minimizing the changes in the final, repaired dataset. The algorithm stops when all the patterns are covered, returning the values combinations that hit the maximum number of uncovered patterns. To summarize, given the set of uncovered patterns as input, this step finds the minimum set of tuples to repair the dataset, along with the indication of which patterns each tuple can cover.

Finally, to decide how many copies of each tuple we insert in the dataset, we take the average of the Unfair Tuple Count of the patterns covered by that tuple.

The current implementation of the algorithm does not prevent the generation of unrealistic tuples (e.g. combining "Sex"="Male" with "Pregnant"="Yes"), since there is no constraint on the value combinations. We plan, in a future work, to improve the system with the addition of a *validation oracle* [4] to identify and prevent the insertion of inconsistent tuples.

Example 2. *For each ACFD in Table 2, the system determines the TOS, then, from these new*

Workclass	Race	Sex	Hours-Per-Week	NC	Age-Range	Education	Income
X	X	Female	X	X	X	X	>50K
X	Black	X	X	X	X	X	>50K
X	A-I-E	X	X	X	X	X	>50K
X	X	X	X	NC-Hisp	X	X	>50K

Table 3
Pattern Generation in the U.S. Census Adult Dataset

Tuple	Add
'Private', 'Black', 'Female', '0-20', 'NC-Hispanic', '75-100', 'HS-College', '>50K'	427
'Private', 'Amer-Indian-Eskimo', 'Female', '0-20', 'NC-US', '75-100', 'Assoc', '>50K'	184

Table 4
The tuples with their cardinality mined from the algorithm to repair the U.S. Census Adult Dataset

ACFDs, it generates the corresponding patterns (reported in Table 3). Note that, in the patterns, the target attribute value is changed in order to compensate the Difference of the original ACFDs. The computed set of patterns is the input of the Greedy Hit-Count algorithm. *Table 4 reports the output composed by 2 tuples, with their cardinalities, that should be added to the dataset. The first tuple derives from 6 uncovered patterns and the second tuple derives from 2 uncovered patterns. The two tuples identified from the algorithm have no inconsistencies.*

3.3. Correction algorithm

The aim of the *Greedy Hit-Count algorithm* phase was to try to enhance the fairness of the initial dataset, by adding a very low number of external elements. Using a greedy approach, and setting a limit to the maximum number of tuples that can be added/removed, we prevent the overloading of the final dataset, but this could potentially not be enough to repair the dataset completely. During this phase, if any of the initial ACFDs is still present; we remove from the dataset some tuples matching them. To determine the number of tuples to remove, for each ACFD still present in the dataset we use β, the second value of the Unfair Tuple Count.

Example 3. *After adding tuples in the previous step, only one of the original ACFD is still found. If the user chooses to apply the* Correction algorithm, *the system computes again, according to δ, the number of tuples to remove for each ACFD and generates the corresponding patterns that, converted into tuples are removed from the dataset. In this case, to solve the ACFD we need to remove 1733 tuples, obtaining a final dataset with 29088 tuples.*

3.4. Statistics Computation

We now present a set of metrics to analyze the quality of the repair procedure. Specifically, for each ACFD given as input and still present in the repaired dataset we compute: the *Cumulative Support*, the percentage of tuples in the dataset involved by the selected ACFDs and the *Mean Difference*, the mean of all the 'Difference' scores of the selected ACFDs. Moreover, to make sure that the dataset can still be used for data science tasks, we compare the distribution of values of each attribute in the initial dataset with its counterpart in the repaired one.

Finally, we apply to the repaired dataset the *Data Bias discovery* procedure to inspect the ghost ACFDs, i.e. new ACFDs that were not mined from the original dataset and might appear now after this phase and might introduce new unfair behaviours.

ACFD	Support	Difference
(Sex = 'Female', Class = 1) → Survived= 1	0.103	0.579
(Survived= 0, Sex = 'Female') → Class = 3	0.082	0.332
(Sex = 'Male') → Survived= 0	0.518	0.197

Table 5
A few dependencies from the Titanic Dataset

Example 4. *Using the corrected version of the dataset we perform again the* Data Bias discovery *procedure; we do not have any ACFDs in common with the ones selected from the input dataset, so the dataset can be considered as repaired. The metrics are actually lower: the Cumulative Support, representing the percentage of tuples originally involved by unfair dependencies (around a 35%), after the repair procedure is 0, the Mean Difference goes from 0.13 to 0. We compare the value distribution of each attribute between the initial and the corrected dataset (we do not report the plots for brevity) and the distributions remain almost unchanged, apart from a slight increase in the number of females and private workers. Moreover no unfair ghost ACFD is discovered.*

4. Experimental Results

We now present the results obtained by our system on another real-world dataset: the *Titanic* dataset[2]. The discovery procedure returns the ACFDs that show unfairness, Table 5 reports some of them. The dataset is unfair with respect to all the protected attributes; specifically, the groups more discriminated are: 'Male' and 'Third-Class' passengers. Due to the discrimination we decide to mitigate the bias applying the *ACFD-Repair* procedure. According to the user-selected dependencies, the algorithm computes the number of tuples that should be added to the dataset to compensate their Difference value, specifically, we add 233 tuples to the dataset. Since no initial dependency is extracted from the repaired dataset, the correction procedure is skipped and we consider this version of the dataset as corrected. The obtained metrics for the final dataset show that the mitigation procedure worked: the Cumulative Support, that was around a 86%), after the repair procedure is 0, the Mean Difference goes from 0.35 to 0.

5. Comparison with similar systems

Three possible approaches can be adopted when trying to enforce fairness in a data analysis application: *preprocessing techniques*, i.e. procedures that, before the application of a prediction algorithm, make sure that the learning data are fair; *(ii) inprocessing techniques*, i.e. procedures that ensure that, during the learning phase, the algorithm does not pick up the bias present in the data, and *(iii)* and *postprocessing techniques*, i.e. procedures that correct the algorithm's decisions with the scope of making them fair.

One of the first preprocessing techniques is [5]. The process, on the basis of discrimination measures used in the legal literature, can identify potentially discriminatory itemsets by discovering association rules. Furthermore, the authors propose a set of *sanitization* methods: given a discriminatory rule, it is sanitized by modifying the itemset distribution in order to prevent discrimination. For each discrimination measure, they propose a method to achieve a fair dataset by introducing a reasonable (controlled) pattern distortion. Unfortunately, this

[2] https://www.kaggle.com/c/titanic

system does not involve user interaction, therefore the user cannot discard the rules that are not interesting for the specific investigation. Our approach provides as a final step a set of summarizing metrics that describes the overall degree of unfairness of the dataset.

In the Machine Learning context, the majority of works that tries to enforce fairness is related to a prediction task, and more specifically to classification algorithms. One of these works is *AI Fairness 360*[6], an open-source framework whose aim is to reach algorithmic fairness for classifiers. It tries to mitigate data bias, quantified using different statistical measures, by exploiting pre-processing, in-processing and post-processing techniques. The results obtained by *AI Fairness 360* are in complete accordance with ours. The competitor checks fairness property only for one binary attribute at the time, while, since the ACFD technique can involve more than one attribute at a time, our tool can detect unfair behaviors at finer level of granularity. The main difference between these approaches and our framework, that solves unfairness adopting a *preprocessing technique*, is that our system does not need a classifier to work, because it is based on finding conditions (in form of approximate constraints) that are already present in the data, even though possibly with some level of approximation.

Another related topic regards rule-based data-cleaning techniques [7]. The scope of these approaches is to eliminate the inconsistencies present in dataset by making sure that all the rules (integrity constraints) hold on the final, clean one. In our case, the final objective is quite different: first, we don't modify the values of specific records present in the dataset, but mainly focus on adding tuples to make it more fair; indeed, we are not interested in ensuring that all the tuples respect the discovered dependencies, but, rather, that the final repaired dataset does not contain unfair dependencies.

6. Conclusion and Future Works

We presented a novel framework, that, through the extraction of a particular type of Functional Dependencies, can discover and mitigate bias and discrimination present in datasets.
Future works will include: *(i)* the study of *data equity*, operationalizing this definition and studying how it augments or contradicts existing definitions of fairness, *(ii)* the development of a graphical user interface to facilitate the interaction of the user with the system, *(iii)* the study of other classes of functional dependencies[8].

References

[1] F. Azzalini, et al., FAIR-DB: Functional dependencies to discover data bias, Workshop Proceedings of the EDBT/ICDT (2021).

[2] F. Azzalini, et al., A short account of FAIR-DB: a system to discover data bias, SEBD (2021).

[3] S. Verma, J. Rubin, Fairness definitions explained, in: FairWare, IEEE, 2018.

[4] A. Asudeh, et al., Assessing and remedying coverage for a given dataset, in: ICDE, 2019.

[5] S. Hajian, et al., Discrimination-and privacy-aware patterns, Data Min. Knowl. Discov. 29 (2015) 1733–1782.

[6] R. K. Bellamy, et al., AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias, in: IBM Journal of Research and Development, IBM, 2019.

[7] I. F. Ilyas, X. Chu, Data cleaning, Morgan & Claypool, 2019.

[8] L. Caruccio, et al., Relaxed functional dependencies—a survey of approaches, TKDE (2015).