

Intro to FairDetect

FairDetect framework allows us to detect and understand machine learning bias by defining 3 major fairness metrics: representation, ability, and performance, and then it defines a series of tests to “identify” levels of disparities between sensitive groups. FairDetect is structured into 3 main core steps:

1. Representation: Compares the representation of sensitive variables and its association with the target
2. Ability: Compares the ability of the sensitive variables
3. Predictive: Compares the distribution of the dataset as opposed to the predicted set

Subsequently, FairDetect allows us to deep dive into the model’s predictions, for us to understand how classifications were made and isolate the most unprivileged groups, in a way clear steps can be taken to eliminate and/or mitigate the adverse effects.

FairDetect Improvements Explained

**Improvement 1:** Whenever a rejection of the null hypothesis happens and bias is detected, the statement is printed out in bold letters to show. This helps the user directly to spot bias detection in a visual way.

|  |  |
| --- | --- |
| **FairDetect** | **FairDetect Improved** |
|  |  |

**Improvement 2:** Added Docstrings understanding the functionality of the larger part of the code: the class FairDetect and its methods. Anyone using the library would be able to know all about the FairDetect package like description, package modules, etc., by could simply using the help function to get all the information. This lowers the time of understanding a library and giving easy to access to those users with less knowledge in Python coding syntax.

|  |  |
| --- | --- |
| **FairDetect** | **FairDetect Improved** |
| Graphical user interface, text, application, email  Description automatically generated | A picture containing graphical user interface  Description automatically generated |

**Improvement 3:** Introduced Disparate Impact Ratio in the class: t ratio of positive outcomes (Loan\_Status=1) in the unprivileged group) divided by the ratio of positive outcomes in the privileged group. This is a common metric in bias detection. In further iterations an improvement to be made would be in the method of disparate impact itself. At the moment there is a hard coding that the favoured group is equal to the sensitive group with the value 0 and unfavoured group is equal to the sensitive group with value 1. A more dynamic data entry could be possible in the next version.

Graphical user interface, text, application, email

Description automatically generated

**Improvement 4:** Introduced **user experience** improvements that helps users select critical features for their bias detection and ML exercise which saves them the hassle of encoding the same features (target, sensitive, labels, etc.) for every piece of redundant code. Introduced codes for feature selection which prompts users to select their **Target Value, Sensitive Value and Label Values**

Graphical user interface, text, application

Description automatically generated

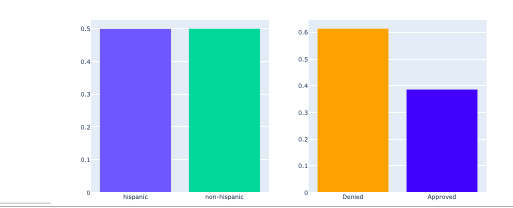
Text

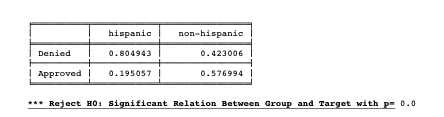
Description automatically generated

Text

Description automatically generated

Why did we introduce these changes? Just imagine users looking at representation charts and seeing their labels encoded with 0s and 1s. It will get confusing to those users less familiar with typical standard statistical terms. The example above just shows that we are now able to have 0s represented by ‘Hispanic’ target labels and 1 represented by ‘Non-Hispanic’ target labels. In addition, the target event is directly labeled instead of using the statistical terms of 0 and 1 in our case “denied” and “approved”





**Improvement 5:** Introduced Quality control method that checks if Target value selected (entered via point number 4 above) was binary and raises out error in case it wasn’t:

Text

Description automatically generated

Improved the **understand\_shap** method which allows us to not only to view feature importance for one class member, in fact it allows us to generate the same visualisations for the other class member (Hispanic and non-hispanic)

Text

Description automatically generated

**Improvement 6 (Bonus):** We accomplished to store the results of our fair detect analysis with SQLite and SQL Alchemy. We selected for our case the KPIs of the p-value of TPR, FPR, TNR and FNR as well as the disparate impact. After each KPIs is calculated is is stored dynamically together with an ID and the date in which it was saved.



**Trade-off:**

The introduction of the improvements unfortunately increased the time until the libraries were loaded. Given the marginal increase of loading vs. the value the introduced improvements provide, we would argue that the trade-off is small.

Loading libraries and functions time before improvement: **2.73 sec**

Loading libraries and functions time after improvements**: 4.84 sec**

Using FairDetect before improvement: **0.5 sec**

Using FairDetect after improvement**: 0.7 sec**

Intro to Aequitas

Aequitas, a second bias framework has been introduced, in order to compare different aspects as functionality, usefulness, methodology regarding bias detection.

Aequitas was created by the Centre for Data Science and Public Policy at the University of Chicago. Users can easily audit models for various bias and fairness indicators in connection to numerous population sub-groups using this open-source tool. Aequitas offers thorough guidance on how to apply its findings in a public policy environment, taking the ensuing interventions and their ramifications into account. Through both seamless integration in the machine learning workflow and a web app designed for non-technical users verifying these models' results, it is meant to be utilised by both data scientists and policymakers.

Tool Comparison

Table below resumes the main differences between FairDetect and Aequitas with respect to main characteristics:Table

Description automatically generated

Tool Comparison and Case Studies

Case Studies 1: Synthetic Credit Card

Graphical user interface, text, application, Word

Description automatically generated

Case Studies 2: Modcloth

To prove the generalisation of the improved FairDetect library also a second dataset besides the synthetic credit card approval was chosen. We decided to leverage the dataset of Modcloth, an American online retailer of indie and vintage-inspired women’s clothing.

Text

Description automatically generated