FairCORELS, an Open-Source Library for Learning Fair Rule Lists
Ulrich Aïvodji, Julien Ferry, Sébastien Gambs, Marie-José Huguet, Mohamed Siala

To cite this version:
Ulrich Aïvodji, Julien Ferry, Sébastien Gambs, Marie-José Huguet, Mohamed Siala. FairCORELS, an Open-Source Library for Learning Fair Rule Lists. ACM International Conference on Information and Knowledge Management, Virtual Event, Nov 2021, Queensland, Australia. pp.4665-4669, 10.1145/3459637.3481965. hal-03427276

HAL Id: hal-03427276
https://laas.hal.science/hal-03427276
Submitted on 13 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
FairCORELS, an Open-Source Library for Learning Fair Rule Lists

Ulrich Aïvodji
UQAM
Montréal, Canada
aivodji.ulrich@uqam.ca

Julien Ferry
LAAS-CNRS, Université de Toulouse, CNRS
Toulouse, France
jferry@laas.fr

Sébastien Gambs
UQAM
Montréal, Canada
gambs.sebastien@uqam.ca

Marie-José Huguet
LAAS-CNRS, Université de Toulouse, CNRS, INSA
Toulouse, France
huguet@laas.fr

Mohamed Siala
LAAS-CNRS, Université de Toulouse, CNRS, INSA
Toulouse, France
msiala@laas.fr

ABSTRACT
FairCORELS is an open-source Python module for building fair rule lists. It is a multi-objective variant of CORELS, a branch-and-bound algorithm to learn certifiably optimal rule lists. FairCORELS supports six statistical fairness metrics, proposes several exploration parameters and leverages on the fairness constraints to prune the search space efficiently. It can easily generate sets of accuracy-fairness trade-offs. The models learnt are interpretable by design and a sparsity parameter can be used to control their length.

CCS CONCEPTS
• Software and its engineering → Software libraries and repositories; • Computing methodologies → Rule learning; Supervised learning; Machine learning algorithms.

KEYWORDS
Machine Learning, Fairness, Interpretability

1 INTRODUCTION
As machine learning techniques are being increasingly used in high stake decision-making systems, fairness and interpretability of such systems have emerged as important ethical issues to address.

FairCORELS, an Open-Source Library for Learning Fair Rule Lists


More precisely, a major concern is to ensure that a model’s predictions are fair, even if it is trained on possibly biased data. Existing fairness-enhancing techniques can be categorized into three main families, mitigating biases at different stages of the machine learning pipeline: pre-processing, post-processing and algorithmic modification techniques [6, 13]. Our approach falls into the last category, as it consists of directly modifying the learning algorithm to enforce fairness.

Another major concern is about facilitating the understanding of a model. While some approaches attempt to explain black-box models or their predictions [14], it has been shown that such explanations may not be trustworthy (e.g., they could be manipulated by the model provider) [1, 19, 20]. For this reason, building models that are interpretable by design (such as rule lists of reasonable size) is an interesting approach for avoiding the a posteriori explanation of black-box models [19]. In particular, rule-based models exhibit interesting properties for interpretability [12] and have been shown to be more expressive than decision trees of equivalent size [18]. While state-of-the-art tools for fair learning exist, such as fairlearn¹, such alternatives are not specifically designed for learning rule lists.

In addition, providing optimality guarantees is important, as there can be societal implications for a lack of optimality [4], especially when addressing fairness.

To address jointly these concerns, we propose FairCORELS [2], an open-source Python library to build rule lists models that are both fair and accurate. FairCORELS leverages on recent advances, provided by CORELS, for learning optimal rule lists [3], by adapting them to also take into account fairness constraints. Given a statistical notion of fairness and a sensitive attribute that could lead to discrimination, our algorithm searches for the rule list optimizing the decrease of both unfairness and misclassification error.

The paper is organized as follows. In the next section, we describe our method for learning fair rule lists. Afterwards in Section 3, we illustrate the use of the proposed Python library for mitigating unfairness in several well-known classification tasks. Finally, we conclude in Section 4.

¹https://fairlearn.org/
2 ALGORITHM FOR LEARNING FAIR RULE LISTS

In this section, we briefly review the fairness notions supported by our library before introducing the optimization problem as well as the FairCORELS algorithm.

Let \( D = (X, Y, A) \) denotes a dataset, in which each example \( e_i = (x_i, y_i, a_i) \in D \) with \( x_i \in [0, 1]^d \) a set of \( n \) unprotected features, \( y_i \in [0, 1] \) a label and \( a_i = (a_{i,p}, a_{i,u}) \in [0, 1]^2 \) with \( a_{i,p} + a_{i,u} \leq 1 \) indicating protected and unprotected group memberships. The goal of a supervised learning algorithm is to learn a classifier whose predictions \( \hat{Y} \) match as accurately as possible the true labels \( Y \).

2.1 Quantifying Fairness

Several fairness notions have been proposed in the machine learning literature, which can be grouped into three types of techniques: individual, causal and statistical fairness [21]. This work is designed to mitigate statistical fairness, also called group fairness, which aims at equalizing some values between distinct protected groups of instances, defined with respect to the value of some sensitive attributes. The core idea is that individuals should be treated similarly, whatever protected group they belong to, because such groups only differ by the value of some features that should not (for ethical, legal or philosophical reasons) be used for decision making.

Depending on the measure being equalized across protected groups, several metrics have been proposed. Our work implements six metrics, namely statistical parity [9], predictive parity [8], predictive equality [8], equal opportunity [15], equalized odds [15] and conditional use accuracy equality [7]. These metrics are summarized in Table 1, along with the statistical measure to be equalized across protected groups.

2.2 Problem Formulation

FairCORELS solves a bi-objective optimization problem minimizing both prediction error and unfairness and it is based on a \( \epsilon \)-constraint approach. Formally, FairCORELS [2] explores the space of rule lists \( \mathcal{R} \) to find the rule list minimizing the objective function \( f_{\text{obj}} \) while having fairness at least \( \epsilon \) (ie exhibiting at most unfairness \( 1 - \epsilon \)).

Function \( \text{misc}(\cdot) \) denotes the misclassification error, and \( \text{unf}(\cdot) \) measures unfairness according to the selected metric. In the provided implementation of FairCORELS, \( \text{unf}(\cdot) \) can be either \( \text{UNF}_{\text{SP}}, \text{UNF}_{\text{PP}}, \text{UNF}_{\text{PE}}, \text{UNF}_{\text{EO}}, \text{UNF}_{\text{Eodd}}, \text{UNF}_{\text{CUAE}} \), as defined in Table 1. In this setting, the rule list \( r^* \) outputted by FairCORELS is the solution of the following problem, in which \( K_r \) is the length of rule list \( r \) and \( \lambda \) is a regularizer parameter controlling the accuracy/sparsity trade-off:

\[
\arg \min_{r \in \mathcal{R}} f_{\text{obj}} = \text{misc}(r, X, Y) + \lambda K_r \\
\text{s.t.} \quad \text{unf}(r, X, Y, A) \leq 1 - \epsilon
\]

2.3 Algorithm Overview

Prior to the use of FairCORELS is the pre-computation of rules, which are given as inputs to FairCORELS. Rules can be any combination of attributes (or even attributes themselves) that evaluate to either 0 or 1 for all instances.

Based on the CORELS algorithm [3], FairCORELS represents the space of rule lists \( \mathcal{R} \) using a prefix tree, in which each node is a rule and each path from the root is a potential solution. Starting from a root node, the exploration frontier is contained in a priority queue \( Q \), which can be ordered following several strategies: depth-first search, breadth-first search or different best-first searches. We also implement several custom breadth-first search heuristics, among which the BFS obj.-aware orders rule lists of equal lengths depending on their objective function values. At each iteration, the prefix \( r \) at the top of \( Q \) is selected. For each possible extension \( r' \) of it, \( f_{\text{obj}} \) is computed. Whenever its value is improved and the solution meets the fairness constraint, the current best solution is updated. All extensions \( r' \) that do not violate any of the bounds are pushed into \( Q \).

FairCORELS exploits several bounds to efficiently prune \( \mathcal{R} \), leveraging both the objective function and the fairness constraint. It returns theoretically certifiable optimal rule lists (i.e., given a set of training instances and rules, it provably returns the rule list with smaller \( f_{\text{obj}} \) among those meeting the fairness constraints). However, as the prefix tree size grows exponentially with the number of rules, a parameter \( n_{\text{iter}} \) can be used to stop the exploration whenever the number of nodes evaluated in the prefix tree reaches \( n_{\text{iter}} \). This parameter is typically used to limit the memory footprint of the program or to quickly obtain some solution.

3 FAIRCORELS PYTHON LIBRARY: EXAMPLE USE

The source code and documentation of FairCORELS is available online\(^2\), and FairCORELS can also be easily installed using Pypi\(^3\), with \texttt{pip install faircorels}. A complete description of the algorithm’s parameters is provided, along with example code and binarized data for two well-known biased datasets. FairCORELS is based on the Python binding of CORELS\(^4\). The core of the algorithm is implemented in C++ and called from the Python wrapper using Cython. Along with the classifier object, FairCORELS also contains some tools for auditing a model’s fairness. A wrapper object for fair ensemble learning using the Bootstrap Aggregating (Bagging) method [23] is also provided.

In this paper we propose an illustration of the use of FairCORELS, for the Statistical Parity metric, on four well-known biased classification tasks:

- The \textit{Adult} dataset [11] in which the objective is to predict whether someone earns more than 50,000$ per year. We consider gender as the sensitive attribute.
- The \textit{COMPAS} dataset [5] in which the objective is to predict whether criminal offenders will recidivate within the next two years. We consider race as the sensitive attribute.
- The \textit{Bank Marketing} dataset [16] in which the objective is to predict whether a customer will subscribe to a term deposit. We use age as the sensitive attribute.
- The \textit{Default of Credit Card} dataset [22], in which the objective is to predict whether a person will default in payment. We consider gender as the sensitive attribute.

\(^2\)https://github.com/ferryjul/fairCORELS  
\(^3\)https://pypi.org/project/faircorels/  
\(^4\)https://github.com/corels/pycorels
The procedure is identical for the four datasets. The first step towards the use of FairCORELS is the binarization of the dataset. Then, given the set of binarized features, a rule mining procedure searches for rules formed by single- and two-clause antecedents, only retaining rules that capture at least 0.05% of the training instances. In our experiments, the sensitive attribute is not used to prevent disparate treatment. We illustrate the use of FairCORELS to enforce the Statistical Parity metric, but similar results can be obtained for any of the six supported group fairness metrics. For all experiments, the maximum size of the prefix tree is fixed to 25.10³, the BFS obj. -aware heuristic is used to order the queue, and the regularization parameter $\lambda = 10^{-3}$. The corresponding model creation is shown in Code snippet 1.

```python
# Create the classifier object
clf = FairCorelsClassifier(
    n_estimators=200, # Max. size of the prefix tree
    n_jobs=1, # Regularization parameter Lambda
    max_card=2, # Rule mining: max. rules cardinality
    min_support = 0.05, # Rule mining: min. support
    policy='bfs', # Exploration heuristic
    bfs_mode=1, # BFS obj. -aware
    fairness='SP', # fairness metric
    epsilon=epsilon, # The fairness constraint
    min_values=[0,1], # Binary vector (protected group membership)
    use_fairness=False # Use the advanced filtering
)

# Train it
clf.fit(X_train, y_train, features=featuresNames, prediction_name="high_income")
```

Code snippet 1: Example to train a rule list with statistical parity constraint.

Note that rule mining can either be performed as pre-processing or using the built-in options. We use the built-in options in Code snippet 1, mining rules that are conjunctions of 2 attributes (or their negations) capturing at least 3% of the training instances. For our experiments, we performed rule mining as pre-processing to control the number of generated rules before training a model. Note that min_values and max_values are used to define the protected (respectively unprotected) groups and measure unfairness while training the model. They can correspond to a particular feature, but can also be any binary vectors, as long as they do not intersect. Our Python package was designed to meet scikit-learn³ naming conventions and can easily be integrated in any piece of code originally written for scikit-learn models.

Our experiments were conducted on an Intel Xeon Processor E3-1271 v3 (3.60 GHz) with 32GB of RAM. Presented values (accuracy, fairness) are averaged using 5-folds cross-validation. The reported model’s statistics are measured on their respective train/test split. Recall that unfairness $= 1 - fairness$.

Generating Single Rule Lists. Rule list 2 has been learned maximizing accuracy only (hence setting the fairness constraint $= 0$). It has high accuracy but exhibits considerable unfairness. As we prevent the use of the sensitive attribute in the model’s decisions, the unfair nature of this interpretable model may not be obvious. Indeed, some of the non-sensitive attributes (or combinations of them) may be correlated with the sensitive one. In this case, statistical fairness measures allow for measuring disparate impact, which happens when a model’s outcomes differ significantly depending on individuals’ membership to some protected group, even without explicit knowledge of such membership.

```python
if (capitalgain>00000) then [high]
else if ([1811.5<capitalgain<=1978.5]) then [high]
else if ([education=twelve AND capitalgain<1754.8]) then [low]
else if ([occupation=whiteCollar AND hoursPerWeek>40.5]) then [high]
else [low]
```

Rule list 2: Example of a rule list found on the Adult Income dataset, for the Statistical Parity metric, maximizing accuracy only ($\epsilon = 0$). Test accuracy on the associated train/test split is 0.817, and unfairness is 0.073. For these running parameters, test accuracy (averaged across the 5 folds) is 0.817, and unfairness is 0.066.

Rule list 3 corresponds to $\epsilon = 1.0$. We want to point out that in this case the only rule list able to achieve perfect fairness is a constant (trivial) classifier. As the used dataset is class-unbalanced, constantly predicting the majority class leads to an acceptable accuracy (better than random guess), without exhibiting unfairness at all.

```python
if [low]
```

Rule list 3: Example of a rule list found on the Adult Income dataset, for the Statistical Parity metric, with fairness constraint $\epsilon = 1.00$. Test accuracy (averaged across the 5 folds) is 0.752, and unfairness is 0.00.

Between these two extremes, we can generate many trade-offs by applying different constraints $\epsilon$. By doing so, we are able to significantly reduce unfairness without reaching trivial accuracy.

---

³https://scikit-learn.org/
For example, Rule list 4 has very low unfairness while maintaining good accuracy. A (possible) measure of the trade-off’s quality has been proposed in [17]: $\delta = \text{accuracy} - \text{unfairness}$. Among our generated solutions, Rule List 4 maximizes this criterion.

**Rule list 4: Example of a rule list found on the Adult Income dataset, for the Statistical Parity metric, with fairness constraint $\epsilon = 0.984$. Test accuracy on the associated train/test split is 0.792, and unfairness is 0.0036. For these running parameters, test accuracy (averaged across the 5 folds) is 0.793 and unfairness is 0.014.**

**Generating a Set of Accuracy/Fairness Trade-offs.** FairCORELS can also be applied to generate a set of accuracy/fairness trade-offs instead of a single solution. This can be done easily by performing successive (possibly parallel) calls to FairCORELS, varying the fairness constraint parameter $\epsilon$. Indeed, this allows domain experts to select the most appropriate solution, depending on the accuracy/fairness trade-off, but also possibly on the associated models themselves. The interpretable nature of the built models makes their analysis easier. Figure 1 shows a set of non-dominated solutions (in terms of error and unfairness, on their test sets) for the four datasets, in which Rule Lists 2 ("max. accuracy"), 3 ("max. fairness") and 4 ("best delta") are represented on the Adult Income dataset plot. We see that the set of possible trade-offs is large enough for the decision maker to select the appropriate one.

## 4 DISCUSSION

With FairCORELS, we propose an in-processing method for learning interpretable and fair models. One of the strengths of our approach is that it includes most of the standard statistical fairness metrics. In addition, our core method is metric-agnostic and thus integrating new fairness metrics should be straightforward.

Our formulation of the fair learning problem is particularly convenient for real-life fair learning problems, where particular unfairness values may correspond to legal requirements (e.g., the 80% rule for Statistical Parity [10]).

As shown in [2], FairCORELS is able to achieve interesting trade-offs between accuracy and fairness, that are competitive with state-of-the-art interpretable and non-interpretable methods.

Additionally, FairCORELS is an exact method, that can be used to certify the optimality of the generated models. As demonstrated throughout this paper (and the related demonstration), FairCORELS scales fairly well and can be used to learn accurate and fair models using datasets of various sizes.
REFERENCES


