ENGINEERING FAIR MACHINE LEARNING PIPELINES

Martin Hirzel, Kiran Kate, and Parikshit Ram
IBM Research, USA

ABSTRACT

Data splits and data preparation during fairness mitigation are known to influence the performance of output models. We propose using protected attributes in stratification when splitting a dataset. We also describe fairness patterns for assembling fair pipelines that include data preparation, estimators, and mitigators. This paper introduces an open-source Python library `lale.lib.aif360` that offers sklearn compatible implementations of fair stratification and fairness patterns.

1 INTRODUCTION

Decisions in finance, hiring, promotions, and even criminal justice are increasingly influenced by machine learning. Ethics, laws, and public opinion demand that these decisions should not discriminate against minorities. Fortunately, for a well-defined fairness dataset and task, the literature on algorithmic fairness offers various fairness metrics and mitigators. Unfortunately, algorithms alone are not enough: fairness also requires sound human judgment and solid engineering. This paper offers improvements on the engineering front. While the broader human issues around fairness are beyond the scope of this paper, we hope our techniques help when applied judiciously.

The problem. Recent work has highlighted the engineering gap for fair machine learning. Friedler et al. (2019) empirically observe that '*algorithms tend to be sensitive to fluctuations in dataset composition* [...] *and to different forms of preprocessing*'. Lee & Singh (2021) interview and survey practitioners, who lament '*limited ability to adapt and integrate the* [*available fairness*] *tools to "plug and play" with the existing workflow*'. We illustrate these gaps with two concrete examples. The datasets used in these examples are the Ricci dataset (ric, 2009), which captures data about firefighter promotion exams, and the CreditG dataset (Lehr & Ohm, 2017), which concerns credit card applications. For illustration, we make particular choices of fairness metric, classifiers, and data preparation in these examples. However, the gaps have been observed for other choices as well.

Our first example illustrates how fluctuations in dataset composition can cause misleading conclusions during testing. This example uses the *disparate impact* metric (Feldman et al., 2015), which has an ideal value of 1, with values below 0.8 considered unfair. We used a random split of the Ricci dataset and fit a logistic regression classifier to the training set. While the training set had a ground-truth disparate impact of 0.39, the classifier predictions on the test set had a disparate impact of 0.75, so one might erroneously conclude that the classifier increased fairness. However, with a different random split of the same dataset, the training set had a ground-truth disparate impact of 0.68 and the classifier predictions on the test set had a disparate impact of 0.11, so one might erroneously conclude that the classifier reduced fairness. When we repeat the experiment 50 times, it turns out that on average, the disparate impact is 0.51 in the ground truth and 0.49 in the predictions, indicating that the bias in the model approximates the bias in the data. This example highlights that fluctuations in the train/test splits can lead to wildly different conclusions and the truth emerges when we repeat the experiment multiple times with different splits. However, a large number of repetitions is not always practical and we need tools that mitigate fluctuations in the train/test splits.

encoder	accuracy	disparate impact
one-hot	$75.4\% \pm 2.0\%$	0.808 ± 0.129
ordinal	$72.4\% \pm 1.9\%$	0.974 ± 0.085
target	$73.3\% \pm 2.2\%$	1.028 ± 0.086

Our second example illustrates how data preparation can interfere with fairness mitigation. This example uses the CreditG dataset, a logistic regression classifier, and a disparate impact mitigator by Feldman et al. (2015). While the data contains categorical features, the mitigator requires numeric features, so our pipeline also encodes categorical features as numbers. We report metrics as average $\pm$ standard deviation of 50 trials with fair stratified (see Section 2) splits. While one-hot encoding yields the best accuracy in these experiments, it makes the mitigator less effective at improving disparate impact. The ordinal encoder improves disparate impact at the expense of accuracy. The target encoder (Micci-Barreca, 2001) recovers some of that accuracy. More generally, we need learning pipelines to integrate data preparation with fairness mitigation and predictions. We must be careful to avoid data leakage from the test set to the train set. For example, the target encoder must only look at the target column from the split of the data used for fitting the pipeline. This example demonstrates that data preparation can significantly affect the success of bias mitigators, and we need tools that allow us to easily and correctly integrate data preparation into the bias mitigation pipelines without introducing data leakage or additional bias. They should also allow easy plug-and-play of mitigation with the training and testing workflow in a consistent and correct way. Our `lale.lib.aif360` library attempts to address these two gaps and is fully compatible with sklearn to be pluggable to existing workflows.

Related work. Multiple recent open-source toolkits offer fairness mitigators. Unfortunately, none of them is sufficiently compatible with sklearn (Buitinck et al., 2013) to address the above-described problems. Themis-ML (Bantilan, 2017) mitigators offer a method `fit(X, y, s)`, which differs from sklearn's `fit(X, y)` by an extra `s` argument for sensitive attributes. AIF360 (Bellamy et al.) provides the most comprehensive collection of fairness metrics and mitigators (Lee & Singh, 2021), but requires data to be either using its own dataset class or to represent protected attributes as a pandas index (McKinney, 2011), neither of which are propagated between operators in a sklearn pipeline. Fairlearn mitigators offer a method `fit(X, y, sensitive_features)`, which again differs from the sklearn API. As Lee & Singh (2021) point out, this makes it hard to adapt and integrate these toolkits into existing workflows. The fair-DAGs library focuses on measuring fairness for the data preparation part of the pipeline (Yang et al., 2020), but offers no mitigation. Valentim et al. (2019) use protected attributes for undersampling but not for stratified splits. The current state of the art lacks support for robust testing of fairness intervention and for preparing data as part of the same pipeline, and thus leaves an engineering gap to the existing ecosystem of sklearn and pandas.

Contributions. We propose novel solutions to the problems outlined above and introduce a new Python library (`github.com/IBM/lale/tree/master/lale/lib/aif360`) that provides (i) fair stratification, (ii) fairness patterns, (iii) sklearn integration of fairness metrics and mitigation.

2 FAIR STRATIFICATION

Splitting data into a training and a test set helps detect and reduce over-fitting. Stratifying by target labels helps represent smaller classes better in each split, thus increasing stability. For fairness, besides class imbalance, there may also be imbalance between unprivileged and privileged groups. In fact, there may be multiple groups (e.g., by gender and age), and ideally, each split should represent all intersections well. Unfortunately, smaller intersections cause observed results to fluctuate more.

While fairness algorithms and toolkits usually assume binary protected attributes and target labels, this is rarely the case for real-world datasets, or even for datasets from repositories such as OpenML (Vanschoren et al., 2014). Furthermore, any data splitting should happen on the raw data before data preparation, because otherwise, data preparation may leak information from the test set (such as in target encoding). Thus, fair stratification should perform the necessary binary encoding. Let a group specification G be given by a set of values (for categorical variables) or ranges (for continuous variables). Then, we can encode an attribute a to a Boolean using function $e_G(a)$ defined as $e_G(a) = \exists g \in G : a = g \vee g_{\text{min}} \leq a \leq g_{\text{max}}$. We can generalize e_G to tuples of a target label y and protected attributes a_i as $e_G(\langle y, a_1, \ldots, a_n\rangle) = \langle e_{G_y}(y), e_{G_1}(a_1), \ldots, e_{G_n}(a_n)\rangle$. Since each of the encodings is binary, this yields 2^{n+1} unique encoded tuple values. Then, we can simply stratify by these tuples. Our library `lale.lib.aif360` supports the group specifications as a JSON value:

```
creditg_fairness_info = {
  "favorable_labels": ["good"],
  "protected_attributes": [
    { "feature": "personal_status",
      "reference_group": ["male div/sep", "male mar/wid", "male single"]},
    { "feature": "age",
      "reference_group": [[26, 1000]]}]}
```

Here, the target label should be encoded via `a=="good"`; attribute `personal_status` should be encoded via `a in ["male div/sep", "male mar/wid", "male single"]`; and `age` should be encoded via `26<=a<=1000`. Our `lale.lib.aif360` library provides an API for fair stratified split that understands this fairness information (in Python, `**` unpacks a dictionary into keyword arguments):

```
train_X, test_X, train_y, test_y = fair_stratified_train_test_split(
    all_X, all_y, **creditg_fairness_info, test_size=0.33, random_state=None)
```

In addition to an outer train-test split, an inner cross validation reduces over-fitting with sklearn's `GridSearchCV` or with automated machine learning tools such as Hyperopt (Bergstra et al., 2015). We provide `FairStratifiedKFold`, which is compatible with cross-validation in sklearn:

```
fair_cv = FairStratifiedKFold(**fairness_info, n_splits=5)
grid = {"C": [0.01, 0.1, 1], "solver": ["liblinear", "saga"]}
hpo = GridSearchCV(estimator=LogisticRegression(), cv=fair_cv, param_grid=grid)
hpo.fit(train_X, train_y)
```

Results. To quantify the advantage of fair stratification, we consider the stability of the fairness metric computed with k-fold cross validation (CV). For CreditG (with protected attributes as presented above) and logistic regression (default hyperparameters), we perform multiple 10-fold CVs and consider the average standard deviation (over the multiple CVs) as the measure of stability. For disparate impact, class stratification improves stability over random stratification by 1% while fair stratification improves it by 19%. The stratification strategy did not affect the mean of the fairness metric. In an additional experiment, we down-sample rows belonging to the unprivileged group with a favorable label by 80%. This worsens the inherent disparate impact by further under-representing the unprivileged group. In this case, class-stratification improves stability over random by 5.5% while fair-stratification improves it by 26%. These results indicate that fair stratification can reduce fluctuations among dataset splits especially if a group is under-represented.

3 Fairness Patterns

Most fairness mitigators require clean numeric data with binary protected attributes and labels (Friedler et al., 2019). However, datasets commonly have missing values and multi-valued categorical attributes and labels (Vanschoren et al., 2014). Therefore, most machine-learning pipelines include data transformers (such as imputers or encoders) followed by an estimator (such as a classifier or regressor). In the context of fairness, we should evaluate the data transformation/preparation, estimator, and mitigator jointly to avoid the problems described in the introduction. We argue that this is best done by assembling them into a single joint pipeline. The benefits of such a pipeline extend beyond the development phase of machine-learning engineering. Such a pipeline can also be used to communicate to stakeholders exactly how its components work together, and to guarantee that the pipeline deployed in production is faithful to what was used for training and testing.

Current toolkits do not support this (Lee & Singh, 2021), leading to the question: *why is it difficult?* We believe it is difficult due to a fundamental dilemma: the pipeline must place data preparation before the mitigator, yet it must hide protected attributes from the data preparation and provide them to the mitigator. Revealing sensitive attributes to the data preparation sub-pipeline can cause problems both in principle (disparate treatment of unprivileged groups by transformers) and in practice (transformation-induced data leakage from protected attributes to proxy attributes). Properly assembling all the pieces (preparation, estimator, and mitigator) into an effective whole (a fair pipeline) is non-trivial, and fairness toolkits should not leave that burden to their users. For instance, when configuring the mitigator, users should not have to mentally simulate the effects of encodings on fairness information; instead, the toolkit should take care of that automatically.

This paper proposes *fairness patterns* as a solution. Figure 1 illustrates our patterns, which vary by mitigator kind: mitigation can happen pre-, in-, or post-estimator. *Redacting* hides protected attributes from the data preparation sub-pipeline. For each protected attribute a, redacting picks one of the values $v \in a$ and sets the entire column to be the constant v. Instead of redaction, another possibility could have been projection, i.e., dropping the column. But we have found that by shifting column indices, projection can invalidate pre-trained transformers; redacting avoids that problem. A separate sub-pipeline of our patterns encodes protected attributes using the e_G transformation from Section 2. Then, the patterns re-join the two transformed intermediate datasets to obtain the input for the mitigator. Finally, the in-estimator pattern and the post-estimator pattern decode labels.

Our `lale.lib.aif360` library implements fairness patterns as sklearn compatible *meta-estimators*. A meta-estimator is a machine learning operator that takes other operators (here, the estimator) or

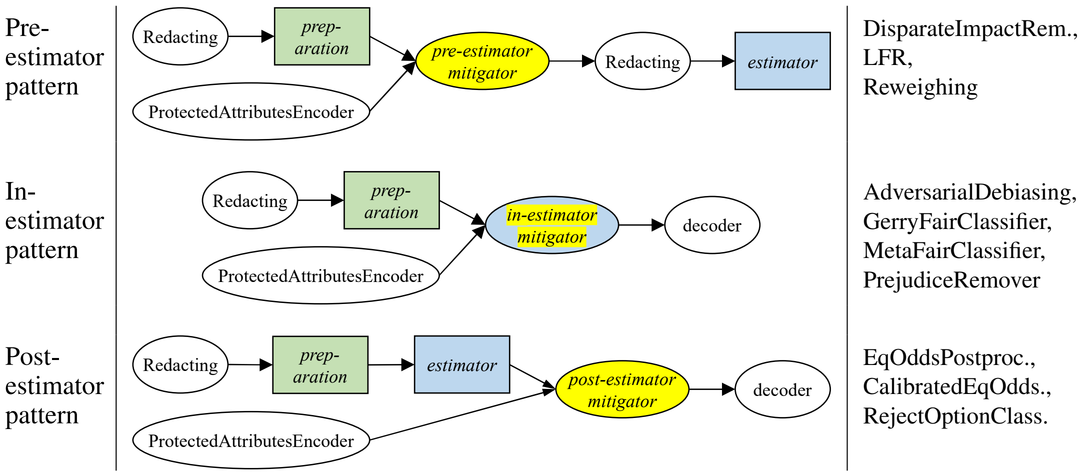
Pre-estimator pattern
Redacting
prep-aration
ProtectedAttributesEncoder
pre-estimator mitigator
Redacting
estimator
DisparateImpactRem., LFR, Reweighing
In-estimator pattern
Redacting
prep-aration
ProtectedAttributesEncoder
in-estimator mitigator
decoder
AdversarialDebiasing, GerryFairClassifier, MetaFairClassifier, PrejudiceRemover
Post-estimator pattern
Redacting
prep-aration
estimator
ProtectedAttributesEncoder
post-estimator mitigator
decoder
EqOddsPostproc., CalibratedEqOdds., RejectOptionClass.

Figure 1: Patterns for fairness mitigation with data preparation. Right: wrapped AIF360 mitigators.

dataset	dataset size	with redaction		without redaction	
	rows × cols	accuracy	disparate impact	accuracy	disparate impact
ricci	118 × 5	83.6% ± 3.8%	1.008 ± 0.229	83.6% ± 3.8%	1.008 ± 0.229
creditg	1,000 × 20	71.1% ± 1.5%	0.988 ± 0.098	72.5% ± 1.5%	0.793 ± 0.113
compas	5,278 × 13	66.6% ± 1.0%	0.985 ± 0.129	66.6% ± 1.0%	0.991 ± 0.126
bank	45,211 × 16	89.0% ± 0.2%	0.744 ± 0.030	89.0% ± 0.1%	0.810 ± 0.021
adult	48,842 × 14	79.7% ± 0.2%	0.425 ± 0.058	79.7% ± 0.2%	0.425 ± 0.060

Table 1: Results of mitigation with and without redaction. Results reported as $mean \pm std$.

sub-pipelines (here, data preparation) as an argument. Currently, `lale.lib.aif360` provides 10 meta-estimators, one for each of 10 mitigators from AIF360 listed in the right column of Figure 1. Each of them supports the same fairness information arguments illustrated in Section 2. The user can specify the fairness information in terms of the input dataset, and the meta-estimator internally adapts it to apply to the intermediate dataset after transformers and encoding. Here is an example:

```
1 prep = make_union(
2    make_pipeline(Project(columns={"type": "string"}), OrdinalEncoder()),
3    Project(columns={"type": "number"}))
4 pipeline = make_pipeline(
5    DisparateImpactRemover(preparation=prep, **creditg_fairness_info),
6    LogisticRegression())
7 trained = pipeline.fit(train_X, train_y)
```

Lines 1–3 use sklearn's `make_union` to specify a data preparation pipeline that ordinal-encodes categorical columns while forwarding numerical columns unchanged. Lines 4–6 use sklearn's `make_pipeline` to create a two-step pipeline. The first step, on Line 5, configures a `lale.lib.aif360.DisparateImpactRemover` meta-estimator with the preparation pipeline and fairness information. The second step, on Line 6, is a classifier. Line 7 trains the pipeline.

Results. To showcase the impact of redaction during mitigation, we run the above pipeline with and without the Redacting step from Figure 1 on 5 datasets. Table 1 summarizes accuracy and disparate impact for ten 75%-25% fair stratified train-test splits for each dataset. Out of the 5 datasets, redaction increased disparate impact for one and decreased it for another, while leaving the rest unaffected. Therefore, `lale.lib.aif360` provides an option to toggle redaction on or off, so users can choose what works best for their setting.

4 Conclusion

This paper described two novel techniques: (i) fair stratification and (ii) fairness patterns. The `lale.lib.aif360` library implements both of these techniques in a sklearn-compatible manner. Fair machine learning needs work on many fronts; we hope this paper offers concrete steps towards better incorporating fairness into the end-to-end machine learning engineering workflow.

REFERENCES

Ricci v. DeStefano, 2009. Supreme Court of the United States, 557 U.S. 557, 174. 2658 pages.

Niels Bantilan. Themis-ML: A fairness-aware machine learn-ing interface for end-to-end discrimination discovery and mitigation, 2017. URL https://arxiv.org/abs/1710.06921.

Rachel K. E. Bellamy, Kuntal Dey, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, Seema Nagar, Karthikeyan Natesan Ramamurthy, John Richards, Diptikalyan Saha, Prasanna Sattigeri, Moninder Singh, Kush R. Varshney, and Yunfeng Zhang. AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. URL https://arxiv.org/abs/1810.01943.

James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D. Cox. Hyperopt: a Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1), 2015. URL http://dx.doi.org/10.1088/1749-4699/8/1/014008.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: Experiences from the scikit-learn project, 2013. URL https://arxiv.org/abs/1309.0238.

Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 259–268, 2015. URL https://doi.org/10.1145/2783258.2783311.

Sorelle A. Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P. Hamilton, and Derek Roth. A comparative study of fairness-enhancing interventions in machine learning. In *Conference on Fairness, Accountability, and Transparency (FAT*)*, pp. 329–338, 2019. URL https://dl.acm.org/doi/10.1145/3287560.3287589.

Michelle Seng Ah Lee and Jatinder Singh. The landscape and gaps in open source fairness toolkits. In *Conference on Human Factors in Computing Systems (CHI)*, 2021. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3695002.

David Lehr and Paul Ohm. Playing with the data: What legal scholars should learn about machine learning. *UC Davis Law Review*, 51:653–718, 2017.

Wes McKinney. pandas: a foundational Python library for data analysis and statistics. *Workshop on Python for High Performance and Scientific Computing (PyHPC)*, pp. 1–9, 2011. URL https://www.dlr.de/sc/Portaldata/15/Resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf.

Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter*, 3(1):27–32, 2001. URL https://dl.acm.org/doi/10.1145/507533.507538.

Inês Valentim, Nuno Lourenço, and Nuno Antunes. The impact of data preparation on the fairness of software systems. In *International Symposium on Software Reliability Engineering (ISSRE)*, pp. 391–401, 2019. URL https://doi.org/10.1109/ISSRE.2019.00046.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations Newsletter*, 15(2):49–60, June 2014. URL http://doi.acm.org/10.1145/2641190.2641198.

Ke Yang, Biao Huang, Julia Stoyanovich, and Sebastian Schelter. Fairness-aware instrumentation of preprocessing pipelines for machine learning. In *Workshop on Human-In-the-Loop Data Analytics (HILDA)*, 2020. URL https://hilda.io/2020/proceedings/HILDA2020_paper9.pdf.