Oversampling to Repair Bias and Imbalance Together

Anonymous[1]

[1]Anonymous Institution

Abstract Both group bias and class imbalance occur when instances with certain characteristics are under-represented in the data. Group bias causes estimators to be unfair and class imbalance causes estimators to be inaccurate. Oversampling ought to address both kinds of under-representation. Unfortunately, it is hard to pick a level of oversampling that yields the best fairness and accuracy for a given estimator. This paper introduces Orbit, an oversampling algorithm that can be precisely tuned for both fairness and accuracy. Orbit is a pre-estimator bias mitigator that modifies the data used to train downstream estimators. This paper demonstrates how to use a Bayesian optimizer to tune Orbit along with the choice of estimator that follows it and empirically compares various approaches for blending multiple metrics into a single optimizer objective. Overall, this paper introduces a new bias mitigator along with a methodology for applying automated machine learning for fairness.

1 Introduction

Machine learning often suffers from the twin problems of group bias and class imbalance. In a classification setting, *class imbalance* occurs when the number of instances with one class label is smaller than with another class label. Class imbalance has long been recognized as a problem, because many models perform poorly for minority classes, and in many applications, the cost of misprediction is unequal across classes [6]. One definition for *group bias* is that instances in one group experience a smaller ratio of favorable outcomes than another group [9]. Here, a *group* comprises all instances for which a protected attribute such as race or gender has a certain value, or a protected attribute such as age falls on one side of a certain threshold. And an *outcome* is the prediction target of the instance, in this paper, a class label. Group bias is increasingly recognized as a problem because it can cause ethical, legal, reputational, and financial issues. Typically, a group experiencing bias is also a minority group, i.e., it is under-represented in the training data.

Both problems, group bias and class imbalance, involve subsets of instances being under-represented. Having fewer samples makes it harder for models to generalize. Sub-dividing data by intersecting groups and classes further exacerbates this limited-data problem. Fortunately, there are algorithms for mitigating group bias (e.g. reject option classification [11]) and class imbalance (e.g. Smote [6]). However, this paper shows that mitigating either goal separately can harm the other goal; for example, when Smote reduces the class imbalance of a dataset, that can exacerbate its group bias. Furthermore, it is not clear how much to repair the data to achieve the desired effect in estimators trained from that data. Since the effect of different mitigation levels is unpredictable, we argue it should be tuned automatically, necessitating repair level hyperparameters.

This paper introduces the Orbit algorithm, which extends Smote [6] to repair for both objectives such that the repair level can be precisely controlled via hyperparameters. In experiments across 12 datasets, Orbit performs well compared to 3 other imbalance mitigators and 9 other bias mitigators from prior work. Since Orbit is designed with automated machine learning (AutoML) in mind, this paper also elaborates on an estimator evaluation workflow for that setting. We define 5 approaches for blending metrics for accuracy and fairness into a single objective and empirically compare what effect each approach has on AutoML performance. The code for Orbit, along with improvements we made to other mitigators used in this paper, are available open-source (URL elided for double-blind review).

 © 2023 the authors, released under CC BY 4.0

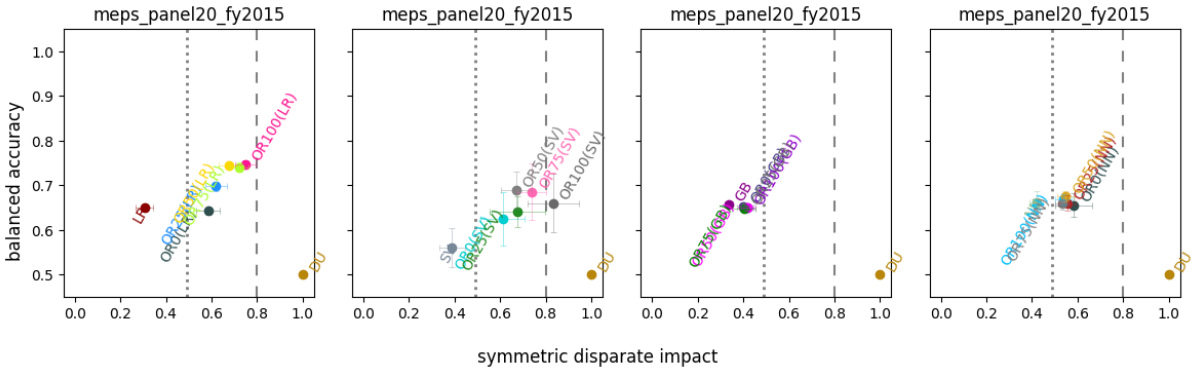


Figure 1: The effect of repair levels and estimators. ORδ_{bias} denotes Orbit with imbalance repair level 80% and bias repair level δ_{bias}. Estimators are LR (logistic regression), SV (support vector machine), GB (gradient boosting), NN (neural network), and DU (dummy estimator).

2 Problem Statement

Figure 1 shows results for Orbit with different repair levels and estimators on the meps20 dataset [1]. The x-axis shows a variant of disparate impact [9], made symmetric so that higher values are more fair, and the y-axis shows balanced accuracy, where higher values are more accurate. Each point is an average of six runs (two repeats of 3-fold cross validation) and the error bars show one standard deviation. The dashed lines at symmetric disparate impact 0.8 indicate the 80% rule [9] and the dotted lines show the base disparate impact of the dataset. The dummy classifier is always at the bottom right, with the best symmetric disparate impact of 1 and the worst balanced accuracy of 0.5.

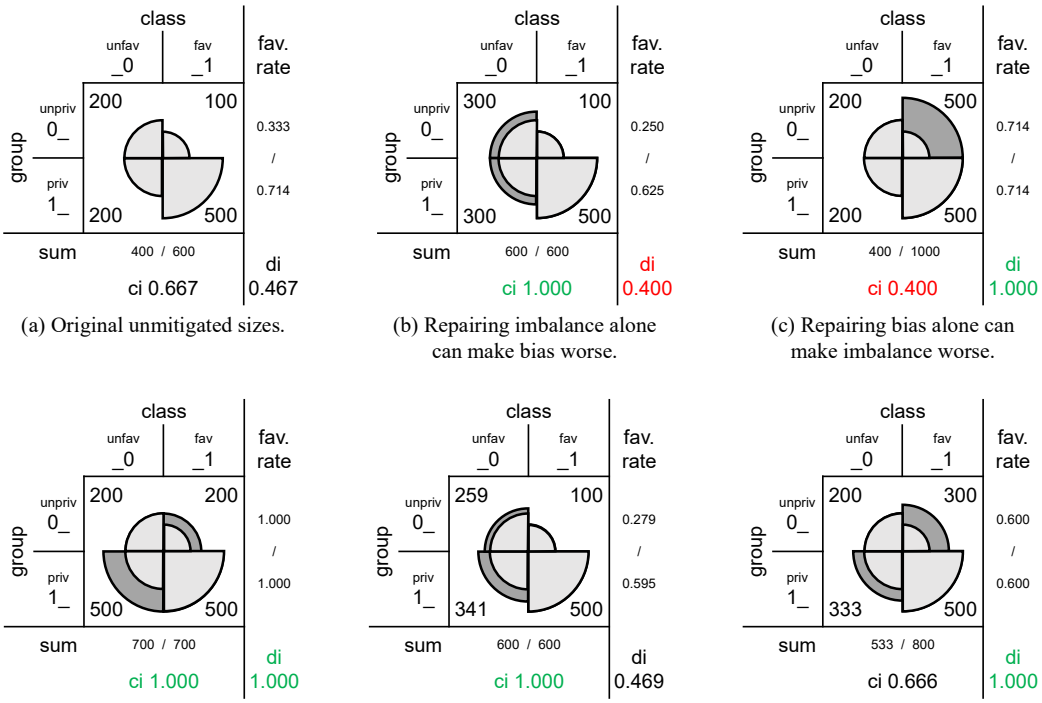
The leftmost plot in Figure 1 shows that for LR (logistic regression), the highest repair level (OR100) yields both the best accuracy and best fairness. Moving to the next plot, for SV (support vector machine), repair levels up to 50% improve both metrics, but above that, higher bias repair causes better fairness at the expense of worse accuracy. For GB (gradient boosting), repair hardly affects either metric. Finally, for NN (neural network), repair has a slightly larger effect than for GB, but the errors bars are too large to draw conclusions. Overall, *the effect of imbalance and bias repair on estimators is unpredictable.* This motivates tunable repair level hyperparameters.

Consider a binary protected attribute whose value can be either unprivileged (0) or privileged (1) and a binary target label whose value can be either unfavorable (0) or favorable (1). This breaks up a dataset into four intersections of sizes o_{00} (unprivileged unfavorable), o_{01} (unprivileged favorable), o_{10} (privileged unfavorable), and o_{11} (privileged favorable). Define the original (before repair) class imbalance o_{ci} and disparate impact o_{di} as

$$o_{\text{ci}} = \frac{o_{00} + o_{10}}{o_{01} + o_{11}} \qquad \text{and} \qquad o_{\text{di}} = \frac{o_{01}/(o_{01} + o_{00})}{o_{11}/(o_{11} + o_{10})}, \tag{1}$$

where the numerator of o_{di} is the favorable rate for the unprivileged group and the denominator is the favorable rate for the privileged group [9].

Figure 2 illustrates different choices for oversampling the intersections of the data to new sizes n_{00}, n_{01}, n_{10}, and n_{11}. The starting point in Figure 2(a) is a dataset with $o_{00} = 200$, $o_{01} = 100$, $o_{10} = 200$, and $o_{11} = 500$. This dataset has a class imbalance of $o_{\text{ci}} = 0.667$ and a group bias of $o_{\text{di}} = 0.467$. Figure 2(b) shows the effect of oversampling to repair class imbalance while being oblivious to group bias, i.e., the effect of using an algorithm such as Smote [6] out of the box. Unfortunately, this makes bias worse, reducing it from $o_{\text{di}} = 0.467$ to $n_{\text{di}} = 0.400$ Similarly, Figure 2(c) shows that a naive bias repair algorithm that only oversamples the unprivileged favorable intersection would make imbalance worse. In contrast, Figure 2(d) shows how Orbit can repair both imbalance and



(a) Original unmitigated sizes.

(b) Repairing imbalance alone can make bias worse.

(c) Repairing bias alone can make imbalance worse.

(d) Using ORBIT to repair both imbalance and bias.

(e) Using ORBIT to repair imbalance without making bias worse.

(f) Using ORBIT to repair bias without making imbalance worse.

Figure 2: Different oversampling sizes for repairing imbalance and/or bias.

bias together. While this is useful, Figure 1 demonstrated that the highest repair for the dataset does not always yield the best metrics for an estimator trained on that data. Therefore, ORBIT lets users tune imbalance and bias separately. Figure 2(e) shows a solution that repairs class imbalance while carefully controlling the new group bias to be no different from the original, i.e., $n_{\text{di}} = o_{\text{di}}$. Similarly, Figure 2(f) shows a solution that repairs group bias while keeping $n_{\text{ci}} = o_{\text{ci}}$.

We can control imbalance and bias in a more fine-grained manner than the examples in Figure 2. Let $\delta_{\text{imbalance}}$ and δ_{bias} be continuous hyperparameters controlling the repair level for class imbalance and group bias, respectively. Further, denote by n_{ci} the new class imbalance and by n_{di} the new disparate impact after oversampling, as computed from the new sizes n_{00}, n_{01}, n_{10}, and n_{11}. The problem statement is to pick these new sizes and oversample to satisfy the following two constraints:

$$n_{\text{ci}} = o_{\text{ci}} + \delta_{\text{imbalance}}(1 - o_{\text{ci}}) \qquad \wedge \qquad n_{\text{di}} = o_{\text{di}} + \delta_{\text{bias}}(1 - o_{\text{di}}) \tag{2}$$

3 ORBIT Algorithm

The core of ORBIT consists of computing the desired intersection sizes and then oversampling the data accordingly. That said, there are also hard-learned lessons for the workflow around this core that may trip up the unwary. Therefore, Figure 3 shows ORBIT along with a recommended workflow for evaluating estimators, suitable for AutoML. The rest of this section explains the steps from Figure 3 given a matrix X of features and a vector y of binary class labels for each instance.

Step 1: Train-test split. This step partitions X, y into X_{train}, y_{train} and X_{test}, y_{test}. One potential pitfall with oversampling is that the test data may contain a synthetic instance obtained by oversampling a real instance or vice versa, causing over-fitting [17]. This must be prevented by only applying oversampling to the training data, i.e., experiments must first split the data and only then perform oversampling. To ensure this by construction, we implemented ORBIT as a meta-estimator that takes

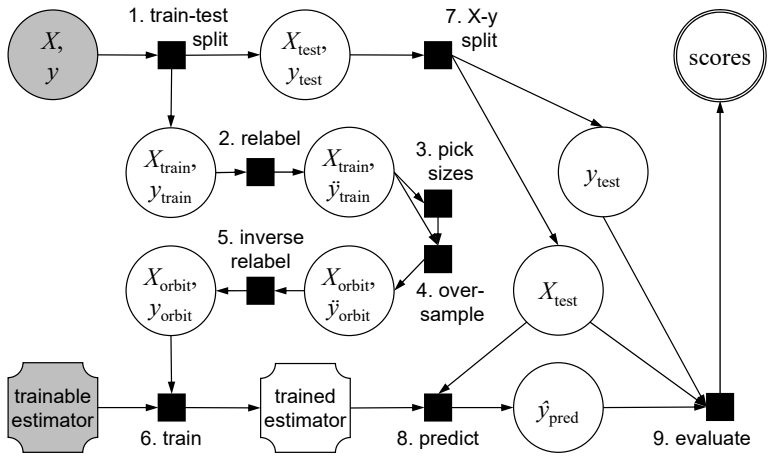


Figure 3: Overview of the Orbit algorithm in the context of an estimator evaluation workflow.

the downstream estimator as an argument, oversamples X_{train}, y_{train} during fitting, and passes an unmodified X_{test} through to the downstream estimator when predicting. The meta-estimator itself can serve as an argument to an automated hyperparameter tuning tool that uses a cross-validation split. We recommend stratifying splits by both groups and classes, because in highly imbalanced data, a non-stratified split risks some intersections of groups and classes being tiny or even empty.

Step 2: Relabel. This step changes X_{train}, y_{train} into X_{train}, $\ddot{y}_{\text{train}}$, where $\ddot{y}_{\text{train}}$ contains *diaeresis labels*. Let $get_group : x_i \rightarrow \{0, 1\}$ be a function that indicates, for a given row x_i representing one instance, whether that instance belongs to the unprivileged (0) or privileged (1) group. For example, *get_group* might retrieve a numeric *age* attribute and apply a threshold to group instances into young or old. Similarly, let $get_class : y_i \rightarrow \{0, 1\}$ be a function that maps labels to unfavorable (0) or favorable (1) classes. The diaeresis labels are simply pairs $\ddot{y}_i = \langle get_group(x_i), get_class(y_i) \rangle \in \{0, 1\}^2$.

Step 3: Pick sizes. Given X_{train}, $\ddot{y}_{\text{train}}$, this step computes the desired new sizes n_{00}, n_{01}, n_{10}, and n_{11}. The diaeresis labels $\ddot{y}_{\text{train}}$ induce a partition on the instances into subsets that share the same label. Orbit computes the original sizes o_{00}, o_{01}, o_{10}, and o_{11} of these intersections, and from these, computes the original class imbalance o_{ci} and disparate impact o_{di} (Equation 1). Next, it uses Equation 2 to compute the desired new class imbalance n_{ci} and disparate impact n_{di} based on hyperparameters $\delta_{\text{imbalance}}$ and δ_{bias}. Without loss of generality, assume $n_{\text{ci}} \leq 1$ (if not, swap classes) and $n_{\text{di}} \leq 1$ (if not, swap groups). The desired solution needs to satisfy two equations:

$$\frac{n_{00} + n_{10}}{n_{01} + n_{11}} = n_{\text{ci}} \qquad \bigwedge \qquad \frac{n_{01}/(n_{01} + n_{00})}{n_{11}/(n_{11} + n_{10})} = n_{\text{di}} \tag{3}$$

Given four unknowns ($n_{00}, n_{01}, n_{10}, n_{11}$), these equations permit many possible solutions. Since $n_{\text{ci}} \leq 1 \wedge n_{\text{di}} \leq 1$, we can eliminate one unknown by simply setting $n_{11} = o_{11}$. Next, we will strive to minimize oversampling the intersection of the unprivileged group with members receiving unfavorable class labels. To do this, we will find the minimum n_{00} for which solving the above equation satisfies $n_{00} \geq o_{00} \wedge n_{01} \geq o_{01} \wedge n_{10} \geq o_{10} \wedge n_{11} \geq o_{11}$. Having eliminated two unknowns, n_{11} and n_{00}, all that remains is to solve for the remaining two unknowns, n_{01} and n_{10}. It can be shown that the equations above imply $n_{10} = \frac{1}{2}(\sqrt{b^2 - 4c} - b)$, where $b = n_{00} + n_{11} - n_{11}n_{\text{ci}} - n_{11}n_{\text{di}}$ and $c = n_{00}n_{11} + n_{11}n_{11}n_{\text{ci}}n_{\text{di}} - n_{00}n_{11}n_{\text{ci}}n_{\text{di}} - n_{11}n_{11}n_{\text{ci}} - n_{00}n_{11}n_{\text{di}}$. And finally, $n_{01} = \frac{n_{00}}{n_{\text{ci}}} + \frac{n_{10}}{n_{\text{ci}}} - n_{11}$. See Appendix A for the detailed size selection scheme.

After picking the sizes, Orbit obtains the final numbers by reversing the swap of classes and groups, if any, that was needed to ensure n_{ci} and n_{di} are at most one. This has the effect that Orbit repairs imbalance or bias symmetrically for whichever class or group exhibits it in the data.

Step 4: Oversample. Given X_{train}, $\ddot{y}_{\text{train}}$ and the desired new sizes n_{00}, n_{01}, n_{10}, and n_{11}, this step creates more balanced training data X_{orbit}, $\ddot{y}_{\text{orbit}}$. This step applies SMOTE [6], which stands for Synthetic Minority Oversampling TEchnique, to each of the four intersections separately. While the intersection has not yet reached its new desired size, repeat the following:

(i) Randomly choose a non-synthetic instance r from the given intersection to oversample.
(ii) Find the k non-synthetic instances that are nearest neighbors of r, and randomly choose an instance v among them that is in the same intersection as r.
(iii) Randomly choose a number φ between 0 and 1.
(iv) Create a new synthetic instance $s = r + \varphi(v - r)$.

One potential problem is that the group of a synthetic instance s might differ from that of the real instance r it was derived from. This can be avoided by ensuring that the function *get_group* satisfies $get_group(r) = get_group(r + \varphi(v - r))$ for any two instances r and v with $get_group(r) = get_group(v)$ and $0 \leq \varphi \leq 1$. Another technical issue is that ORBIT should handle categorical features; for instance, protected attributes are often categorical. Fortunately, the imbalanced-learn library provides SMOTE variants that handle categorical features [13].

Step 5: Inverse relabel. This step changes X_{orbit}, $\ddot{y}_{\text{orbit}}$ into X_{orbit}, y_{orbit}. It simply retrieves the class component of the diaeresis label $\ddot{y}_{\text{orbit}}$ and applies the inverse of the *get_class* function.

Step 6: Train. Given the oversampled training data X_{orbit}, $\ddot{y}_{\text{orbit}}$ and the trainable downstream estimator, this step creates the trained estimator. Recall that the trainable downstream estimator is itself an argument to the meta-estimator. In fact, an AutoML tool can even treat it as a hyperparameter, to be tuned for automated algorithm selection. In our experiments, the downstream estimator is actually a pipeline comprising first an ordinal encoder for categorical features (forwarding continuous features as-is), followed by one of the classifiers listed in the caption of Figure 1.

Step 7: X-y split. This step separates X_{test}, y_{test} into X_{test} and y_{test}.

Step 8: Predict. This step applies the trained estimator on X_{test} to obtain predictions $\hat{y}_{\text{pred}}$.

Step 9: Evaluate. The last and final step of the workflow from Figure 3 computes scores. Unlike accuracy metrics that only require ground-truth labels y_{test} and predicted labels $\hat{y}_{\text{pred}}$, fairness metrics typically also require X_{test} to inspect protected attributes. This step can compute multiple metrics separately, such as symmetric disparate impact (DI) and balanced accuracy (BA) serving as the x-axis and y-axis in Figure 1. In addition, for use with a single-objective optimizer, this step can also compute blended metrics. Since there is no consensus on the best approach for blending metrics, this paper considers a variety of approaches:

- Arithmetic mean, $\mathsf{AM} = \frac{\mathsf{BA}+\mathsf{DI}}{2}$, is the most familiar and straight-forward to explain.
- Geometric mean, $\mathsf{GM} = \sqrt{\mathsf{BA} \cdot \mathsf{DI}}$, quantifies the area of Pareto dominance in the scatter plot.
- Harmonic mean, $\mathsf{HM} = \frac{2 \cdot \mathsf{BA} \cdot \mathsf{DI}}{\mathsf{BA}+\mathsf{DI}}$, also encourages a larger area of Pareto dominance while tolerating differences in scale between the component metrics better than geometric mean does.
- Hard threshold, $\mathsf{HT} = \begin{cases} \frac{\mathsf{DI}}{2 \cdot 0.8} & \text{if } \mathsf{DI} < 0.8 \\ \mathsf{BA} & \text{otherwise} \end{cases}$,
  focuses exclusively on DI when DI is below the fairness threshold of 0.8, and on BA above.
- Soft threshold, $\mathsf{ST} = \begin{cases} \mathsf{BA} \cdot (\frac{\mathsf{DI}}{0.8})^4 & \text{if } \mathsf{DI} < 0.8 \\ \mathsf{BA} & \text{otherwise} \end{cases}$,
  focuses mostly on DI when DI is below the fairness threshold of 0.8, but also rewards improvements to BA in that regime a little, and focuses on BA when DI is above the 0.8 threshold.

Table 1: Datasets in ascending order of #ROWS. Columns o_{ci} and o_{di} show original class imbalance and disparate impact. Datasets marked with † are from AIF360, the remainder are from OpenML.

DATASET	DESCRIPTION	PROTECTED ATTRIBUTE	#ROWS	o_{ci}	o_{di}
Ricci	Fire department promotion exam results	race	118	0.90	0.50
TAE	University teaching assistant evaluation	TA-native-speaker	151	0.50	1.74
Credit-g	German bank data quantifying credit risk	sex,age	1,000	0.43	0.75
Titanic	Survivorship of Titanic passengers	sex	1,309	0.62	0.26
COMPAS Violent†	Correctional offender violent recidivism	sex,race	3,377	0.21	0.82
COMPAS†	Correctional offender recidivism	sex,race	5,278	0.89	0.69
SpeedDating	Speed dating experiment at business school	samerace,imp-samerace	8,378	0.20	0.85
Nursery	Slovenian nursery school application results	parents	12,960	0.45	0.46
MEPS 19†	Utilization results from Panel 19 of MEPS	RACE	15,830	0.21	0.49
MEPS 20†	Same as MEPS 19 except for Panel 20	RACE	17,570	0.21	0.49
Bank	Portuguese bank subscription predictions	age	45,211	0.13	0.84
Adult	1994 US Census salary data	sex,race	48,842	0.31	0.23

We chose to formulate this as a single-objective hyperparameter optimization or HPO problem by considering different strategies of combining the predictive and fairness performance. This could also have been posed as multi-objective HPO. But our HPO problem is more constrained since we really desire the fairness metric to be above 80%, which does not directly fit into usual multi-objective HPO solvers, while some of our combined objectives do encode this constraint explicitly. Furthermore, we often need to find a single solution (which our scheme produces) instead of requiring the user to select from a (potentially large) set of Pareto-optimal solutions.

4 Empirical Study

This section empirically studies three research questions:
RQ1. How do different imbalance mitigators affect fairness and predictive performance?
RQ2. How do different bias mitigators affect fairness and predictive performance?
RQ3. How do different approaches for blending metrics affect single-objective AutoML?

Datasets. We consider 12 binary classification datasets (4 from AIF360 [1] and 8 from OpenML [19]) shown in Table 1. In only 3 of the 12 datasets, the disparate impact would be considered as fair (with o_{di} above 0.8), and only 2 datasets are relatively balanced (with o_{ci} around 0.9), highlighting the need to study bias in conjunction with class imbalance. While MEPS 19 and MEPS 20 are different datasets with no overlap, their imbalance and fairness characteristics are similar. On the other hand, even though COMPAS Violent is a subset of COMPAS, their characteristics are quite different: COMPAS Violent is significantly less balanced but has better base disparate impact.

For each dataset and each configuration, we perform a total of six runs, comprising two repeats of 3-fold cross validation. Figures 4, 5, and 6 show the results. Each figure has 12 subfigures, one for each dataset, sorted by size. The axes, metrics, error bars, and dotted and dashed vertical lines have the same meaning as in Figure 1. Symmetric disparate impact is consistently more noisy (horizontal error bars) than balanced accuracy (vertical error bars), an effect that would be even more visible if both axes used the same scale. Larger datasets tend to have smaller error bars.

RQ1: How do different imbalance mitigators affect fairness and predictive performance? Figure 4 shows results for several mitigators that either repair only class imbalance or use rebalancing to repair group bias. As expected, SMOTE [6], and its variants SMOTEN/SMOTENC depending on the data, improve balanced accuracy over unmitigated LR significantly in 4 datasets, while never being significantly worse. ORBIT improves disparate impact over SMOTE for most datasets while maintaining the same level of balanced accuracy. Fair-SMOTE [4] usually performs similarly to ORBIT, but ORBIT has the additional advantage of being tunable, as shown in Figure 1. Reweighing [10]

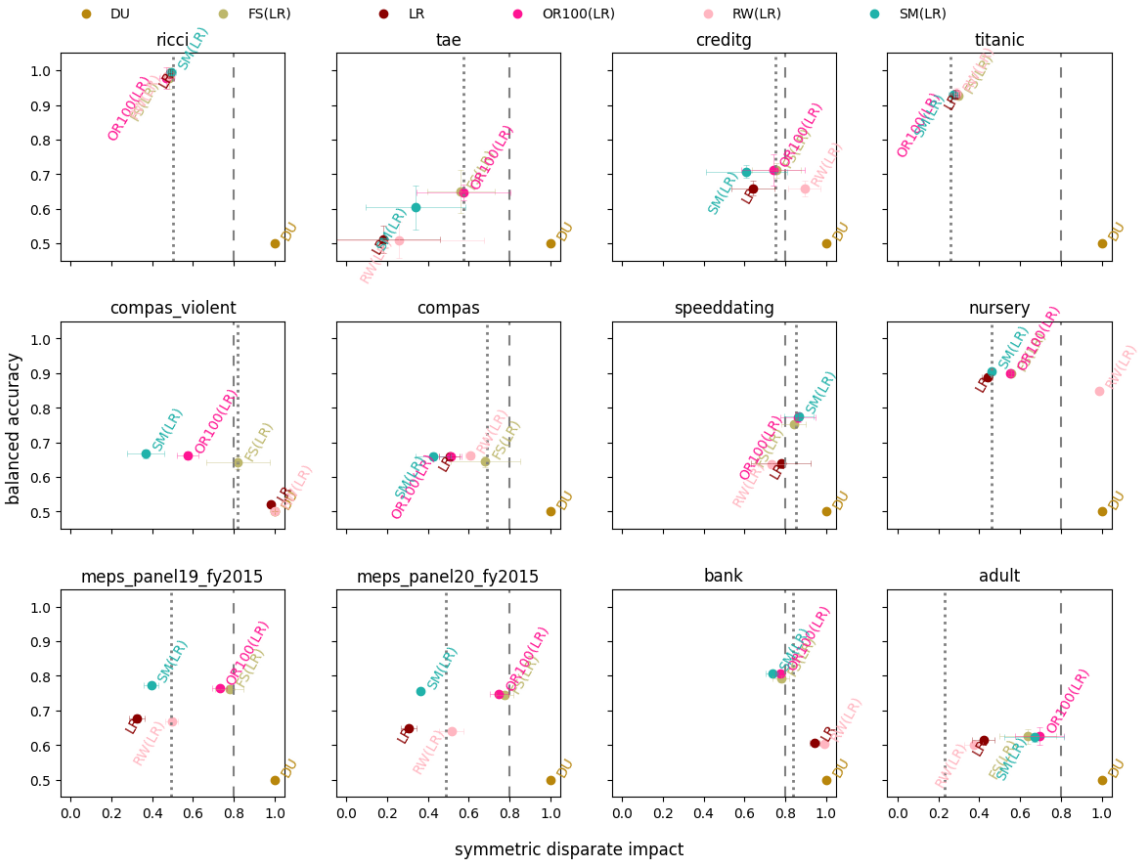


Figure 4: Comparing class imbalance mitigators. DU is the dummy estimator, FS is Fair-SMOTE [4], LR is unmitigated logistic regression, OR100 is Orbit with $\delta_{\text{imbalance}} = 0.8$ and $\delta_{\text{bias}} = 1$, RW is reweighing [10], and SM is Smote [6].

usually does worse than the oversampling based approaches, but excels for creditg and nursery. Overall, Figure 4 shows that even without hyperparameter tuning, Orbit is very competitive.

RQ2: How do different bias mitigators affect fairness and predictive performance? Figure 5 compares Orbit against nine other bias mitigators from AIF360 [1]. In general, different mitigators trade-off predictive performance and bias to different degrees, sometimes tracing out a Pareto frontier. Orbit is Pareto-optimal for most of the datasets, more often than any other mitigator, since it tends to yield high balanced accuracy while also improving fairness. RO (reject-option classification [11]) is also often a front-runner. However, RO sometimes degenerates to perform like a dummy classifier, with optimal fairness but low accuracy. There is no "one size fits all" for bias mitigators, and it is important to try available options rigorously.

RQ3: How do different approaches for blending metrics affect single-objective AutoML?. Figure 6 shows results from using Hyperopt [2] to jointly tune the hyperparameters and select the downstream estimator passed to Orbit. We let Hyperopt tune $\delta_{\text{imbalance}}$ and δ_{bias}, both in the range from 0 to 1, while selecting among a choice between logistic regression, support vector machine, and gradient boosting. For each blending approach from Step 9 of Section 3, we launch 3 Hyperopt runs with randomly shuffled data, where each run has 20 trials, and each trial performs 3-fold cross validation. The scatter plot shows the average result of the best configuration found, with error bars for standard deviation across the 3 runs. Overall, geometric mean is usually effective at

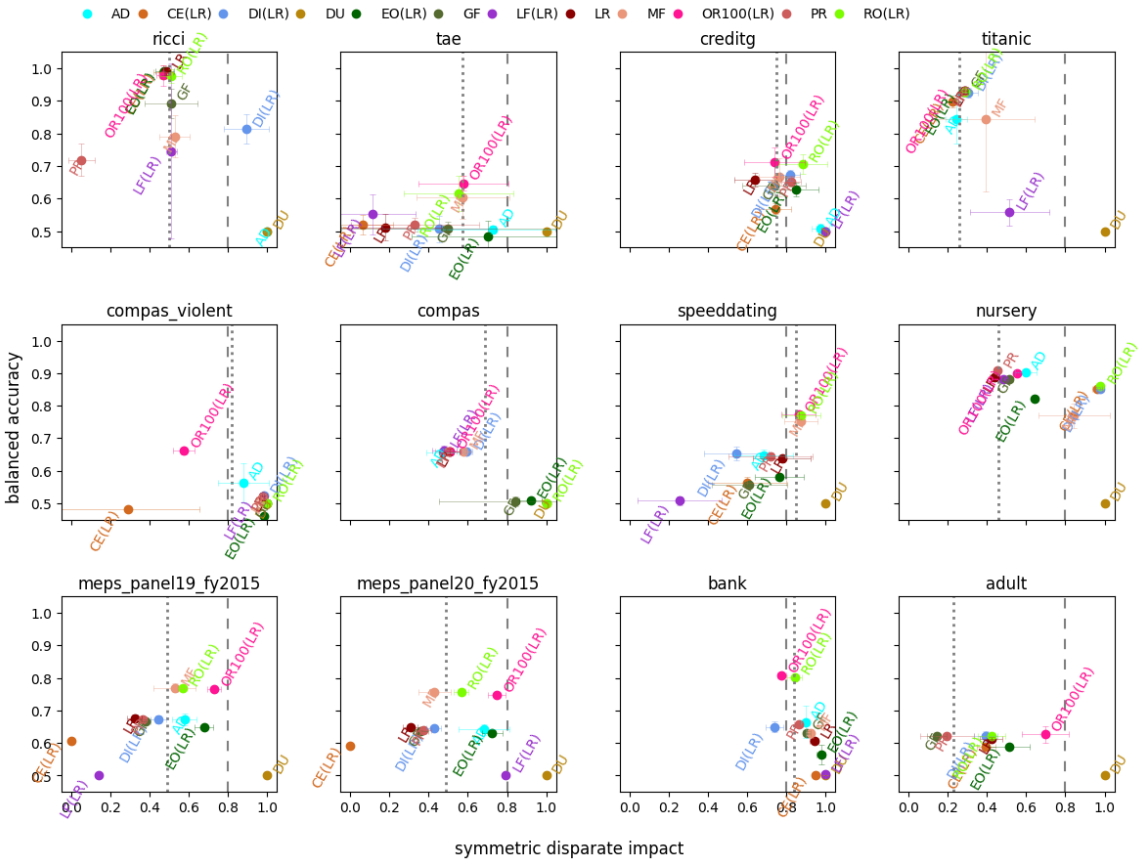


Figure 5: Comparing group bias mitigators. AD is adversarial debiasing, CE is calibrated equalized-odds post-processing, DI is disparate impact remover, DU is the dummy estimator, EO is equalized-odds post-processing, GF is gerry-fair classifier, LF is learning fair representations, LR is unmitigated logistic regression, MF is meta-fair classifier, OR100 is Orbit with $\delta_{\text{imbalance}} = 0.8$ and $\delta_{\text{bias}} = 1$, PR is prejudice remover, and RO is reject-option classification.

finding an Orbit configuration that does well on both axes. The threshold approaches sometimes do well at approaching or surpassing a disparate impact of 0.8, but struggle from noise with some datasets and degenerate to perform like dummy for some others. Innovation in reining in noise in the metrics across folds could make AutoML more effective at mitigating bias.

5 Related Work

Smote [6] is a popular class imbalance mitigator. We show how to extend it to also mitigate group bias. Fair-SMOTE [4] applies Smote to oversample all non-majority intersections of groups and classes to the size of the majority, thereby allowing only the highest level of mitigation for imbalance and bias. Furthermore, this technique can lead to unnecessarily high amounts of synthetic data, which can adversely affect the modeling both statistically and computationally. FOS [8] also extends Smote for bias mitigation. It takes a slightly different approach, internally class-balancing each group. Orbit reduces to FOS when the repair level is set to the highest value for both imbalance and bias, but FOS does not consider intermediate levels of mitigation. Furthermore, the FOS paper only uses 3 datasets and does not report error bars, making it hard to judge the generality of its findings. Reweighing [10] is a group bias mitigator that, like Orbit, effectively changes the total "size" of certain data subsets. It does not address class imbalance. Undersampling-multivariate [18] is a group

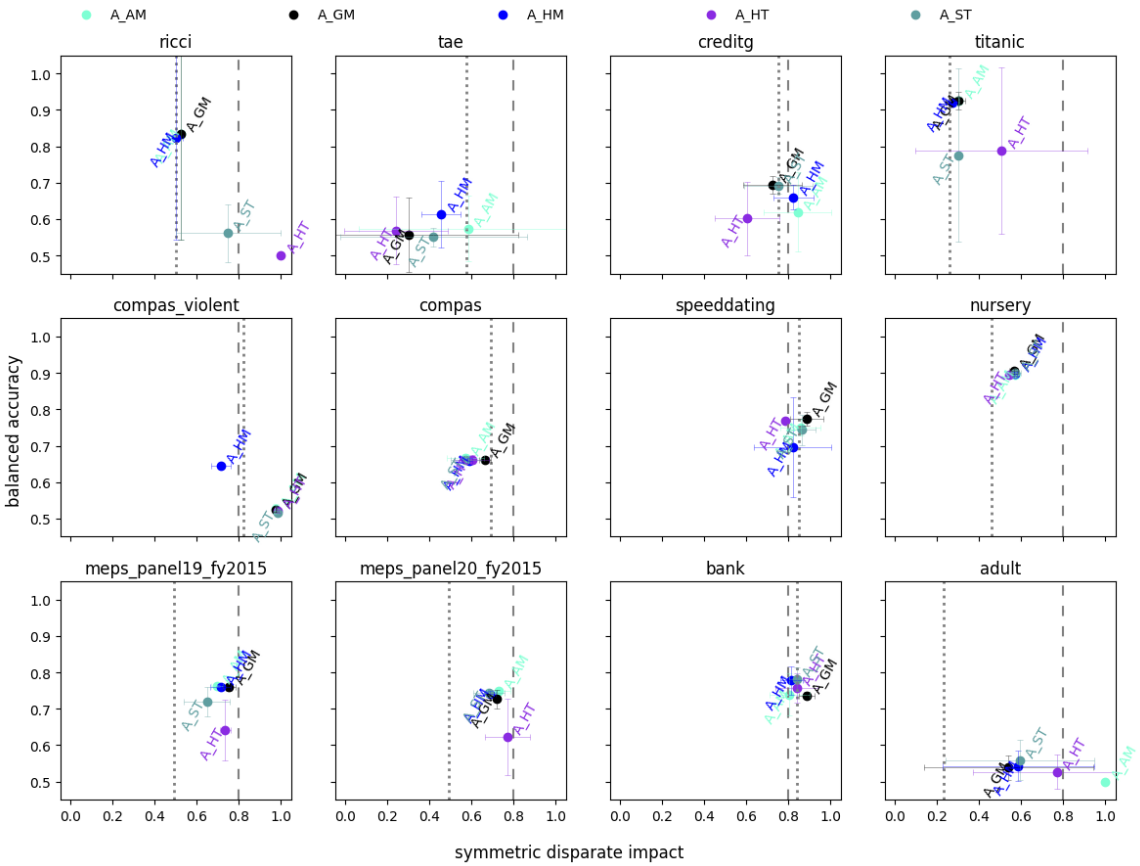


Figure 6: Comparing different approaches for blending metrics into an AutoML objective. AM, GM, and HM are arithmetic, geometric, and harmonic mean. HT and ST are hard and soft threshold.

bias mitigator that undersamples where Orbit oversamples. The authors discuss undersampling by group or class label or both, but only evaluate it on 2 datasets.

Fairness-aware AutoML [20] incorporates fairness either as (i) an objective alongside the predictive performance and utilizes multi-objective hyperparameter optimization [15, 12], or (ii) as a constraint for a given threshold [14, 16]. However, neither studies the class imbalance and bias mitigation simultaneously as we do. Existing fairness-aware AutoML results [5, 7] seem to indicate that we can achieve a better accuracy-bias tradeoff by tuning model hyperparameters than by utilizing bias-mitigation techniques, although, to the best of our understanding, these existing studies do not try to optimize over the mitigation hyperparameters (bias as well as imbalance) as we do. In our current evaluation, we consider a blended metric for hyperparameter optimization, studying the effect of combining accuracy and bias in different ways. Given the search space definition we propose, we can consider a constrained multi-objective hyperparameter optimizer.

6 Conclusion

This paper demonstrates that repairing bias or imbalance separately can harm imbalance or bias, respectively. Furthermore, full mitigation is not always best, and indeed, it is difficult to decide ahead of time which mitigation level to apply. Next, this paper introduces Orbit, an algorithm that mitigates imbalance and bias together, and that allows the user to choose the exact repair levels for both. This paper discusses how to use Orbit in an AutoML context, and includes an extensive experimental evaluation.

7 Broader Impact Statement

After careful reflection, the authors have determined that this work presents no notable negative impacts to society or the environment.

8 Submission Checklist

1. For all authors…

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes]

   (c) Did you discuss any potential negative societal impacts of your work? [N/A]

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

2. If you are including theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [Yes]

   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.

3. If you ran experiments…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] In the supplemental material (tgz file).

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] In the supplemental material (tgz file).

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] In the supplemental material (tgz file).

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes]

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes]

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] Figure 1 reports results for different settings of the hyperparameters and different downstream estimators.

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes]

   (i) Did you compare performance over time? [N/A]

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes] We performed multiple runs, but did not report random seeds, since we did not record them.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A]

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] All experiments ran on the laptop of one of the authors without GPU, with the runtime recorded in the raw results.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

(a) If your work uses existing assets, did you cite the creators? [Yes] We used datasets from OpenML [19] and AIF360 [1], models from scikit-learn [3], imbalanced-learn [13], and Hyperopt [2].

(b) Did you mention the license of the assets? [Yes] BSD (3-Clause) license: scikit-learn, OpenML, Hyperopt; Apache-2.0 license: AIF360; MIT license: imbalanced-learn.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The supplemental material has code for running experiments with requirements.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

[1] Bellamy, R. K. E., Dey, K., Hind, M., Hoffman, S. C., Houde, S., Kannan, K., Lohia, P., Martino, J., Mehta, S., Mojsilovic, A., Nagar, S., Ramamurthy, K. N., Richards, J., Saha, D., Sattigeri, P., Singh, M., Varshney, K. R., and Zhang, Y. (2018). AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias.

[2] Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. (2015). Hyperopt: a Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1).

[3] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: Experiences from the scikit-learn project.

[4] Chakraborty, J., Majumder, S., and Menzies, T. (2021). Bias in machine learning software: Why? how? what to do? In *Symposium on the Foundations of Software Engineering (FSE)*, pages 429–440.

[5] Chakraborty, J., Xia, T., Fahid, F. M., and Menzies, T. (2019). Software engineering for fairness: A case study with hyperparameter optimization.

[6] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research (JAIR)*, (16):321–357.

[7] Cruz, A. F., Saleiro, P., Belém, C., Soares, C., and Bizarro, P. (2021). Promoting fairness through hyperparameter optimization. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1036–1041. IEEE.

[8] Dablain, D., Krawczyk, B., and Chawla, N. (2022). Towards a holistic view of bias in machine learning: Bridging algorithmic fairness and imbalanced learning.

[9] Feldman, M., Friedler, S. A., Moeller, J., Scheidegger, C., and Venkatasubramanian, S. (2015). Certifying and removing disparate impact. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 259–268.

[10] Kamiran, F. and Calders, T. (2012). Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33:1–33.

[11] Kamiran, F., Karim, A., and Zhang, X. (2012). Decision theory for discrimination-aware classification. In *International Conference on Data Mining (ICDM)*, pages 924–929.

[12] Karl, F., Pielok, T., Moosbauer, J., Pfisterer, F., Coors, S., Binder, M., Schneider, L., Thomas, J., Richter, J., Lang, M., et al. (2022). Multi-objective hyperparameter optimization–an overview.

[13] Lemaître, G., Nogueira, F., and Aridas, C. K. (2017). Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research (JMLR)*, 18(17):1–5.

[14] Liu, S., Ram, P., Vijaykeerthy, D., Bouneffouf, D., Bramble, G., Samulowitz, H., Wang, D., Conn, A., and Gray, A. (2020). An ADMM based framework for AutoML pipeline configuration. In *Conference on Artificial Intelligence (AAAI)*, pages 4892–4899.

[15] Morales-Hernández, A., Van Nieuwenhuyse, I., and Rojas Gonzalez, S. (2022). A survey on multi-objective hyperparameter optimization algorithms for machine learning. *Artificial Intelligence Review*, pages 1–51.

[16] Perrone, V., Donini, M., Zafar, M. B., Schmucker, R., Kenthapadi, K., and Archambeau, C. (2021). Fair Bayesian optimization. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 854–863.

[17] Santos, M. S., Soares, J. P., Abreu, P. H., Araujo, H., and Santos, J. (2018). Cross-validation for imbalanced datasets: Avoiding overoptimistic and overfitting approaches. *Computational Intelligence Magazine (CEM)*, 13(4):59–76.

[18] Valentim, I., Lourenço, N., and Antunes, N. (2019). The impact of data preparation on the fairness of software systems. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 391–401.

[19] Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). OpenML: Networked science in machine learning. *SIGKDD Explorations Newsletter*, 15(2):49–60.

[20] Weerts, H., Pfisterer, F., Feurer, M., Eggensperger, K., Bergman, E., Awad, N., Vanschoren, J., Pechenizkiy, M., Bischl, B., and Hutter, F. (2023). Can fairness be automated? guidelines and opportunities for fairness-aware AutoML.

A Picking sizes for repair

With $n_{11} = o_{11}$, for a size n, let us define the following terms:

$$b(n) \triangleq n + n_{11} - n_{11}n_{\text{ci}} - n_{11}n_{\text{di}}, \quad c(n) \triangleq nn_{11} + n_{11}^2 n_{\text{ci}}n_{\text{di}} - nn_{11}n_{\text{ci}}n_{\text{di}} - n_{11}^2 n_{\text{ci}} - nn_{11}n_{\text{di}}. \tag{4}$$

This is exactly the definition of b and c in Section 3. Then, we search for the size n_{00}, and consequently n_{10}, n_{01} as follows, given $n_{11}, n_{\text{ci}}, n_{\text{di}}$ and the old sizes $o_{00}, o_{01}, o_{10}, o_{11}$:

- For $n \in [o_{00}, (o_{00} + o_{01} + o_{10} + o_{11})]$:
  - Compute $b(n)$ and $c(n)$ as in equation 4
  - If $b^2(n) < 4c(n)$ `continue` with next iteration
  - Set $n_{10} \leftarrow \frac{1}{2}(\sqrt{b^2(n)) - 4c(n)} - b(n))$
  - Set $n_{01} \leftarrow \frac{n_{00}}{n_{\text{ci}}} + \frac{n_{10}}{n_{\text{ci}}} - n_{11}$
  - If $n_{10} < o_{10}$ or $n_{01} < o_{01}$ `continue` with next iteration
  - $n_{00} \leftarrow n$ and `break` from the loop

With this procedure, we have selected the smallest n_{00} that satisfies the desired repair level constraints (equation 3). We try to select the smallest n_{00} to ensure that the least amount of data is synthetically generated since that can lead to statistical and computational issues.

We arrrived at the formulas for b and c as follows. First, we solved the n_{ci} equation for n_{01}. Next, we substituted that formula for n_{01} into the n_{di} equation. Then, we rewrote the resulting formula into the standard form of a quadratic equation for n_{10}. Then we substituted b and c for the appropriate terms in the quadratic equation. After computing b and c, we used those to compute n_{10}. Finally, we substituted that solution for n_{10} into the rewritten n_{ci} equation to obtain n_{01}.

B Results with different repair levels

Figure 1 in the main paper showed results for Orbit with different repair levels for one dataset only, namely meps20. For completeness, we show the results for all 12 datasets in Figures 7–10.

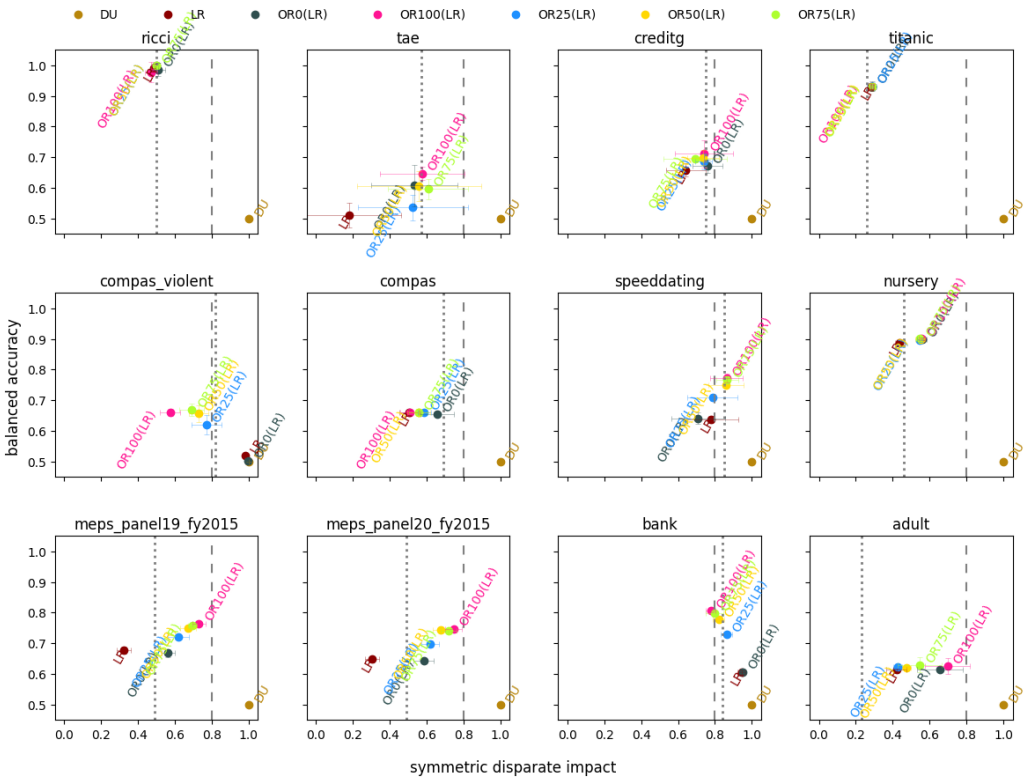


Figure 7: The effect of repair levels with Orbit and LR (logistic regression).

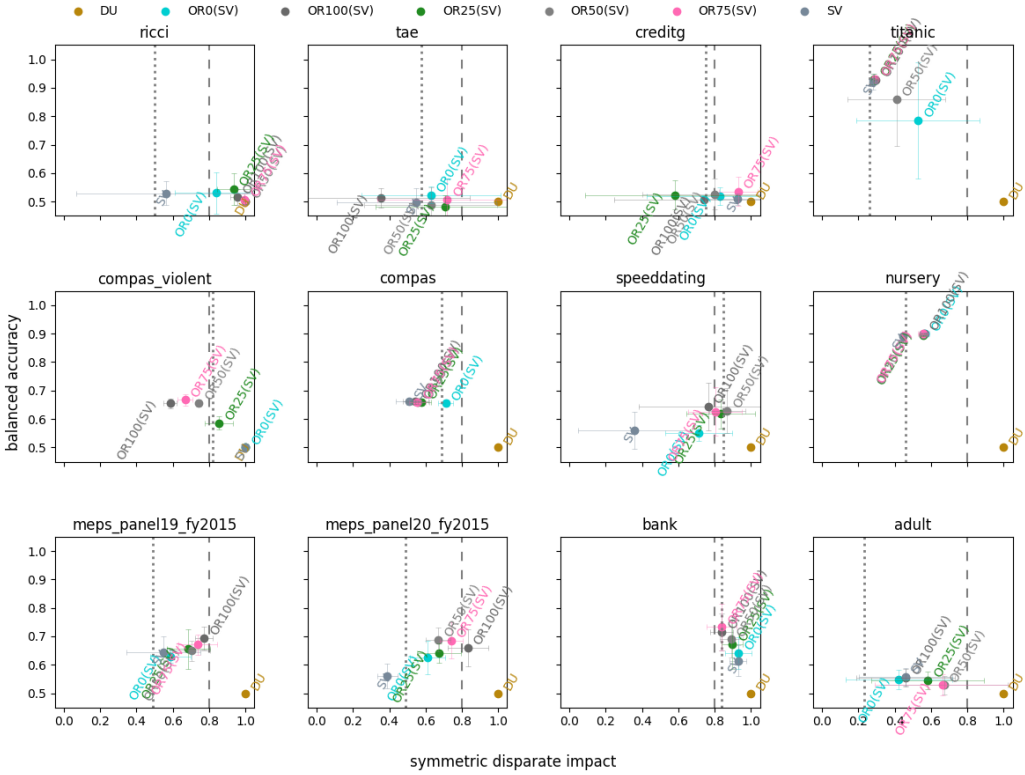


Figure 8: The effect of repair levels with Orbit and SV (support vector machine).

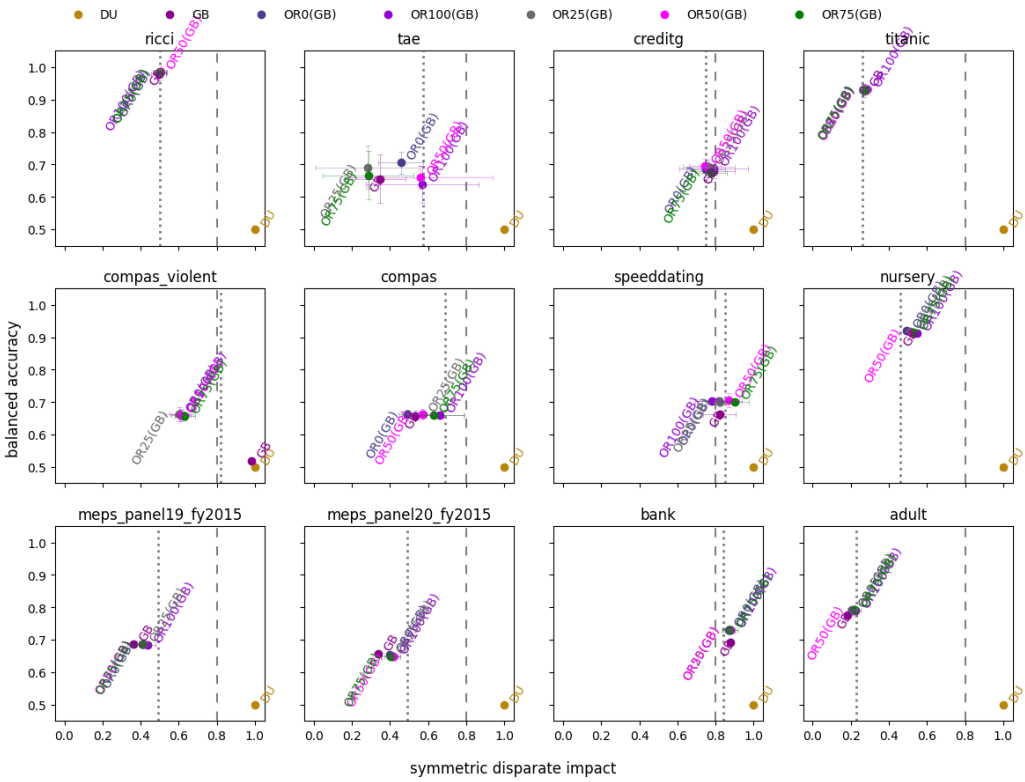


Figure 9: The effect of repair levels with Orbit and GB (gradient boosting).

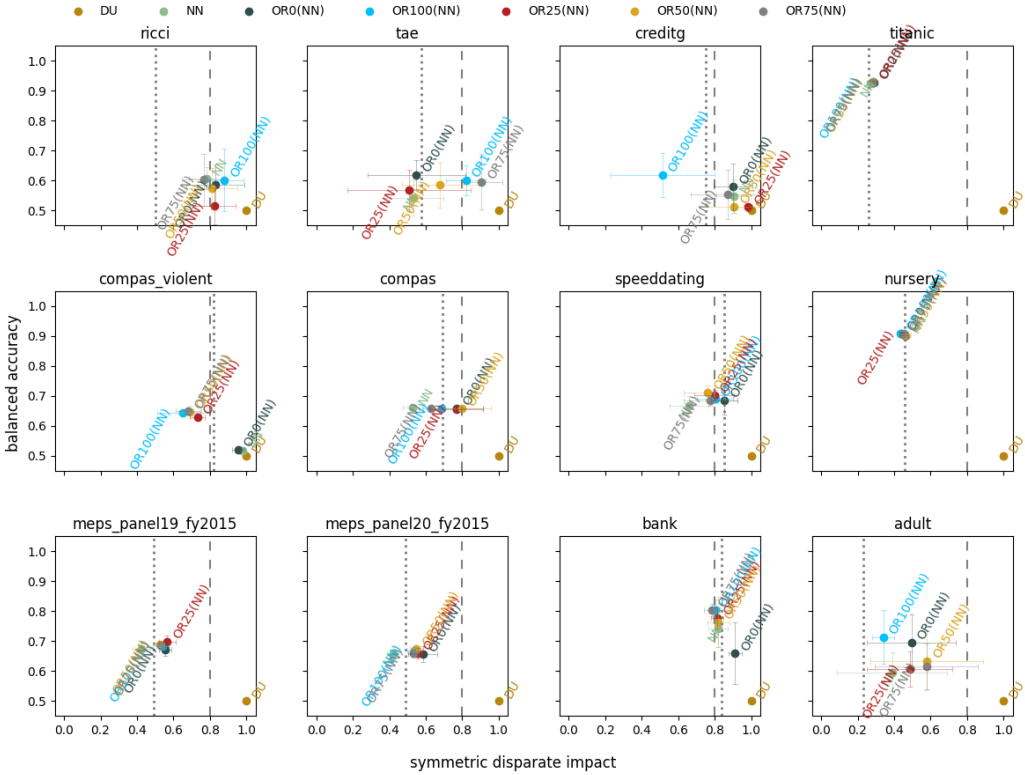


Figure 10: The effect of repair levels with Orbit and NN (neural network).