Boosting for Fairness-Aware Classification (Machine Learning 2023 Course)

Julia Kudryavtseva [1] **Xavier Aramayo Carrasco** [1]
Alisa Kalacheva [1]
Vo Ngoc Bich Uyen [1] **Alexander Lepinskikh** [1]

Abstract

This report investigates the effectiveness of four state-of-the-art fairness-aware classification algorithms, namely SMOTEBoost, AdaFair, CUSBoost, and RAMOBoost, in mitigating bias in four real-world datasets: Adult census, COMPAS, Bank, and KDD. These datasets are notorious for being imbalanced and biased against certain protected groups, and therefore pose significant challenges for building fair and accurate classifiers.

Github repo: Boosting-for-Fairness-Aware-Classification

Presentation file: JAX Boosting for Fairness-Aware Classification

1. Introduction

Machine learning algorithms are widely used for decision-making in various domains, including employment, criminal justice, and finance. However, these algorithms are prone to bias and discrimination against certain protected groups, such as gender, race, and age. To address this issue, fairness-aware classification algorithms have been developed, which aim to ensure fairness and equity in the decision-making process without compromising the accuracy and overall performance of the models.

To address this problem, several techniques and algorithms have been developed, which can be broadly categorized into two groups: data-level and algorithm-level methods.

Data-level methods aim to balance the class distribution by either oversampling the minority class, undersampling the majority class, or generating synthetic examples. Examples of such techniques include Random Oversampling Minority Class (ROSE) (Moreo et al., 2016), Synthetic Minority Over-sampling Technique (SMOTE) (Chawla et al., 2002), and Tomek Links (Elhassan et al., 2016). These methods, however, may not be suitable for highly imbalanced datasets or can cause overfitting.

Algorithm-level methods, on the other hand, modify the algorithm's objective or decision boundary to account for class imbalance. Examples of such algorithms include Cost-Sensitive Learning (Thai-Nghe et al., 2010), AdaBoost (Li & Jain, 2009), and ensemble methods such as Balanced Random Forest (Chen & Breiman, 2004). These methods can improve the performance of classifiers on the minority class without sacrificing overall accuracy.

In the context of this report, we focus on the use of four fairness-aware classification algorithms, namely SMOTEBoost, AdaFair, CUSBoost, and RAMOBoost, which combine both data-level and algorithm-level techniques to mitigate bias and improve fairness in the context of unbalanced datasets, all the algorithms are variations of the original AdaBoost algorithm, they mostly differ in the technique used to obtain samples from the unbalanced datasets.

Some of the algorithms analyzed in this report have previously been explored in the original AdaFair paper (Iosifidis & Ntoutsi, 2019), and this report serves as an extension of that work. The datasets utilized in both AdaFair and this report are the Adult Census Income, Bank, COMPAS, and KDD Census Income datasets. These datasets were preprocessed independently, using the same methods as in the original paper, to ensure reproducibility.

To evaluate the performance of these algorithms, we use several fairness metrics, including equalized odds, which measures the difference in true positive rates (TPRs) and true negative rates (TNRs) between protected and unprotected groups. Accuracy and balanced accuracy were utilized to evaluate the predictive performance of the models.

The main contributions of this report are as follows:

- To Reproduce the results of the AdaFair original paper.

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Julia Kudryavtseva <Julia.Kudryavtseva@skoltech.ru>, Xavier Aramayo Carrasco <aramayo.carrasco@skoltech.ru>, Alisa Kalacheva <Alisa.kalacheva@skoltech.ru>, Vo Ngoc Bich Uyen <VoNgoc.BichUyen@skoltech.ru>, Alexander Lepinskikh <a.korotin@skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

- To extend the results in the paper by reporting the scores of other similar approaches.

- To provide clean and reproducible code of the implemented methods.

2. Related work

Several algorithms and techniques have been proposed to address the problem of biased datasets and datasets in machine learning. In this section, we briefly discuss some of the most relevant works, we also explore techniques that are not related to the use of Adaboost to provide a better overview of different alternatives.

A popular approach to addressing class imbalance is to use cost-sensitive learning (Thai-Nghe et al., 2010). This technique modifies the cost matrix of the classifier to penalize errors on the minority class more heavily than the majority class. While this approach can be effective, it relies heavily on the quality of the cost matrix and can be sensitive to the choice of cost values.

Ensemble methods have also been proposed to address class imbalance, such as the Balanced Random Forest (BRF) algorithm (Chen & Breiman, 2004). BRF combines random undersampling and bagging with a decision tree algorithm to improve classification performance. The main advantage of BRF is that it is a simple and effective method for handling imbalanced datasets. By undersampling the majority class, it reduces the bias towards the majority class that can occur in traditional random forest. However, BRF may still suffer from some of the limitations of random forest, such as being susceptible to noise and outliers in the data.

Deep learning-based methods have also been explored for dealing with imbalanced datasets, such as Imbalanced Adversarial Training with Reweightingand (IATwR) (Wang et al., 2021). This methods aims to address class imbalance in a dataset by utilizing adversarial training and reweighting techniques. The algorithm works by training a neural network with two objectives: to minimize the classification loss and to maximize the adversarial loss, which encourages the network to learn features that are discriminative for the minority class. One of the advantages of IATwR is that it can be applied to a wide range of deep learning architectures. Additionally, the adversarial loss helps the network to learn features that are less biased towards the majority class. However, IATwR requires large amounts of labeled data and can be computationally expensive, making it challenging to apply to certain domains.

One last algorithm that was discarded from being incorporated into this report was the Adaptive Sensitive Reweighting to Mitigate Bias in Fairness (ASR) (Krasanakis et al., 2018), this is a method designed to mitigate bias in machine learning models and it is based on the idea of reweighting the training data to decrease the impact of sensitive attributes such as race or gender. ASR is an adaptive method, which means that it uses the current model's predictions to update the weights in a way that promotes fairness. One of the main advantages of ASR is that it is compatible with several machine learning algorithms. On the other hand, determining the optimal weights for the sample requires solving a convex optimization problem, which can sometimes be unsolvable within the given bounds provided to the solver or may take too long to converge.

In summary, while there are many techniques and algorithms available for addressing the problem of biased datasets and fairness, each approach has its own strengths and limitations. The algorithms used in this report, namely SMOTEBoost, AdaFair, CUSBoost, and RAMOBoost, offer a good trade-off between fairness and accuracy, making them suitable for real-world applications.

3. Preliminaries.

Definition of Fairness. The definition of fairness we decided to strictly refer in our report is based on the AdaFair paper (Iosifidis & Ntoutsi, 2019) in exact same author words. Let us define a dataset D consisting of n samples drawn from a joint distribution $P(F, S, y)$, where S and F denotes sensitive (i.e. gender or race) and non-sensitive attributes, respectively, y is the class label. For simplicity, we consider that the classification problem is binary, that is, $y \in \{+1, -1\}$ and that there exists a single sensitive attribute S, also binary. That is, $S \in \{s, \overline{s}\}$ with s and $\overline{s}$ denoting the protected and non-protected group, respectively. We use the notation s_+, (s_-), $\overline{s}_+$, $\overline{s}_-$ to denote the protected and non- protected group for the positive (negative, respectively) class.

To define fairness from here, it is required to introduce the concept of Equalized Odds (Eq. Odds) which measures the difference in prediction errors between the protected and non-protected groups. Let δFPR (δFNR) be the difference in false positive rates (false negative rates, respectively) between the protected and non-protected group, defined as follows ($\hat{y}$ denotes the predicted label):

$$\delta FPR = P(y \neq \hat{y}|\overline{s}_-) - P(y \neq \hat{y}|s_-)$$
$$\delta FNR = P(y \neq \hat{y}|\overline{s}_+) - P(y \neq \hat{y}|s_+)$$

The goal is to minimize both differences, the so-called Eq.Odds:

$$\text{Eq.Odds} = |\delta FPR| + |\delta FNR|$$

Finally, the goal of fairness-aware classification is finding a mapping function $f(\cdot)$ that minimizes Eq.Odds discrimination while maintaining good predictive performance.

In order to measure the predictive ability of the models we used accuracy and balanced accuracy scores as well as the True Positive and True Negative Rates (TPR, TNR).

4. Algorithms and datasets

In this section we briefly introduce our selected approaches as well as the testing datasets chosen to perform our experiments.[1]

AdaFair This approach (Iosifidis & Ntoutsi, 2019) extends the instance weighting strategy of AdaBoost (Li & Jain, 2009) for each round, at each boosting round, the discriminated group is dynamically identified for each class and its effect on instance weighting is evaluated based on a cumulative notion of fairness. This enables achieving parity between the two groups while maintaining high true positive rates (TPRs) and true negative rates (TNRs) while minimizing the Eq. Odds metric.

Algorithm 1 AdaFair algorithm

Input: $D(x_i, y_i)^N, T, \epsilon$
Output: Ensemble H

1: Initialize $w_i = \frac{1}{N}$ and $u_i = 0$, for $i = 1, 2, ..., N$
2: **for** j = 1 to T: **do**
3: Train a classifier h_j to the training data using weight w_i
4: Compute the error rate $err_j = \frac{\sum_{i=1}^{N} w_i I(y_i \neq h_j(x_i))}{\sum_{i=1}^{N} w_i}$
5: Compute the weight $\alpha_i = \frac{1}{2} ln(\frac{1-err_j}{err_j})$
6: Compute fairness-related $\delta FNR^{1:j}$
7: Compute fairness-related $\delta FPR^{1:j}$
8: Compute fairness-related costs u_i
9: Compute fairness-related $\delta FNR^{1:j}$
10: Update the distribution as
$w_i \leftarrow \frac{1}{Z_j} w_i \cdot e^{\alpha_j h_j(x) I(y_i \neq h_j(x_i))}(1 + u_i)$
11: Output $H(x) = \sum_{j=1}^{T} \alpha_i h_j(x)$

In Algorithm 1 AdaFair, the whole structure of the algorithm is similar to AdaBoost with some crucial differences such as the instance weighting strategy. While AdaBoost assigns weights to instances based on their classification error rate, AdaFair extends this strategy to incorporate fairness constraints. Specifically, AdaFair assigns weights to instances based on their contribution to both classification accuracy and fairness, this can be seen in steps from 6 to 9, then the new parameters are updated based on the measure of this rate. This allows AdaFair to achieve parity between groups while maintaining high true positive rates and true negative rates.

[1]See the git repository: Boosting-for-Fairness-Aware-Classification

SMOTEBoost The SMOTEBoost algorithm (Chawla et al., 2002) integrate SMOTE - a technique of eliminating dataset imbalance in the boosting procedure by creating new synthetic examples from the minority class during each iteration.

SMOTE operates in the feature space to create synthetic instances of the minority class, which allows inductive learners to broaden their decision regions. SMOTE for classification creates the new synthetic minority samples as follows:

- Take majority vote between the feature vector under consideration and its k nearest neighbors for the nominal feature value. In the case of a tie, choose at random.
- Assign that value to the new synthetic minority class sample.

Using this technique, a new minority class sample is created in the neighborhood of the minority class sample under consideration. The neighbors are proportionately utilized depending upon the amount of SMOTE.

So, next, we can describe a SMOTEBoost algorithm. Boosting assigns equal weights to all misclassified examples, but if the training set is dominated by the majority class, subsequent samplings may remain skewed towards it. Boosting can reduce variance and bias in the final ensemble, but may not be as effective for skewed data sets.

To reduce bias from class imbalance in the learning procedure, SMOTE is introduced in each boosting round. This allows each base estimator to train on more of the minority class cases and broaden decision regions for the minority class. SMOTE is only applied to the minority class examples in the distribution D_t at iteration t, effectively increasing their sampling weights. Synthetically created minority class cases are discarded after learning a classifier at iteration t.

The combination of SMOTE and boosting is a variant of AdaBoost.M2, where a weak learning algorithm is presented with a different distribution D_t emphasizing particular training examples. The distribution is updated to give wrong classifications higher weights. Pseudocode for Algorithm 2. SMOTEBoost:

CUSBoost. CUSBoost is an algorithm that combines cluster-based undersampling and AdaBoost techniques to address imbalanced datasets. Unlike SMOTEBoost, which oversamples the minority class instances using SMOTE method, CUSBoost employs cluster-based undersampling from the majority class. To achieve this, CUSBoost first separates the majority and minority class instances from the original dataset and clusters the majority class instances into k clusters using k-means clustering algorithm. The hyperparameter k is optimized using hyperparameter optimization techniques. Then, random under-sampling is performed on

Algorithm 2 SMOTEBoost algorithm

Input: Set $S\{(x_1, y_1), ..., (x_m, y_m)\}$ $x_i \in X$, with labels $y_i \in Y = \{1, ..., C\}$, where C_p, $(C_p < C)$ corresponds to a minority (positive) class.

Output: Ensemble H

Let $B = \{(i, y) : i = 1, ..., m, y \neq y_i\}$
Initialize the distribution D_1 over the examples, such that $D_1(i) = \frac{1}{m}$.
for t = 1 to T: **do**
    Modify distribution D_t by creating N synthetic examples from minority class C_p using the SMOTE algorithm
    Compute the error rate $err_j = \frac{\sum_{i=1}^{N} w_i I(y_i \neq h_j(x_i))}{\sum_{i=1}^{N} w_i}$
    Train a weak learner using distribution D_t
    Compute weak hypothesis $h_t : X \times Y \rightarrow [0, 1]$
    Compute the pseudo-loss of hypothesis h_t:
    $\epsilon_t = \sum_{(i,y)\in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$
    Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ and $w_t = \frac{1}{2} \cdot (1 - h_t(x_i, y) + h_t(x_i, y_i))$
    Update $D_t : D_{t+1}(i, y) = (D_t(i, y)/Z_t) \cdot \beta_t^{w_t}$
    Update the distribution as
    $w_i \leftarrow \frac{1}{Z_j} w_i \cdot e^{\alpha_j h_j(x) I(y_i \neq h_j(x_i))}(1 + u_i)$
    where Z_t is a normalization constant
Output $h_{fn} = \arg\max_{y \in Y} \sum_{t=1}^{T} \log(\frac{1}{\beta_t}) h_t(x, y)$

each of the created clusters by randomly selecting an equal portion of the instances and removing the rest. The number of samples chosen for each cluster depends on the degree of imbalance in the original dataset. This algorithm works best when the dataset is highly clusterable. Finally, the representative samples from the clusters are combined with the minority class instances to obtain a balanced dataset.

The CUSBoost method involves using a set of decision trees generated through an algorithm to classify new instances. At the beginning of the process, all training instances are assigned equal weights, denoted by d_1 where d represents the total number of training instances. The weights of the instances are then adjusted based on their classification accuracy, with incorrectly classified instances having their weight increased and correctly classified instances having their weight decreased. The instance weight serves as an indicator of how challenging it is to classify a particular instance. To calculate the error rate of the model M_i, the method sums the weights of misclassified instances in the D_i sub-data set.

$$error(M_i) = \sum_{i=1}^{d} w_i \star err(x_i)$$

If an instance, x_i is misclassified, then $err(x_i)$ is one. Otherwise, $err(x_i)$ is zero.

In each iteration i, the weight of an instance x_i is multiplied by the error $\frac{error(M_i)}{1-error(M_i)}$ if it is correctly classified. Afterward, all the instance weights are normalized by multiplying each weight by the sum of old weights divided by the sum of new weights. This approach results in an increase in the weights of misclassified instances and a decrease in the weights of correctly classified instances. If the error rate of the model M_i exceeds 0.5, then M_i is abandoned, and a new M_i is derived by generating a new sub-dataset D_i. The entire process is outlined in Algorithm 3. CUSBoost:

Algorithm 3 CUSBoost algorithm

Input: Imbalanced data D, number of iterations k, decision tree induction algorithm

Output: Ensemble H

Initialize weights $x_i \in D$ to $\frac{1}{d}$.
for i = 1 to k: **do**
    Create balanced dataset D_i with distribution D using cluster-based under-sampling
    Derive a tree, M_i from D_i employing an input decision tree induction algorithm
    Compute the error rate of M_i, $error(M_i)$
    if $error(Mi) \geq 0.5$ **then**
        Go back to dataset creation step and try again
    for each $x_i \in D_i$ that correctly classified **do**
        Multiply weight of x_i by $(\frac{error(M_i}{1-error(M_i)})$
    Normalise the weight of each instances, x_i
Initialise weight of each class to 0
for i = 1 to k: **do**
    $w_i = \log \frac{1-error(M_i)}{error(M_i)}$
    $c = M_i(x_{New})$
    Add w_i to weight for class c
Return the class with largest weight

RAMOBoost The last algorithm we have implemented was RAMOBoost which enhances learning on imbalanced datasets through oversampling and an ensemble approach. Unlike SMOTE, which randomly and uniformly samples minority examples from an initial dataset, RAMOBoost assesses the potential value of each minority example and determines their sampling weights based on this evaluation. The method calculates the distance of each individual minority example from the set of nearest neighbors to determine their sampling weight.

RAMOBoost has a twofold objective: to reduce biases from imbalanced data and adaptively learn information from data distribution. To achieve this, it employs an adaptive weight adjustment procedure that shifts the decision boundary towards difficult-to-learn examples and a ranked sampling probability distribution to generate synthetic minority instances.

RAMOBoost follows the classic AdaBoost.M2 procedure

to apply the boosting process on the misclassified dataset B. For the n-class classification problem, B is defined to be the set where each example in D is replicated $n-1$ times with a different class label other than the true one. $\{\hat{r}_i\}$ serves as the i distribution function determining the probability that examples in B are chosen for generating the synthetic instances, it is calculated by mapping the number of majority cases in the k_1-nearest neighbors of each example in B into the range $(0, 1)$.

Overall, RAMOBoost has two mechanisms to improve learning from imbalanced data. The first generates more synthetic instances for minority class examples based on their distributions. The second mechanism uses the pseudo-loss of the current hypothesis to update the sampling distribution and shift the final hypothesis toward the decision boundary. The complete RAMOBoost algorithm was moved to Appendix C due to readability preferences, it is described in Algorithm C.

5. Evaluation

The main goal of our experiments is to evaluate if to replicate the results introduced at the original AdaFair paper (Iosifidis & Ntoutsi, 2019) and to provide a perspective of the predictive accuracy-increment of fairness trade-off for all the depicted algorithms (AdaFair, SMOTEBoost, CUSBoost, RAMOBoost). Accuracy and balanced accuracy would address how good an algorithm classifies, whereas fairness would be reported by aggregated measure Eq.Odds and also by TPR and TNR values for both protected and non-protected groups.

5.1. Experimental setup

5.1.1. Datasets.

Four real-world datasets were used for evaluation, most of them with serious variation not only in classes imbalance (class ratio varying from 1:1.12 to 1:15), but also in cardinality and dimensionality. Consequently, they are suitable considering our experimental setup. The datasets used are Adult Census [2], Bank [3], Compass [4] and KDD [5], note that the shorten links provided lead to the original datasets, we encourage to use the datasets in our Github repository as well as our code to reproduce the results in this report. Link to Github repository: Boosting-for-Fairness-Aware-Classification. Table 1 illustrates the main features of these sets.

[2] https://tinyurl.com/4ruwvtz4
[3] https://tinyurl.com/5b8dhr5y
[4] https://tinyurl.com/59cejx9s
[5] https://tinyurl.com/2xwyd56e

5.1.2. Preprocessing details.

To ensure reproducibility with respect to the original AdaFair paper we implemented the same preprocessing methods, each dataset was treated separately, details can be found on Table 2.

Table 1. An overview of the datasets.

	Adult C.	Bank	Compass	KDD
Instances	45,175	40,004	5,278	299,285
Atributes	14	16	9	41
Sen.Attr.	Gender	M. Status	Gender	Gender
Class ratio	1:3.03	1:7.57	1:1.12	1:15.11
Pos. class	>50K	Subs	recidivism	>50K

Table 2. An overview of the datasets.

	Adult C.	Bank	Compass	KDD
One Hot enc.	√	√	√	×
Ordinal enc.	×	×	×	√
Standard	√	√	√	√
Target	income	y	two_year_r	income
NaN values	?	unknown	N/A	N/A

Standarization was done using the StandardScaler provided by sklearn package, in the case of One Hot encoding and Ordinal encoding, they refer to the transformation of some columns with categorical features, only one of the techniques was used per dataset, One Hot implementation was done from scratch while the other uses the Ordinal Encoder from sklearn. Sensitive columns (gender) were masked as True/False to be used for the models. In the case of Compas only the following columns were kept: industry code, occupation code and own business or self employed.

None of the datasets was treated for outliers and all rows with NaN values were dropped, each dataset had a specific value for NaNs, this is specified in the previous table. The two_year_r refers to two_year_recid, shortened for convenience.

5.1.3. Training details.

In all the cases, the dataset was split into 80% to be used for training and 20% into testing, using suffling and random state set in 42.

The experiments were conducted using a Repeated Stratified KFold from sklearn with 2 splits and 5 repeats with random state equal to 42.

All metrics are provided in the Github repository and were explained in previous sections of this report.

All the experiments were conducted using a laptop with an Intel Core i5 CPU (@2.50GHz) with 8GB of RAM, but it can also run using Colab.

A total of seventeen runs were performed and the hyperparameters were tuned mainly based on reproducibility of the original AdaFair paper, similar specifications were used. In the case of CUSBoost, the number of clusters was fixed as well as assuming the number of samples from each cluster to the be the half of the total amount, we tried to change this parameter but no big improvement was reported.

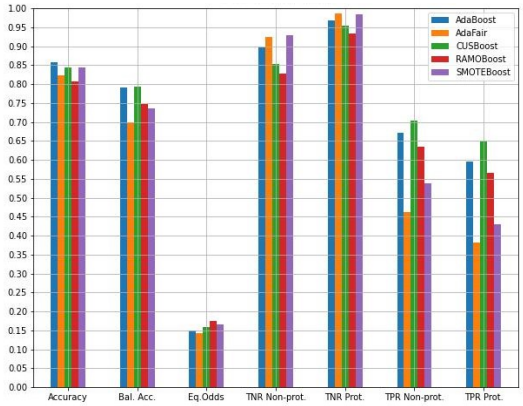

Figure 1. Adult census: Predictive and fairness metrics for imbalanced value of annual income of a person exceeds 50K dollars for demographic data from the U.S.. The sensitive attribute is S = Gender with s = female being the protected group; the positive class is people receiving more than 50K.

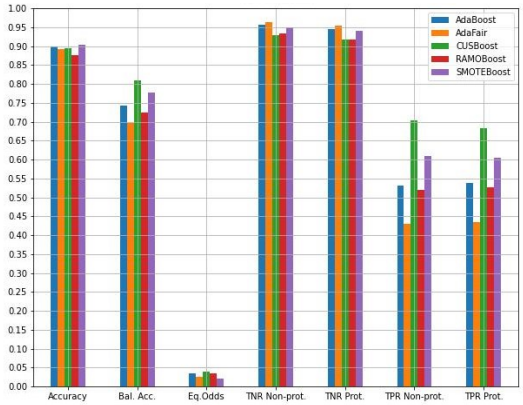

Figure 2. Predictive and fairness metrics for imbalanced dummy variable of determining if a person subscribes to the product (bank term deposit) in direct marketing campaigns of a Portuguese banking institution. As positive class we consider people who subscribed to a term deposit. We consider as S = maritial status with s = married being the protected group.

5.1.4. Parameter selection and evaluation.

AdaBoost, SMOTEBoost and AdaFair require a base classifier, in the three cases, the one chosen was a Decision Tree Classifier of depths 2, 5 and 2, using 200, 500, 500 estimators, respectively.

CUSBoost used a default of 23 clusters and picking a half of the data from each cluster, the number of estimators was 10 with depth 5.

RAMOBoost utilized a Decision Tree Classifier to feed its internal AdaBoost classifier, this tree has 50 estimators with default depth.

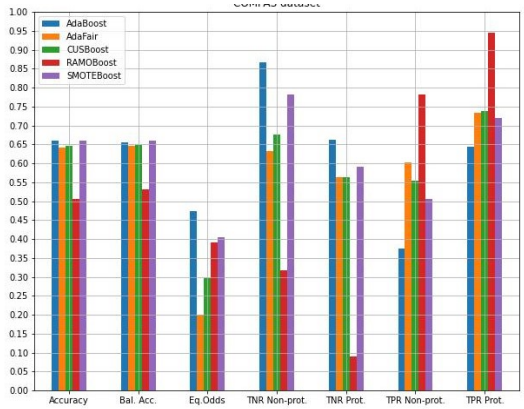

Figure 3. Predictive and fairness metrics for imbalanced dummy variable: if a person will be re-arrested within two years (recidivism). Data contains information on prisoners in Broward County such as the number of juvenile felonies. We consider recidivism as the positive class and S = Gender with s = female as the protected group.

5.2. Predictive and fairness performance

5.2.1. Adult census income.

In Figure 5.1.3, we show the performance of the different approaches on Adult census dataset. Regarding predictive performance, the best balanced accuracy is achieved by CUSBoost and AdaBoost (0.79); RAMOBoost, SMOTEBoost and AdaFair have a drop of 5 %, 7 % and 11.7 % respectively in their balanced accuracy.

Regarding Eq.Odds, AdaFair showed the best result (0.142), followed by ADABoost (0.149). Surprisingly, the last one is more concerned about the minority class compared to the other, and this is outperformed only by CUSBoost with the highest values of TPRs for both, protected (0.649) and non-protected groups (0.704).

5.2.2. Bank.

The results are shown in Figure 5.1.3. All approaches achieved low Eq.Odds - less than 0.039. A closer look at Eq.Odds and namely at TPRs shows that CUSBoost

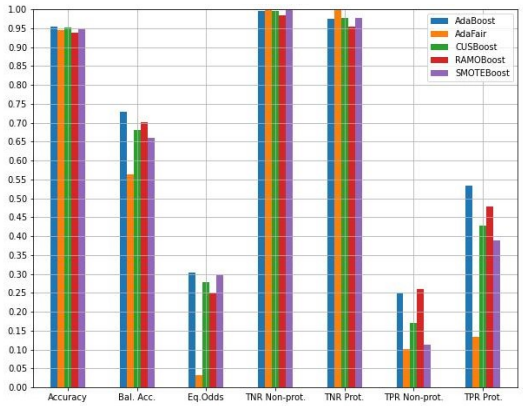

Figure 4. Predictive and fairness metrics for imbalanced dummy variable: positive class - people receiving more than 50K annually. The class labels were drawn from the total person income field rather than the adjusted gross income. We consider S = Gender with s = female as the protected group.

surpasses other boosting algorithms, followed by SMOTEBoost with 11 % drop for protected group and 13 % drop for non-protected group. For TNRs the difference is "insignificant". In the prediction performance CUSBoost dominates others with balanced accuracy (0.809). The second is SMOTEBoost (0.777) and the last one is AdaFair (0.696).

5.2.3. COMPASS.

The results are shown in Figure 5.1.3. Regarding balanced accuracy, SMOTEBoost performs best (0.661) and RAMOBoost worse (0.531), as it is biased to positive class (TPR NON-PROT. is 0.781, TPR PROT is 0.945, but TNR NON-PROT. is 0.318 TNR PROT. is 0.090). AdaFair again outoperforms others in fairness with Eq.Odds equles 0.199. CUSBoost, RAMOBoost and SMOTEBoost have an increase of Eq.Odds equals to 49 %, 96 % and 103 % respectively.

5.2.4. KDD CENSUS INCOME.

Figure 4. The dataset suffers from extremely high class imbalance, with a ratio of 1:15 (c.f., Table 1). In terms of balanced accuracy, AdaBoost benchmark performance is the best (0.729) and followed by RAMOBoost (0.701), CUSBoost (0.659) and SMOTEBoost (0.659). The worst in accuracy of prediction is AdaFair (0.563). For TNR NON-PROT. and TNR PROT. rates all algorithms showed the results higher 0.95. However, TPRs demonstrates that AdaBoost surpasses other boosting algorithms, followed by RAMOBoost with 10 % drop for protected group, whereas for non-protected group AdaBoost (0.249) and RAMOBoost (0.26) are approximately the same.

6. Conclusions

In this work, we have proposed four different techniques based on AdaBoost to boost for fairness-aware classification, namely Adafair, SMOTEBoost, CUSBoost, and RAMOBoost. These techniques aim to address the problem of class imbalance in machine learning datasets while ensuring fairness in the classification results.

Simulation results on four real-world datasets across various assessment metrics, including accuracy, balanced accuracy, equalized odds, TNR non-prot., TNR prot., TPR non-prot., and TPR prot., demonstrate the difference between these proposed approaches. Achieving fairness often comes at a cost to accuracy. AdaFair is a good choice for those who are specifically interested in fairness, as it consistently achieves the lowest equalized odds score across all datasets, while SMOTEBoost, CUSBoost, and RAMOBoost are more focused on predicting classes accurately. Based on the high performance in balanced accuracy across our datasets, CUSBoost appears to be the best algorithm for prediction in imbalanced scenarios.

Overall, these techniques are useful tools for ensuring fairness in classification tasks and addressing the problem of class imbalance. It is important for data scientists to consider the specific goals and priorities of the task at hand when selecting an appropriate algorithm for a particular application.

The current report evaluates several proposed techniques that use boosting as the basis of their algorithms. There is a wide variety of future paths in this direction such as the use of neural networks, indeed, under current developments of generative models, a fair and balanced representation of sensitive groups is crucial.

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. Smote: Synthetic minority over-sampling technique. 16(1):321–357, jun 2002. ISSN 1076-9757.

Chen, C. and Breiman, L. Using random forest to learn imbalanced data. *University of California, Berkeley*, 01 2004.

Elhassan, T., M, A., F, A.-M., and Shoukri, M. Classification of imbalance data using tomek link (t-link) combined with random under-sampling (rus) as a data reduction method. *Global Journal of Technology and Optimization*, 01, 01 2016. doi: 10.4172/2229-8711.S1111.

Iosifidis, V. and Ntoutsi, E. Adafair: Cumulative fairness adaptive boosting. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, pp. 781–790, New

York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369763. doi: 10.1145/3357384.3357974. URL https://doi.org/10.1145/3357384.3357974.

Krasanakis, E., Spyromitros-Xioufis, E., Papadopoulos, S., and Kompatsiaris, Y. Adaptive sensitive reweighting to mitigate bias in fairness-aware classification. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pp. 853–862, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186133. URL https://doi.org/10.1145/3178876.3186133.

Li, S. Z. and Jain, A. (eds.). *AdaBoost*, pp. 9–9. Springer US, Boston, MA, 2009. ISBN 978-0-387-73003-5. doi: 10.1007/978-0-387-73003-5_825. URL https://doi.org/10.1007/978-0-387-73003-5_825.

Moreo, A., Esuli, A., and Sebastiani, F. Distributional random oversampling for imbalanced text classification. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pp. 805–808, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340694. doi: 10.1145/2911451.2914722. URL https://doi.org/10.1145/2911451.2914722.

Thai-Nghe, N., Gantner, Z., and Schmidt-Thieme, L. Cost-sensitive learning methods for imbalanced data. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2010. doi: 10.1109/IJCNN.2010.5596486.

Wang, W., Xu, H., Liu, X., Li, Y., Thuraisingham, B., and Tang, J. Imbalanced adversarial training with reweighting. *CoRR*, abs/2107.13639, 2021. URL https://arxiv.org/abs/2107.13639.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Julia Kudryavtseva (20% of work)

- Coding the SMOTEBoost algorithm
- Preparing the GitHub Repo
- Preparing the Section Evaluation of this report
- Preparing the presentation

Xavier Aramayo Carrasco (20% of work)

- Coding the CUSBoost algorithm, preprocessing and pipeline of the notebook.
- Experimenting with model parameters on datasets
- Preparing the Section Related work and References of this report
- Preparing the GitHub Repo
- Preparing the presentation

Vo Ngoc Bich Uyen (20% of work)

- Reviewing literate on the topic (6 papers)
- Coding the AdaFair algorithm
- Preparing the GitHub Repo
- Preparing the Sections Conclusions of this report
- Preparing the presentation

Alisa Kalacheva (20% of work)

- Reviewing literate on the topic (3 papers)
- Coding the RAMOBoost algorithm
- Preparing the GitHub Repo
- Preparing the Section Preliminaries and Introduction of this report
- Preparing the presentation

Alexander Lepinskikh (20% of work)

- Preparing the presentation
- Preparing the GitHub Repo
- Preparing algorithms' pseudocodes and their descriptions in this report.

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: RAMOBoost and CUSBoost algorithms were replicated on 40 % from an open-source code.

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   ☐ Yes.
   ☐ No.
   ☑ Not applicable.

   Students' comment: We didn't perform data collection, we used existing datasets, the links are provided in the report and they are also available in the Github repository.

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

12. Clearly defined error bars are included in the report.

    ☑ Yes.

- ☐ No.
- ☐ Not applicable.

Students' comment: None

13. A description of the computing infrastructure used is included in the report.

- ☑ Yes.
- ☐ No.
- ☐ Not applicable.

Students' comment: None

C. RAMOBoost algorithm.

Algorithm 4 RAMOBoost algorithm

Input:

1) Training dataset with m class examples $\langle (x_1, y_1), ..., (x_m, y_m) \rangle$, where $x_i (i = 1, ..., m)$ is an instance of the n dimensional feature space X and $y_i \in Y = major, minor$ is the class identity label associated with instance xi
2) Number of synthetic data samples to be generated at each iteration N
3) Number of iterations; i.e., the number of base classifiers T
4) Number of nearest neighbors in adjusting the sampling probability of the minority examples k_1
5) Number of nearest neighbors used to generate the synthetic data instances k_2
6) The scaling coefficient α

Output: Ensemble H

Let $B = \{(i, y) : i = 1, ..., m, y \neq y_i\}$

Initialize the distribution D_1 over the examples, such that $D_1(i) = \frac{1}{|B|}$ for $(i, y) \in B$ (for two class problems, $|B| = m$).

for t = 1 to T: **do**

Sample the mislabeled training data with D_t and get back the sampled dataset S_o of identical size. Slice S_o into the majority subset e_1 and the minority subset e_2 of size m_{lt} and m_{ot} , respectively.

For each example $x_i \in e_2$, find its k_1 nearest neighbors in the dataset S_o according to the Euclidean distance in n-dimensional space and calculate r_i defined as:

$$r_i = \frac{1}{1-exp(-\alpha \cdot \delta_i)}, i = 1, 2, ...m_o t$$

where δ_i is the number of majority cases in k_1 examples.

Normalize r_i according to:

$$\hat{r_i} = \frac{r_i}{\sum_{i=1}^{m_{ot}} r_i}$$

such that $\{\hat{r_i}\}$ is a distribution function $\sum_{i=1}^{m_{ot}} r_i = 1$. Define $d_t = \{\hat{r_i}\}$

Sample e_2 with d_t and get back a sampling minority dataset g_t, of size m_{ot}.

For each example $x_i \in g_t$, find its k_2 nearest neighbors in e_2 according to the Euclidean distance in n dimensional space and use linear interpolation to generate N synthetic data samples.

Provide the base classifier with sampling dataset S_o and the N synthetic data samples.

Get back a hypothesis $h_t : X \times Y \rightarrow [0, 1]$

Compute the pseudo-loss of hypothesis h_t:

$$\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$$

Set $\beta_t = \epsilon_t / 1 - \epsilon_t$

Update $D_t : D_{t+1}(i, y) = (D_t(i, y)/Z_t) \cdot \beta_t^{(1+h_t(x_i, y_i) - h_t(x_i, y))}$

where Z_t is a normalization constant

Output hypothesis $h_{final}(x)$ is calculated as follow:

$$h_{final}(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} \log(\frac{1}{\beta_t}) h_t(x, y)$$

D. Experiments results

Table 3. Mean of metrices on K-folds on the dataset Adult census.

	ACCURACY	BAL. ACC.	EQ.ODDS	TNR NON-PROT.	TNR PROT.	TPR NON-PROT.	TPR PROT.
ADABOOST	0.858	0.792	0.149	0.896	0.968	0.672	0.595
ADAFAIR	0.824	0.699	0.142	0.924	0.986	0.462	0.382
CUSBOOST	0.843	0.794	0.158	0.852	0.955	0.704	0.649
RAMOBOOST	0.808	0.747	0.176	0.828	0.934	0.636	0.566
SMOTEBOOST	0.843	0.736	0.166	0.928	0.985	0.539	0.429

Table 4. Std of metrices on K-folds on the dataset Adult census.

	ACCURACY	BAL. ACC.	EQ.ODDS	TNR NON-PROT.	TNR PROT.	TPR NON-PROT.	TPR PROT.
ADABOOST	0.003974	0.006354	0.027291	0.008255	0.006426	0.014959	0.033746
ADAFAIR	0.002912	0.007892	0.031542	0.012244	0.004662	0.023744	0.039454
CUSBOOST	0.007036	0.017277	0.092870	0.037535	0.009014	0.062414	0.023912
RAMOBOOST	0.005168	0.009379	0.055566	0.009133	0.007169	0.018318	0.050753
SMOTEBOOST	0.009956	0.021202	0.053835	0.012968	0.007575	0.050689	0.047632

Table 5. Mean of metrices on K-folds on the dataset Bank.

	ACCURACY	BAL. ACC.	EQ.ODDS	TNR NON-PROT.	TNR PROT.	TPR NON-PROT.	TPR PROT.
ADABOOST	0.899	0.743	0.035	0.957	0.945	0.531	0.538
ADAFAIR	0.893	0.696	0.024	0.964	0.954	0.431	0.434
CUSBOOST	0.895	0.809	0.039	0.929	0.918	0.704	0.682
RAMOBOOST	0.876	0.725	0.034	0.933	0.918	0.520	0.526
SMOTEBOOST	0.904	0.777	0.020	0.950	0.941	0.610	0.606

Table 6. Std of metrices on K-folds on the dataset Bank census.

	ACCURACY	BAL. ACC.	EQ.ODDS	TNR NON-PROT.	TNR PROT.	TPR NON-PROT.	TPR PROT.
ADABOOST	0.002453	0.008502	0.031983	0.004444	0.004715	0.032537	0.029814
ADAFAIR	0.004849	0.010706	0.014987	0.003973	0.004172	0.026009	0.024161
CUSBOOST	0.004739	0.023219	0.011980	0.009883	0.012135	0.057105	0.056773
RAMOBOOST	0.005133	0.014326	0.023450	0.006308	0.009773	0.042699	0.033199
SMOTEBOOST	0.004777	0.022559	0.021119	0.008295	0.010435	0.054455	0.053282

Table 7. Mean of metrices on K-folds on the dataset KDD.

	ACCURACY	BAL. ACC.	EQ.ODDS	TNR NON-PROT.	TNR PROT.	TPR NON-PROT.	TPR PROT.
ADABOOST	0.955	0.729	0.305	0.997	0.976	0.249	0.533
ADAFAIR	0.944	0.563	0.033	0.999	0.998	0.101	0.134
CUSBOOST	0.951	0.680	0.279	0.997	0.977	0.169	0.428
RAMOBOOST	0.937	0.701	0.249	0.985	0.955	0.260	0.479
SMOTEBOOST	0.948	0.659	0.296	0.999	0.978	0.113	0.389

Table 8. Std of metrices on K-folds on the datasets KDD.

	ACCURACY	BAL. ACC.	EQ.ODDS	TNR NON-PROT.	TNR PROT.	TPR NON-PROT.	TPR PROT.
ADABOOST	0.000822	0.005479	0.024866	0.001032	0.002354	0.021986	0.013412
ADAFAIR	0.000375	0.003989	0.009245	0.000366	0.000211	0.008138	0.009149
CUSBOOST	0.000964	0.012362	0.053205	0.001399	0.005301	0.031412	0.034037
RAMOBOOST	0.021482	0.037678	0.039086	0.019286	0.041853	0.129546	0.099375
SMOTEBOOST	0.001556	0.023035	0.071280	0.000747	0.010427	0.011051	0.064318

Table 9. Mean of metrices on K-folds on the datasets Compass.

	ACCURACY	BAL. ACC.	EQ.ODDS	TNR NON-PROT.	TNR PROT.	TPR NON-PROT.	TPR PROT.
ADABOOST	0.659	0.656	0.474	0.866	0.661	0.375	0.644
ADAFAIR	0.642	0.647	0.199	0.632	0.564	0.602	0.733
CUSBOOST	0.646	0.650	0.297	0.676	0.562	0.555	0.738
RAMOBOOST	0.505	0.531	0.391	0.318	0.090	0.781	0.945
SMOTEBOOST	0.659	0.661	0.405	0.781	0.590	0.505	0.719

Table 10. Std of metrices on K-folds on the datasets Compass.

	ACCURACY	BAL. ACC.	EQ.ODDS	TNR NON-PROT.	TNR PROT.	TPR NON-PROT.	TPR PROT.
ADABOOST	0.010302	0.010793	0.222027	0.058628	0.050842	0.097383	0.052013
ADAFAIR	0.011731	0.017878	0.082693	0.127953	0.105001	0.143385	0.139381
CUSBOOST	0.017209	0.016103	0.260876	0.114799	0.058385	0.105874	0.030313
RAMOBOOST	0.019066	0.017426	0.218807	0.113954	0.053980	0.079110	0.014915
SMOTEBOOST	0.022588	0.021049	0.148442	0.104923	0.068870	0.115152	0.059591