Achieving Fairness without Demographic Information

Alessandro Pasi, Matteo Belletti, Razvan Ciprian Stricescu

June 26, 2024

Abstract

In much of the existing machine learning (ML) fairness research, protected features like race and sex are typically included in datasets and used to address fairness concerns. However, due to privacy concerns and regulatory restrictions, collecting or using these features for training or inference is often not feasible. This raises the question: how can we train an ML model to be fair without knowing the protected group memberships? This work tackles this issue by proposing Adversarially Reweighted Learning (ARL). The key idea is that non-protected features and task labels can help identify fairness problems. ARL uses these features to co-train an adversarial reweighting approach that enhances fairness. The results indicate that ARL improves Rawlsian Max-Min fairness and achieves significant AUC improvements for the worst-case protected groups across various datasets, outperforming current leading methods.

1 Introduction

This project aims to enhance fairness in machine learning (ML) without relying on protected demographic features. In order to implement the code we studied the Fairness without Demographics through Adversarially Reweighted [3] paper by Preethi Lahoti, and we adopted the code implementation by J. Mohazzab, L.R. Weytingh, C.A. Wortmann and B. Brocades Zaalberg [4].

Learning as ML systems are increasingly deployed in critical decision-making contexts, ensuring they do not discriminate is crucial. Recent studies have highlighted significant accuracy disparities across demographic groups in applications such as face detection, healthcare, and recommendation systems. In response, extensive research has focused on defining fairness and developing methods to reduce bias. However, these efforts often assume that the model has access to protected attributes like race or gender during training, if not during inference.

In reality, collecting or using protected features is often impractical due to privacy concerns, legal constraints, or regulatory restrictions. Despite these limitations, it remains essential for ML systems to be fair. Surveys of ML practitioners from both the public sector and industry emphasize the challenge of achieving fairness without demographics, identifying it as a critical problem.

To address this challenge, we investigate Adversarially Reweighted Learning (ARL), a technique that uses non-protected features and task labels to identify and correct fairness issues. By employing an adversarial reweighting mechanism, ARL aims to improve performance for the worst-off protected groups.

We build on the Rawlsian principle of Max-Min welfare [1], which seeks to maximize the minimum expected utility across protected groups, even when their membership is unknown. This approach differs from parity-based fairness, which aims to minimize disparities across groups, and is particularly suited to high-stakes applications like healthcare and face recognition, where improving the worst-off groups is paramount.

The hypothesis is that protected groups correlate with observable features and class labels. For example, race might correlate with zip code, and class labels might be imbalanced across groups. These correlations, often seen as problematic, can be harnessed to enhance fairness metrics.

We also conduct detailed analyses to understand ARL's mechanisms, examining learned example weights and assessing robustness to worst-case training distributions. Our results show that ARL achieves high AUC for worst-case protected groups and overall robustness against training data biases.

2 Background and Motivation

2.1 Fairness in Machine Learning:

Recent research in machine learning has focused extensively on addressing fairness concerns. Various notions of fairness have been proposed, which can be categorized into:

1. individual fairness, ensuring similar treatment for similar individuals;
2. group fairness, aiming for statistical parity across different demographic groups;
3. improving per-group performance, such as Rawlsian Max-Min fairness for exemple.

This work adopts the third approach, aiming to enhance the performance for each group.

Incorporating these fairness concepts into ML models has involved approaches such as improving representation learning, integrating fairness constraints into learning objectives, post-processing decisions, and adversarial learning. These methods generally require knowledge of protected attributes to directly optimize fairness metrics. However, in many real-world applications, such attribute information may be unavailable or sparse.

2.2 Fairness Without Demographics:

To address the challenge of missing demographic information, some works use proxy features or assume slight perturbations in attributes, though these methods can introduce estimation bias.

Other approaches involve transfer learning to handle limited demographic data. For instance, domain adaptation techniques have been used where group labels are known for either source or target datasets, or models are trained to optimize for worst-case distributions over clients in federated learning settings. Additionally, some methods rely on trusted third parties to securely handle protected data using secure multi-party computation or differential privacy.

2.3 Computational-Identifiability:

Similar to some intersectional fairness works, this approach optimizes for fairness across computationally identifiable groups within the input space. Unlike those works, which assume available protected features and aim to minimize utility gaps across groups via regularization, this approach does not assume the availability of protected features.

2.4 Modeling Technique Inspirations:

The Adversarially Reweighted Learning (ARL) method draws inspiration from various modeling techniques. Re-weighting strategies address class imbalance by upweighting minority class examples. Focal loss directs learning algorithms to focus on more difficult examples by upweighting them based on their losses. Domain adaptation ensures model robustness and generalizability across different domains.

3 Problem Description

3.1 Binary classification scenario

We have a training dataset comprising n individuals, denoted as $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \sim \mathcal{X}$ represents an m-dimensional input vector of non-protected features, and $y_i \sim \mathcal{Y}$ indicates a binary class label. We assume the presence of K protected groups such that for each example x_i, there is an unobserved $s_i \sim \mathcal{S}$, where $\mathcal{S}$ is a random variable over the set $\{k\}_{k=0}^{K-1}$. The set of examples belonging to group s is represented by $\mathcal{D}_s := \{(x_i, y_i) : s_i = s\}_{i=1}^{n}$. Specifically, we assume protected groups $\mathcal{S}$ are unobserved attributes unavailable during training or inference. Nonetheless, evaluations are presented in terms of these groups.

3.2 Problem Definition

Given a dataset $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$, with no observed protected group memberships $\mathcal{S}$, such as race or gender, the objective is to learn a model $h_\theta : \mathcal{X} \to \mathcal{Y}$ that is fair to the groups in $\mathcal{S}$.

Similar to DRO [2], it adopts the Rawlsian Max-Min fairness principle of distributive justice [1]: the goal is to maximize the minimum utility U that a model achieves across all groups $s \in \mathcal{S}$, as defined in the following Definition. It assumes that correct predictions by the model increase the utility for the respective examples, hence, U can be any standard accuracy metric used in machine learning models.

3.3 Definition (Rawlsian Max-Min Fairness)

Let H be a set of hypotheses, and let $U_{\mathcal{D}_s}(h)$ denote the expected utility of hypothesis h for individuals in group s. A hypothesis h^* satisfies the Rawlsian Max-Min fairness principle if it maximizes the utility of the least advantaged group, i.e., the group with the lowest utility.

$$h^* = \arg\max_{h \in H} \min_{s \in \mathcal{S}} U_{\mathcal{D}_s}(h)$$

For evaluation (Section 6), we utilize AUC as the utility metric and report the minimum utility across protected groups $\mathcal{S}$ as AUC(min).

4 Implementation and algorithms

Before presenting the actual algorithms used in the Adversarially Reweighted Learning (ARL) framework, it is essential to explain some key concepts that form the foundation of this approach. This section provides a detailed overview of these concepts, which include the adversarial network and the main model's loss functions. Understanding these components will clarify the subsequent algorithmic implementation.

4.1 Adversarial Network for Weighting

The adversarial network (AdversaryNetwork) aims to identify regions of the input space where the model makes significant errors. This network helps in assigning weights to different training examples based on their potential to reduce errors in these regions. It's objective is to learn to weight samples such that the other model focuses more on examples from error-prone regions.

- **Structure**: Typically a neural network with input features (non-protected attributes) and output probabilities or weights. In this case the architecture is formed by three dense layers followed by ReLU activations.

- **Loss calculation**: The adversarial loss encourages the model to improve its performance on the regions identified by the adversary as having higher errors. This adversarial loss is calculated as a negative mean of the adversary's output multiplied by the main model's loss. It can be defined as:

$$J(\phi) = -E_{(x,y)\sim\mathcal{D}}[f_\phi(x) \cdot L(y, h_\theta(x))] \tag{1}$$

  where:

  - $f_\phi(x)$ is the weight assigned by the adversary to the input x.
  - $L(y, h_\theta(x))$ is the loss function evaluating the error between the true value y and the prediction $h_\theta(x)$.
  - The expectation is taken over the data distribution $\mathcal{D}$.

  The actual implementation is done as such:

```
adv_loss = -torch.mean(adversary(x) * loss.detach())
```

  Here's a breakdown of this expression:

  - `adversary(x)`: Computes the weights for each sample x using the adversarial network.
  - `loss.detach()`: Represents the error between the main model's predictions and the true target values, detached from the computation graph.
  - `torch.mean(...)`: Computes the mean of the weighted losses over the batch.
  - `-` (Negative sign): Converts the maximization problem into a minimization problem.

4.2 Learner Model

In the ARL framework, the learner model (h_θ) is responsible for making predictions based on input features. The goal of the main model is to minimize prediction errors while addressing fairness by focusing on underrepresented or challenging samples as identified by the adversarial network (f_ϕ).

- **Structure**: The architecture of the main model can be chosen based on the specific task (e.g., regression, classification). In this case it is quite similar to the Adversarial Network, as it is a series of dense layers followed by ReLU activations, with the main difference being it also uses a sigmoid layer as it provides an output in the [0, 1] range, which is what we desire in a binary classifier.

- **Loss calculation**: The main model seeks to minimize a weighted loss function, where the weights are provided by the adversarial network. The objective can be formulated as:

$$\min_\theta E_{(x,y)\sim\mathcal{D}}[w(x) \cdot L(y, h_\theta(x))] \tag{2}$$

  where:

  - $h_\theta(x)$ is the main model's prediction for input x.
  - $L(y, h_\theta(x))$ is the loss function measuring the error between the true target y and the prediction $h_\theta(x)$.
  - $w(x)$ is the weight assigned by the adversarial network to the input x.

– The expectation is taken over the data distribution $\mathcal{D}$.

For binary classification tasks, the base loss function can be the Binary Cross-Entropy Loss:

$$L_{\text{BCE}}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{3}$$

where:

– y_i is the true label (0 or 1).

– $\hat{y}_i = \sigma(h_\theta(x_i))$ is the predicted probability from the sigmoid output.

4.3 Training

The main model is trained iteratively, in conjunction with the adversarial network, using the weighted loss function.

1. **Initialize Weights**: The weights of the main model (θ) are initialized randomly.
2. **Forward Pass**:
   - Compute predictions $\hat{y}$ from input x.
   - Compute the loss $L(y, h_\theta(x))$.
3. **Loss Calculation**:
   - Use weights $w(x)$ from the adversary to compute the weighted loss.
4. **Backward Pass**:
   - Compute gradients of the weighted loss with respect to the model parameters θ.
5. **Weight Update**:
   - Update θ using an optimizer (e.g., gradient descent).

5 Key classes

We present some key classes to the implementation, that have not been explained yet as they are not model classes. Following in order the classes handling the dataset, tuning parameters and measuring fairness.

5.1 loadDataset

The `loadDataset` class is designed to facilitate the loading and preprocessing of various datasets (e.g., COMPAS, Adult, LSAC, and students-dataset) for use in PyTorch models. This class handles the complexities of reading data, normalizing features, binarizing target variables and protected attributes, and preparing the data for efficient loading during model training and evaluation.

The key functionalities of the `loadDataset` class are as follows:

1. **Data Loading and Preparation:** The constructor (`__init__`) initializes the dataset by loading the data and dataset statistics, such as feature names, target column name, and sensitive attributes. It ensures the data is in the correct format and ready for processing.

2. **Data Normalization:** The `normalize()` method standardizes numerical data by subtracting the mean and dividing by the standard deviation, ensuring zero mean and unit variance for each feature. This is essential for improving model convergence and performance.

3. **Binarization:** The `binarize()` method converts the target variable and sensitive attributes into binary format. This step is crucial for fair machine learning tasks where protected attributes are considered.

4. **Embedding Calculation:** If an embedding size is provided, the `calculate_embedding()` method computes the embedding sizes for categorical features. This allows the use of embeddings for categorical data in neural network models, which can capture complex relationships between categories.

5. **Subgroup Extraction:** The `set_subgroups()` method identifies all protected subgroups within the dataset. It generates combinations of protected features and determines the minority subgroup, which is important for analyzing fairness and bias in model predictions.

   Overall, the `loadDataset` class streamlines the process of preparing datasets for machine learning tasks, ensuring that data is appropriately preprocessed and ready for model training and evaluation.

5.2 hyperparameter.py

The `optimize_parameters` function is the core of hyperparameter.py, designed to identify the optimal hyperparameters for a machine learning model by performing an exhaustive grid search. The function systematically evaluates combinations of hyperparameters, such as batch size and learning rates, using cross-validation and selects the combination that yields the highest performance metric, specifically the Area Under the Curve (AUC).

The key functionalities of the `optimize_parameters` function are as follows:

1. **Hyperparameter Grid Definition:** The function takes a dictionary of hyperparameters and their possible values. It then creates all possible combinations of these hyperparameters using the `itertools.product` function, facilitating an exhaustive grid search.

2. **Data Loading and Preparation:** The function loads the training dataset using the `loadDataset` class. This includes reading the data, normalizing features, binarizing target variables, and preparing the data for model training.

3. **Cross-Validation:** The function employs K-fold cross-validation (with `KFold` from `sklearn.model_selection`) to ensure robust evaluation of hyperparameters. Each combination of hyperparameters is evaluated across multiple splits of the training data, providing a reliable estimate of model performance.

4. **Model Training and Evaluation:** For each combination of hyperparameters, the function trains the model using the `train_for_n_iters` function. It records the performance metrics (AUC) for each fold and computes the mean AUC across folds to determine the effectiveness of the hyperparameters.

5. **Optimal Hyperparameter Selection:** After evaluating all combinations, the function identifies the hyperparameters that achieve the highest mean AUC. It returns these optimal hyperparameters along with the corresponding AUC score and training step at which the best performance was achieved.

5.3 RobustFairnessMetrics Class

The `RobustFairnessMetrics` class implements evaluation metrics designed for measuring robustness performance in machine learning fairness evaluations. This class is particularly useful in scenarios where fairness metrics need to be computed across different protected groups and their intersections.

- **Initialization:** The class requires the initialization parameters such as the name of the target variable (`label_column_name`), a list of protected features (`protected_groups`), a list enumerating subgroup indices (`subgroups`), and an optional directory path for printing TensorFlow variables.
- **Control Dependencies for Printing:** It includes a method (`_get_control_dependencies_for_print`) to set up TensorFlow print operations for variables like example weights, labels, predictions, and protected groups, facilitating debugging and analysis.
- **Protected Group Masks:** Another method (`_get_protected_group_masks`) initializes binary masks for each protected group and their subgroups, based on the provided labels. This is crucial for isolating metrics based on different sensitive attributes.
- **Metric Functions:** The class defines various metric functions (`metrics_fn` and `metrics_fn_th`) for evaluating fairness metrics such as accuracy, recall, precision, true positives, false positives, true negatives, false negatives, false positive rate, false negative rate, and metrics at specific thresholds. These metrics are computed for the entire dataset, specific protected groups, and subgroups.
- **Fairness Metrics Function Creation:** Finally, the method `create_fairness_metrics_fn` creates a TensorFlow metric function suitable for use with `tf.estimator.add_metrics()`, integrating all defined metrics and their variations across different groups and thresholds.

6 Results

This section presents a thorough examination of the results obtained from our study, focusing on reproducibility, replicability, and performance analysis of the Adversarially Reweighted Learning (ARL) framework across multiple datasets. Our analysis includes a detailed comparison with the original results reported by Lahoti et al. [3] and provides critical insights into the strengths and limitations of the ARL approach.

6.1 Reproducibility

The goal was to reproduce the results of the ARL framework, adhering strictly to the procedures and hyperparameters described in the original paper. Using the default settings and hyperparameters, we noted some deviations in the AUC scores compared to the original results. Table 1 summarizes the AUC scores from our reproduced experiments alongside the original scores reported by Lahoti et al.[3]

6.1.1 Adjustments

We encountered various problems in the model implementation which required manual adjustments not detailed originally. The optimal hyperparameters were also not stated so we had to perform additional tuning. Here are the ones we found Table **??**:

Dataset	AUC (original)	AUC (reproduced)
Adult	0.907	0.904
LSAC	0.823	0.820
COMPAS	0.743	0.721
Students	/	0.726

Table 1: Original results and reproduced ones

Dataset	Batch size	LR learner	LR adversary	Train steps
Adult	256	0.01	1	990
LSAC	256	0.1	0.01	990
COMPAS	32	0.01	1	470
Students	16	0.001	0.001	190

Table 2: Best hyperparameters for each dataset

6.2 Performance evaluation

Here we present the performances we observed Table 3:

Dataset	AUC	Macro-AUC	Min-AUC	Minority-AUC
Adult	0.904	0.906	0.903	0.909
LSAC	0.820	0.809	0.755	0.826
COMPAS	0.721	0.726	0.686	0.702
Students	0.726	0.762	0.665	0.751

Table 3: Performance results on averaged AUCs values on different datasets.

To verify the statistical significance of the observed performance differences, t-tests comparing the AUC scores across different implementations were conducted. In Table 4 baseline indicate models that don't use Adversarial Networks and use the protected features.

Typically, a p-value threshold of 0.05 is used to determine statistical significance. If the p-value is below this threshold, the null hypothesis (no difference between models) is rejected, suggesting that the observed difference is statistically significant. All p-values obtained in this study (0.963 for UCI Adult, 0.917 for LSAC, 0.977 for COMPAS, and 0.999 for the Students dataset) are well above the 0.05 threshold (except LSAC which is still low enough). This indicates that there is no significant difference between the ARL model and the baseline models for any of the datasets.

ARL demonstrated improvements in AUC for the worst-case protected groups across all datasets. This improvement is particularly noteworthy in high-stakes applications such as healthcare and criminal justice, where fairness is critical. ARL consistently performed well in achieving high AUC for the worst-case protected groups and maintaining overall robustness against training data biases. Evaluations on four real-world datasets demonstrated ARL's effectiveness in improving fairness metrics without demographic data.

In summary, ARL has proven to be a robust and effective approach for enhancing fairness in machine learning models. By leveraging non-protected features and task labels, ARL addresses fairness issues without relying on sensitive demographic information, making it a valuable tool in privacy-constrained environments. The results confirm ARL's potential in achieving equitable performance across various datasets and scenarios, contributing to the broader goal of fair and just machine learning applications.

Dataset	**Model**	**AUC**	**Macro-AUC**	**Min-AUC**	**Minority-AUC**	P-value
Adult	ARL	0.904	0.906	0.903	0.909	0.956
	Baseline	0.904	0.905	0.902	0.908	
LSAC	ARL	0.820	0.809	0.755	0.826	0.917
	Baseline	0.821	0.811	0.766	0.825	
COMPAS	ARL	0.721	0.726	0.686	0.702	0.980
	Baseline	0.721	0.725	0.682	0.702	
Students	ARL	0.726	0.762	0.665	0.751	0.999
	Baseline	0.726	0.761	0.662	0.743	

Table 4: Significance testing on different datasets.

7 Conclusion

Improving model fairness without directly observing protected features presents a significant challenge in the practical application of machine learning fairness goals. Prior work has primarily focused on enhancing model performance for the worst-case distribution, which is notably susceptible to noisy outliers. The key insight is that improving model performance for worst-case groups benefits from concentrating the objective on computationally-identifiable regions of errors, i.e., regions within the input and label space that exhibit significant errors.

Adversarially Reweighted Learning (ARL) enhances AUC for worst-case protected groups across multiple datasets and various training data biases. This method provides a possible approach to achieving fairness in the absence of demographic data, making it particularly useful in scenarios where such data is unavailable or unusable due to privacy or legal constraints.

Through experiments, we observed that ARL not only improves the overall AUC but also manages to address class imbalance problems inherently.These results underscore the effectiveness of ARL in improving model fairness by focusing on regions with significant errors, thereby minimizing the impact of noisy outliers. This insight, coupled with ARL's performance, offers a promising direction for future research and applications in fair machine learning.

Despite these advancements, challenges remain, particularly in evaluating the effectiveness of methods like ARL in real-world applications where demographic data is absent. Future research should aim to validate the proposed approach across a wider variety of real-world scenarios and datasets to ensure its robustness and applicability.

8 References

[1] J. Rawls. 2001. Justice as fairness: A restatement. Harvard University Press
[2] T. B. Hashimoto, M. Srivastava, H. Namkoong, and P. Liang. 2018. Fairness Without Demographics in Repeated Loss Minimization. In ICML.
[3] Preethi Lahoti. 2020. Fairness without Demographics through Adversarially Reweighted Learning
[4] J. Mohazzab, L.R. Weytingh, C.A. Wortmann and B. Brocades Zaalberg. 2021. Reimplementing the Adversarially Reweighted Learning model by Lahoti et al. (2020) to improve fairness without demographics

9 Appendices

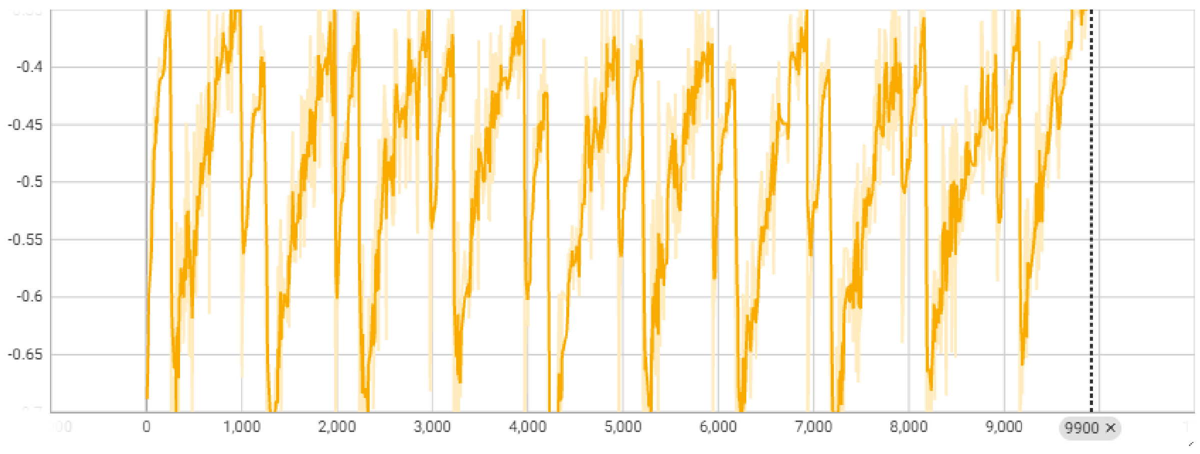

Figure 1: Adversarial loss on the Students dataset.

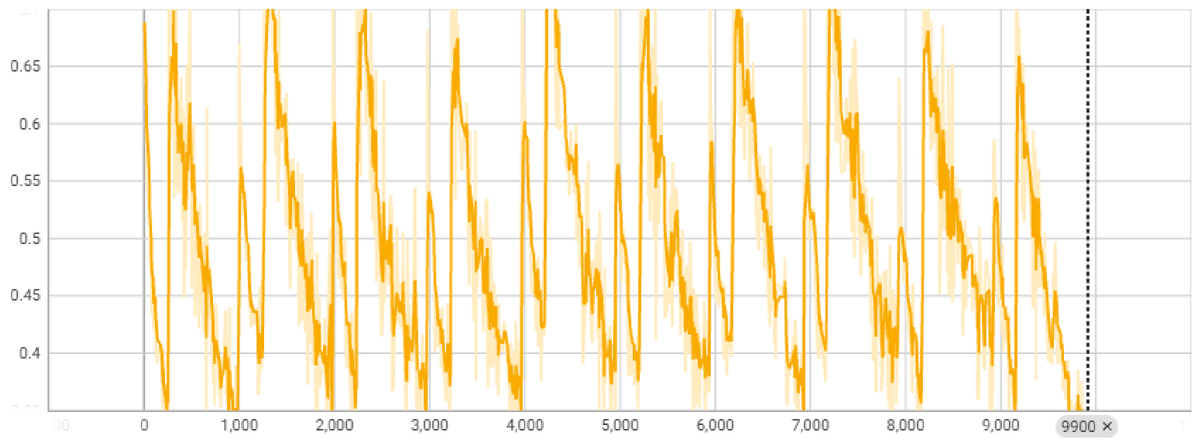

Figure 2: Learner loss on the Students dataset.

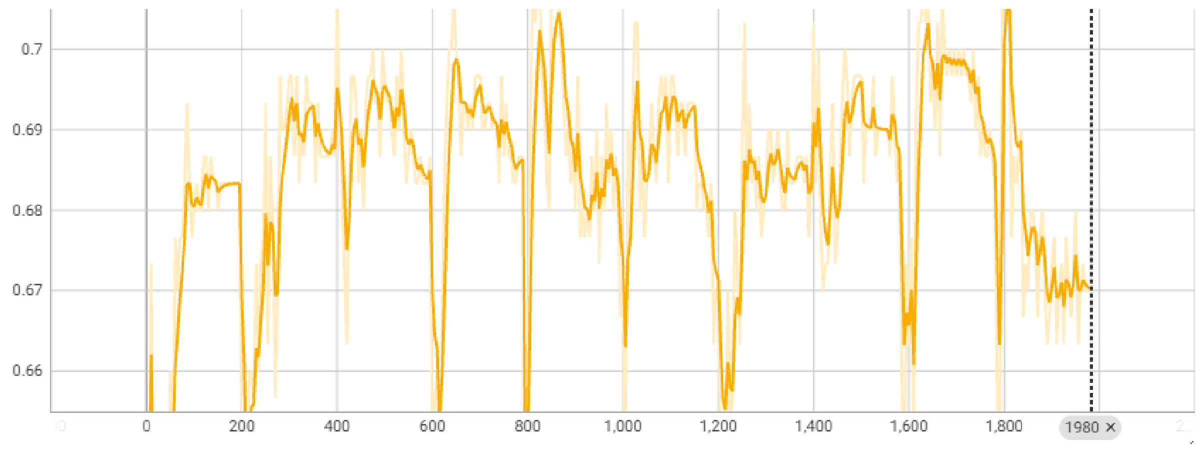

Figure 3: Accuracy on the Students dataset.

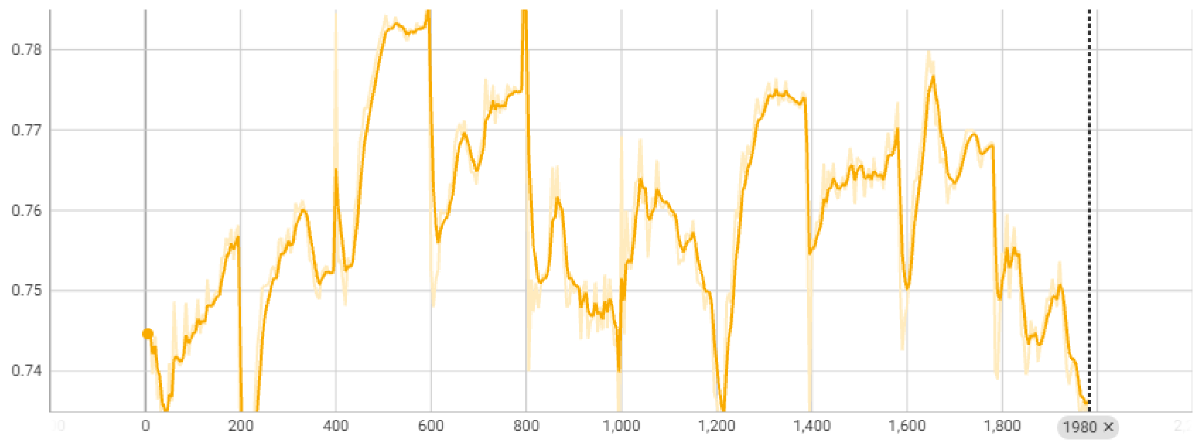

Figure 4: Macro-AUC loss on the Students dataset.

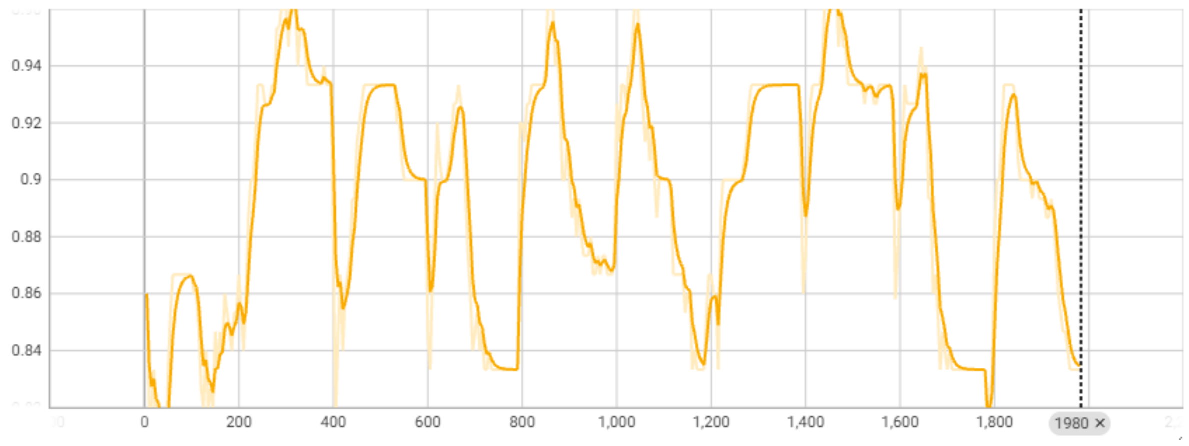

Figure 5: Max-AUC on the Students dataset.

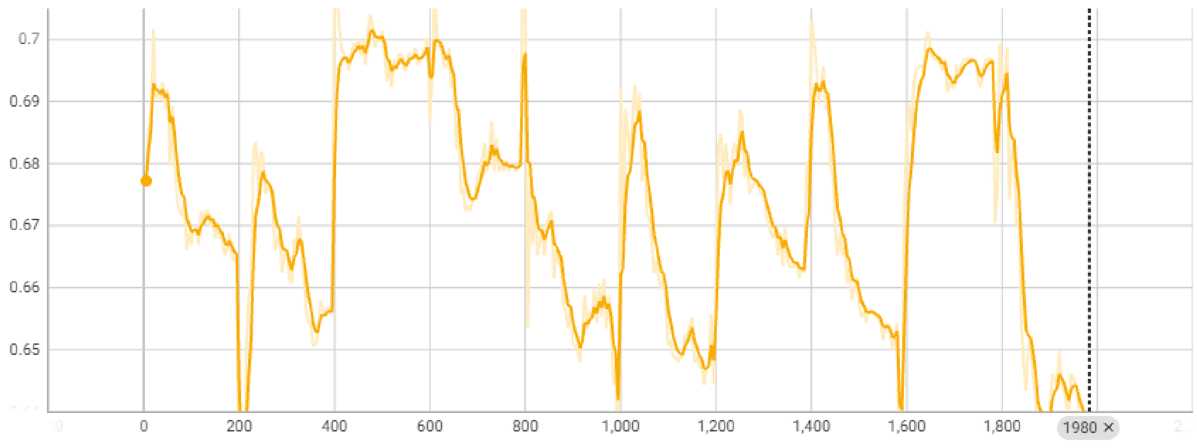

Figure 6: Min-AUC on the Students dataset.

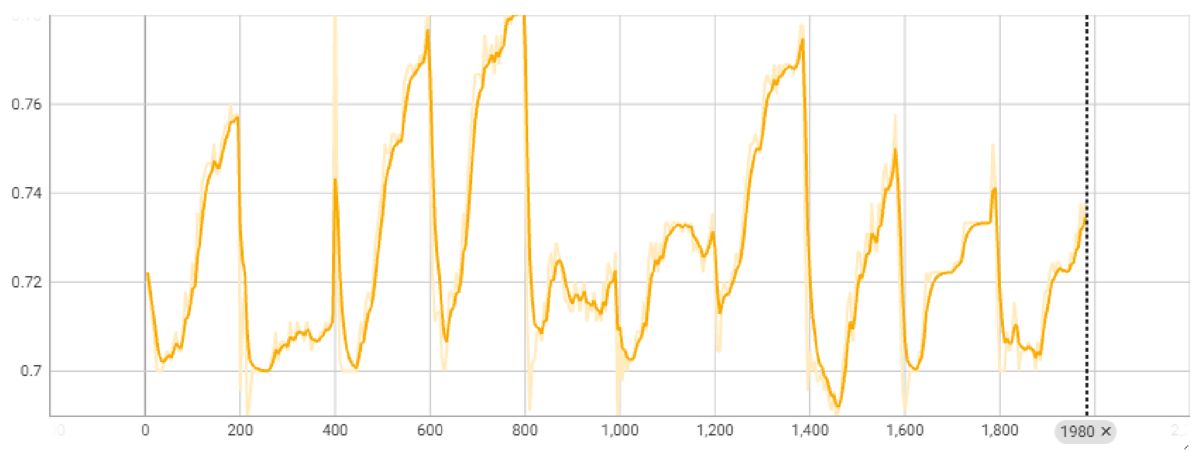

Figure 7: Minority subgroup - AUC on the Students dataset.

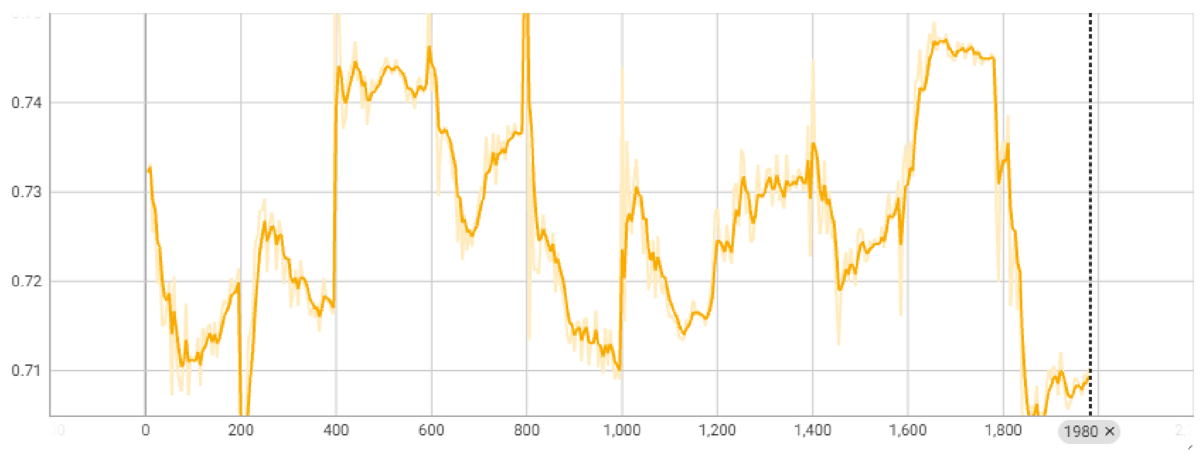

Figure 8: AUC on the Students dataset.