FairGBM

Gradient Boosting with Fairness Constraints

International Conference on Learning Representations
ICLR 2023

André Cruz Catarina Belém João Bravo

Pedro Saleiro Pedro Bizarro

Motivation

- ML has become ubiquitous in a multitude of high-stakes applications.
- At the same time, numerous reports have warned of AI bias in production systems.

RELEASE THE PRISONER —

Algorithms should have made courts more fair. What went wrong?

Biased Lending Evolves, and Blacks Face Trouble Getting Mortgages

Amazon scraps secret AI recruiting tool that showed bias against women

Motivation

- ML has become ubiquitous in a multitude of high-stakes applications.
- At the same time, numerous reports have warned of AI bias in production systems.

- GBMs are still the state-of-the-art on tabular data, often outperforming Deep Learning approaches.

Motivation

- ML has become ubiquitous in a multitude of high-stakes applications.
- At the same time, numerous reports have warned of AI bias in production systems.

- GBMs are still the state-of-the-art on tabular data, often outperforming Deep Learning approaches.
- There are no (fairness-)constrained optimization methods tailored for GBM.

Constrained Optimization w/ GBM

Goal: Training GBM to minimize loss under fairness constraints.

Constrained Optimization w/ GBM

Goal: Training GBM to minimize loss under fairness constraints.

$$\theta^* = \arg\min_{\theta \in \Theta} \overbrace{l(\theta)}^{\text{predictive loss}} \quad \text{s.t.}_{i \in [m]} \quad \overbrace{c_i(\theta) \leq 0}^{\text{constraints}}$$

Constrained Optimization w/ GBM

Goal: Training GBM to minimize loss under fairness constraints.

$$\theta^* = \arg\min_{\theta\in\Theta} \overbrace{l(\theta)}^{\text{predictive loss}} \quad \underset{i\in[m]}{\text{s.t.}} \quad \overbrace{c_i(\theta) \leq 0}^{\text{constraints}}$$

Approach

- The *standard* approach for constrained optimization is to use the method of Lagrange multipliers.

$$\mathcal{L}(\underbrace{\theta}_{\text{model parameters}}, \underbrace{\lambda}_{\text{Lagrange multipliers}}) = l(\theta) + \sum_{i=1}^{m} \lambda_i c_i(\theta)$$

Constrained Optimization w/ GBM

Goal: Training GBM to minimize loss under fairness constraints.

$$\theta^* = \arg\min_{\theta \in \Theta} \overbrace{l(\theta)}^{\text{predictive loss}} \quad \text{s.t.}_{i \in [m]} \quad \overbrace{c_i(\theta) \leq 0}^{\text{constraints}}$$

Approach

- The *standard* approach for constrained optimization is to use the method of Lagrange multipliers.

$$\mathcal{L}(\underbrace{\theta}_{\text{model parameters}}, \underbrace{\lambda}_{\text{Lagrange multipliers}}) = l(\theta) + \sum_{i=1}^{m} \lambda_i c_i(\theta)$$

- Solution lies on saddle point of the Lagrangian function, $\mathcal{L}(\theta, \lambda)$.

Constrained Optimization w/ GBM

Approach

- Finding the saddle-point of $\mathcal{L}(\theta, \lambda) = l(\theta) + \sum_{i=1}^{m} \lambda_i c_i(\theta)$, at $\nabla \mathcal{L}(\theta, \lambda) = \vec{0}$.

Constrained Optimization w/ GBM

Approach

- Finding the saddle-point of $\mathcal{L}(\theta, \lambda) = l(\theta) + \sum_{i=1}^{m} \lambda_i c_i(\theta)$, at $\nabla \mathcal{L}(\theta, \lambda) = \vec{0}$.

- In practice, this is done via iterative gradient $\underbrace{\text{descent}}_{\theta\text{-player}}$ and $\underbrace{\text{ascent}}_{\lambda\text{-player}}$ steps.

Constrained Optimization w/ GBM

Approach

- Finding the saddle-point of $\mathcal{L}(\theta, \lambda) = l(\theta) + \sum_{i=1}^{m} \lambda_i c_i(\theta)$, at $\nabla \mathcal{L}(\theta, \lambda) = \vec{0}$.

- In practice, this is done via iterative gradient $\underbrace{\text{descent}}_{\theta\text{-player}}$ and $\underbrace{\text{ascent}}_{\lambda\text{-player}}$ steps.

- Gradient descent over θ : $\dfrac{\partial \mathcal{L}}{\partial \theta} = \dfrac{\partial l}{\partial \theta} + \sum_{i=1}^{m} \lambda_i \dfrac{\partial c_i}{\partial \theta}$

- Gradient ascent over λ : $\dfrac{\partial \mathcal{L}}{\partial \lambda_i} = c_i(\theta)$

Constrained Optimization w/ GBM

Approach

- Finding the saddle-point of $\mathcal{L}(\theta, \lambda) = l(\theta) + \sum_{i=1}^{m} \lambda_i c_i(\theta)$, at $\nabla \mathcal{L}(\theta, \lambda) = \vec{0}$.

- In practice, this is done via iterative gradient descent and ascent steps.

  θ-player λ-player

- Gradient descent over θ : $\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial l}{\partial \theta} + \sum_{i=1}^{m} \lambda_i \frac{\partial c_i}{\partial \theta}$ we require differentiable constraints

- Gradient ascent over λ : $\frac{\partial \mathcal{L}}{\partial \lambda_i} = c_i(\theta)$ but *only* for the descent step!

Constrained Optimization w/ GBM

feedzai

Approach

- Finding the saddle-point of $\mathcal{L}(\theta, \lambda) = l(\theta) + \sum_{i=1}^{m} \lambda_i c_i(\theta)$, at $\nabla \mathcal{L}(\theta, \lambda) = \vec{0}$.

- In practice, this is done via iterative gradient descent and ascent steps.

  θ-player λ-player

- Gradient descent over θ : $\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial l}{\partial \theta} + \sum_{i=1}^{m} \lambda_i \frac{\partial c_i}{\partial \theta}$ we require differentiable constraints

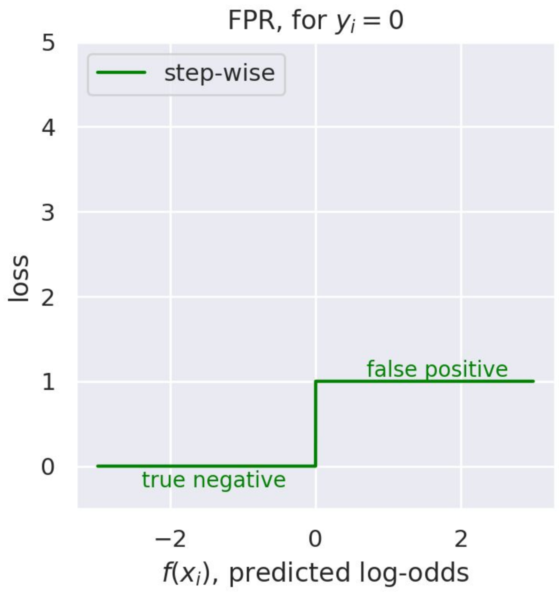
- Gradient ascent over λ : $\frac{\partial \mathcal{L}}{\partial \lambda_i} = c_i(\theta)$ but *only* for the descent step!

However, fairness metrics are **non-differentiable** (and also non-convex)

Differentiable Proxy Metrics

Issue:

Fairness metrics are non-differentiable (and non-convex).



Differentiable Proxy Metrics

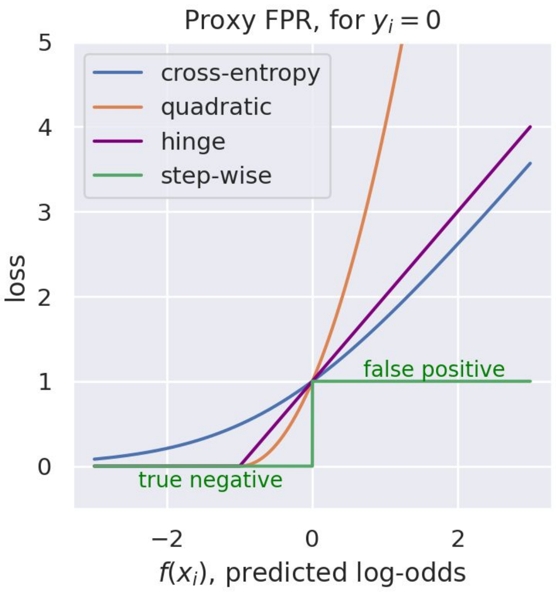
Issue:

Fairness metrics are non-differentiable (and non-convex).

Solution

- Use a differentiable upper-bound proxy: $c_i(\theta) \leq \tilde{c}_i(\theta) \leq 0$.
  - Gradient descent on the proxy-Lagrangian:

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \theta} = \frac{\partial l}{\partial \theta} + \sum_{i=1}^{m} \lambda_i \frac{\partial \tilde{c}_i}{\partial \theta}$$



Differentiable Proxy Metrics

Issue:

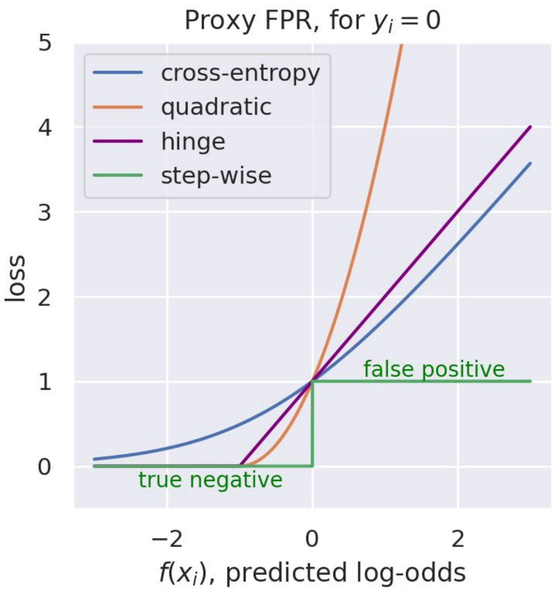
Fairness metrics are non-differentiable (and non-convex).

Solution

- Use a differentiable upper-bound proxy: $c_i(\theta) \leq \tilde{c}_i(\theta) \leq 0$.
  - Gradient descent on the proxy-Lagrangian:

$$\frac{\partial \tilde{\mathcal{L}}}{\partial \theta} = \frac{\partial l}{\partial \theta} + \sum_{i=1}^{m} \lambda_i \frac{\partial \tilde{c}_i}{\partial \theta}$$

- However, we'll be doing gradient descent and ascent on different functions: $\tilde{\mathcal{L}}, \mathcal{L}$.



Randomized Classifiers

Issue

- We're doing gradient descent over $\tilde{\mathcal{L}}$, and ascent over $\mathcal{L}$.
    - It's no longer a zero-sum game.
- Our proxy constraints are still non-convex (although differentiable).

Randomized Classifiers

Issue

- We're doing gradient descent over $\tilde{\mathcal{L}}$, and ascent over $\mathcal{L}$.
    - It's no longer a zero-sum game.
- Our proxy constraints are still non-convex (although differentiable).

Solution

- Instead of finding a pure Nash equilibrium: a single (θ, λ) pair (a single classifier).
- The solution lies on a distribution over (θ, λ) pairs (a distribution of classifiers) [Cotter et al., 2019]
    - A **randomized classifier**.

Randomized Classifiers

Related Work

- [Agarwal et al., 2018]: an *ensemble* of standard strong learners.
    - General method that is also compatible with GBM.
    - Increases training time dramatically (default: 50x increase).

Randomized Classifiers

feedzai

Related Work

- [Agarwal et al., 2018]: an *ensemble* of standard strong learners.
    - General method that is also compatible with GBM.
    - Increases training time dramatically (default: 50x increase).

FairGBM

- Boosting fits an additive model: $f(x) = \sum_{i=1}^{M} \beta_i \underbrace{b(x, \gamma_i)}$.

  additive basis function
  (e.g., regression tree for GBM)

- Model iterate at iteration *k* : simply add the first *k* trees!

Randomized Classifiers

Related Work

- [Agarwal et al., 2018]: an *ensemble* of standard strong learners.
    - General method that is also compatible with GBM.
    - Increases training time dramatically (default: 50x increase).

FairGBM

- Boosting fits an additive model: $f(x) = \sum_{i=1}^{M} \beta_i \underbrace{b(x, \gamma_i)}_{\text{additive basis function (e.g., regression tree for GBM)}}$.

- Model iterate at iteration *k* : simply add the first *k* trees!

- FairGBM $\underbrace{\text{randomized classifier}}_{\text{over the model iterates}}$ carries virtually **no CPU or memory overhead**.

FairGBM Training Algorithm

feedzai

Algorithm 1 *FairGBM* training pseudocode

Input: $T \in \mathbb{N}$, number of boosting rounds
$\mathcal{L}, \tilde{\mathcal{L}} : \mathcal{F} \times \mathbb{R}_+^m \rightarrow \mathbb{R}$, Lagrangian and proxy-Lagrangian
$\eta_f, \eta_\lambda \in \mathbb{R}_+$, learning rates

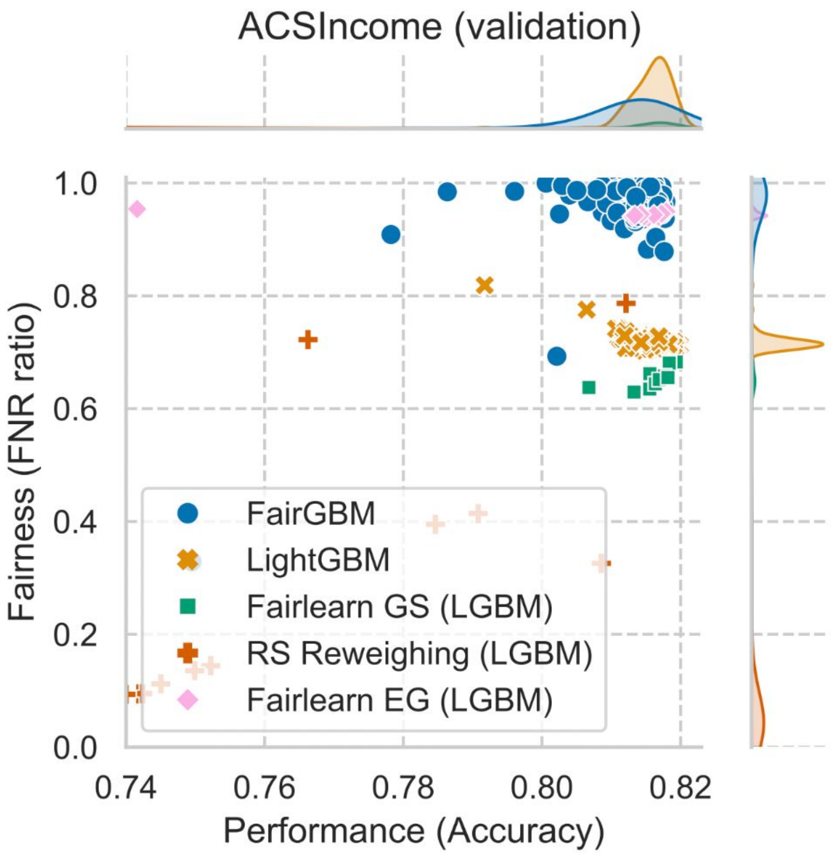
Let $h_0 = \arg\min_{\gamma \in \mathbb{R}} \tilde{\mathcal{L}}(\gamma, 0)$ ▷ Initial constant "guess"
Initialize $f \leftarrow h_0$
Initialize $\lambda \leftarrow 0$
for $t \in \{1, \ldots, T\}$ **do**
    Let $g_i = \frac{\partial \tilde{\mathcal{L}}(f(x_i), \lambda)}{\partial f(x_i)}$ ▷ Gradient of proxy-Lagrangian w.r.t. model
    Let $\Delta = \frac{\partial \mathcal{L}(f(x_i), \lambda)}{\partial \lambda}$ ▷ Gradient of Lagrangian w.r.t. multipliers
    Let $h_t = \arg\min_{h_t \in \mathcal{H}} \sum_{i=1}^{N} \left(-g_i - h_t(x_i)\right)^2$ ▷ Fit base learner
    Update $f \leftarrow f + \eta_f h_t$ ▷ Gradient descent
    Update $\lambda \leftarrow (\lambda + \eta_\lambda \Delta)_+$ ▷ Projected gradient ascent
return $h_0, \ldots, h_T$

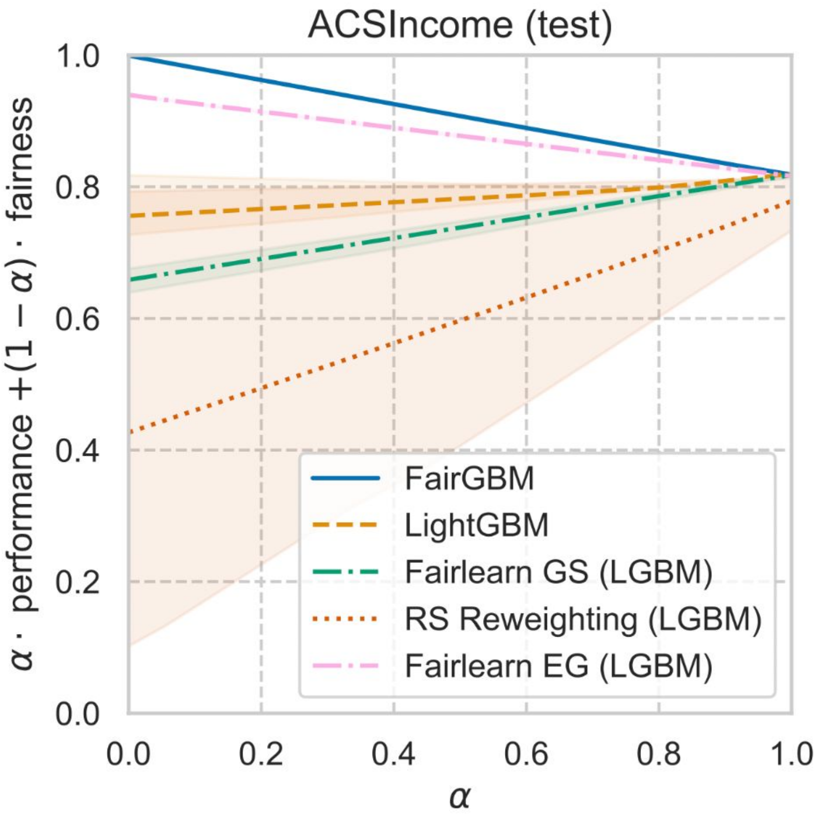
Experimental Setup

- Datasets
  - ACS folktables datasets [Ding et al., 2021]
    - Five public datasets based on US census data;
    - Includes modern-day version of UCI Adult dataset;
    - 1M to 3M rows;
  - Account Opening Fraud dataset
    - In-house real-world data stream of account opening fraud on a major European bank;
    - 500K rows;

- Literature baselines:
  - Fairlearn Exponentiated Gradient Reduction [Agarwal et al., 2018]
  - Fairlearn Grid Search [Agarwal et al., 2018]
  - Fairlearn Random Search
  - Unconstrained LightGBM [Ke et al., 2017]

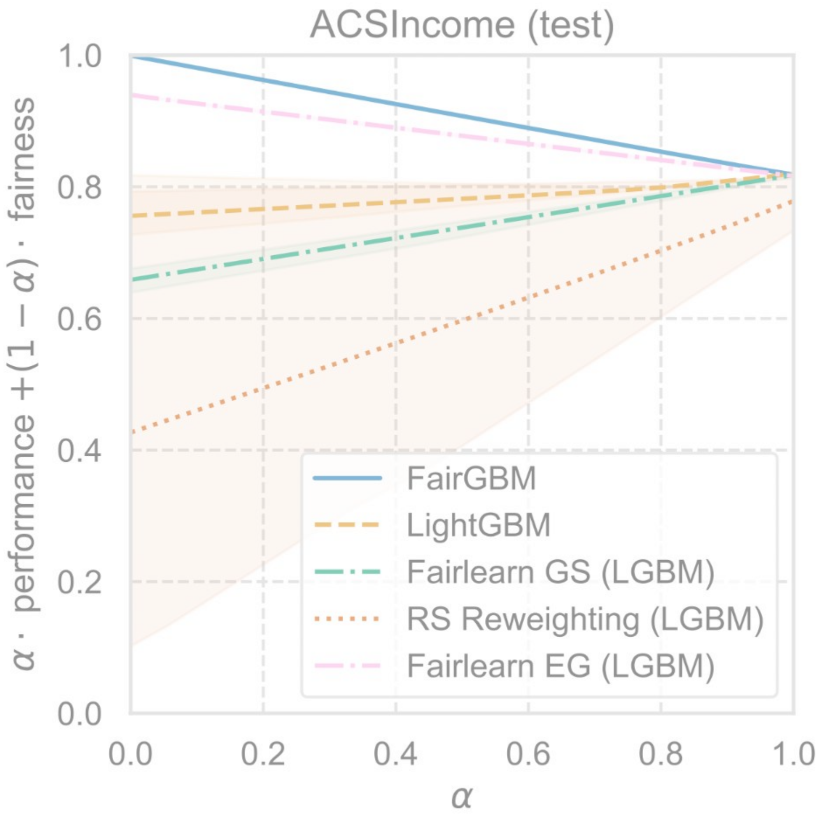
Results on ACSIncome dataset





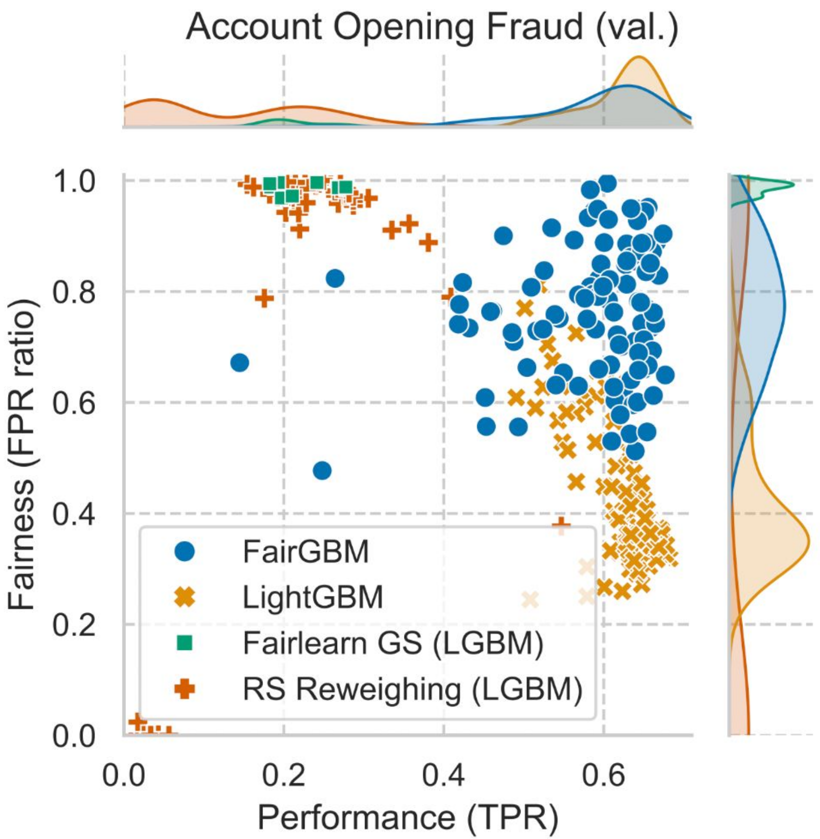
Results on ACSIncome dataset

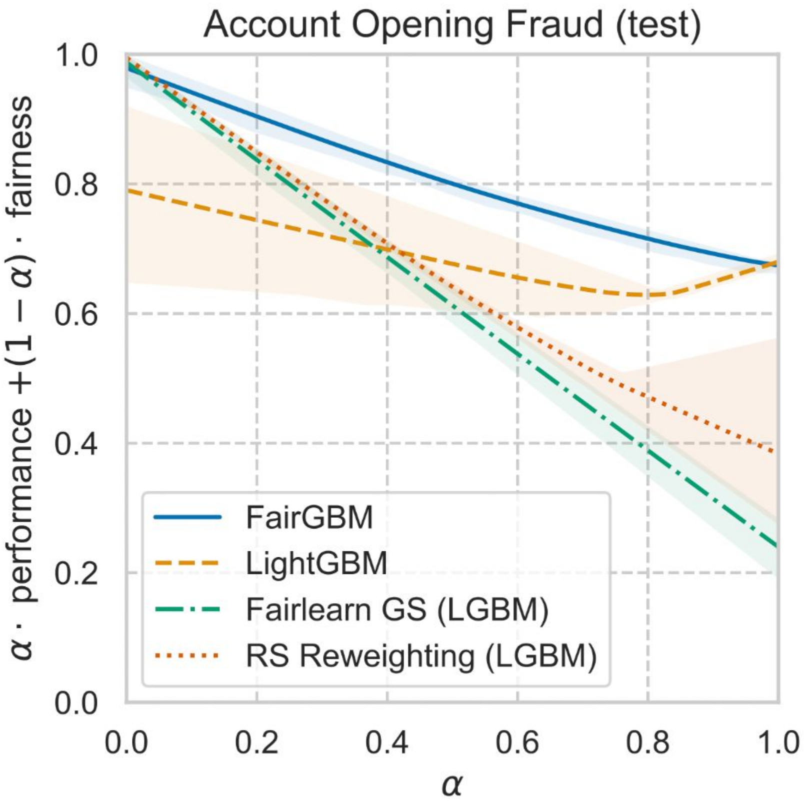
Algorithm	Run-time	
	Total (h)	**Relative**
	ACSIncome	
FairGBM	9.9	x2.1
LightGBM	4.6	*baseline*
GS	43.8	x9.6
RS	37.1	x8.1
EG	99.4	x21.7



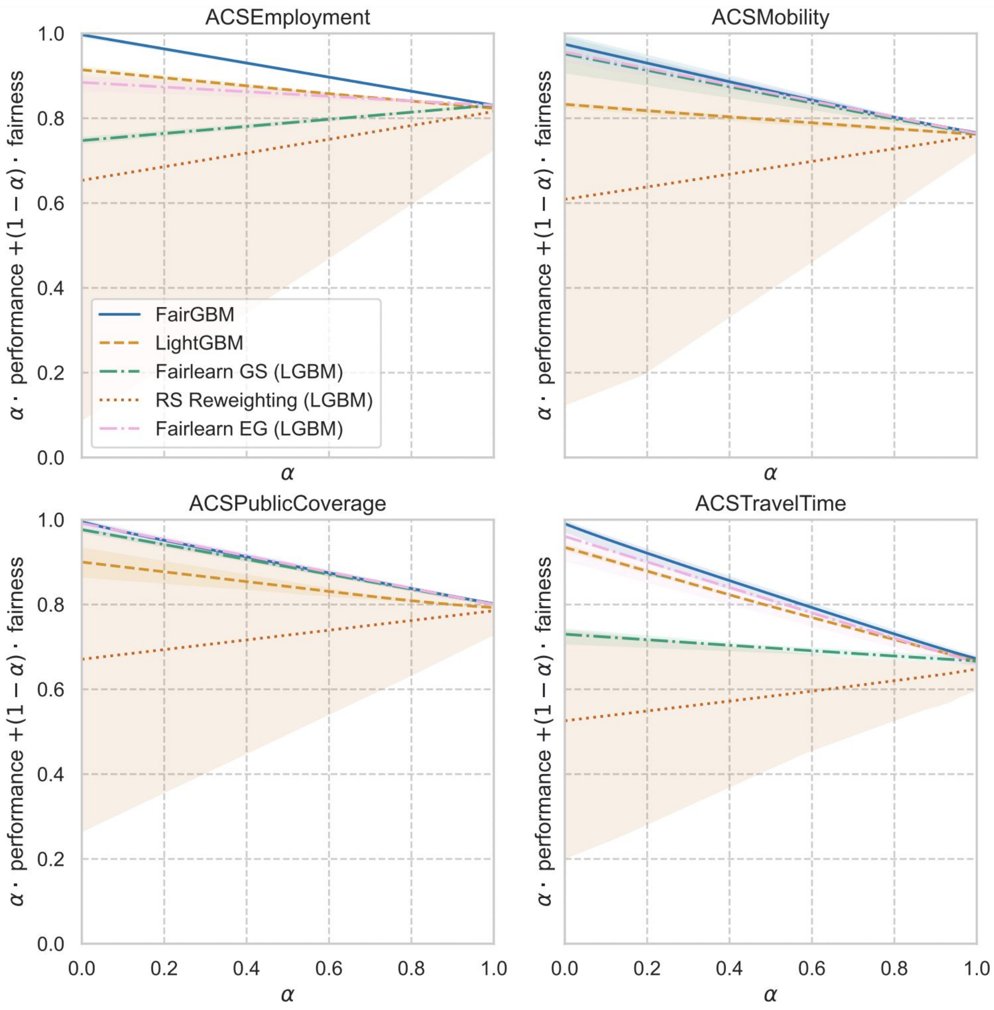
Results on AOF dataset

feedzai

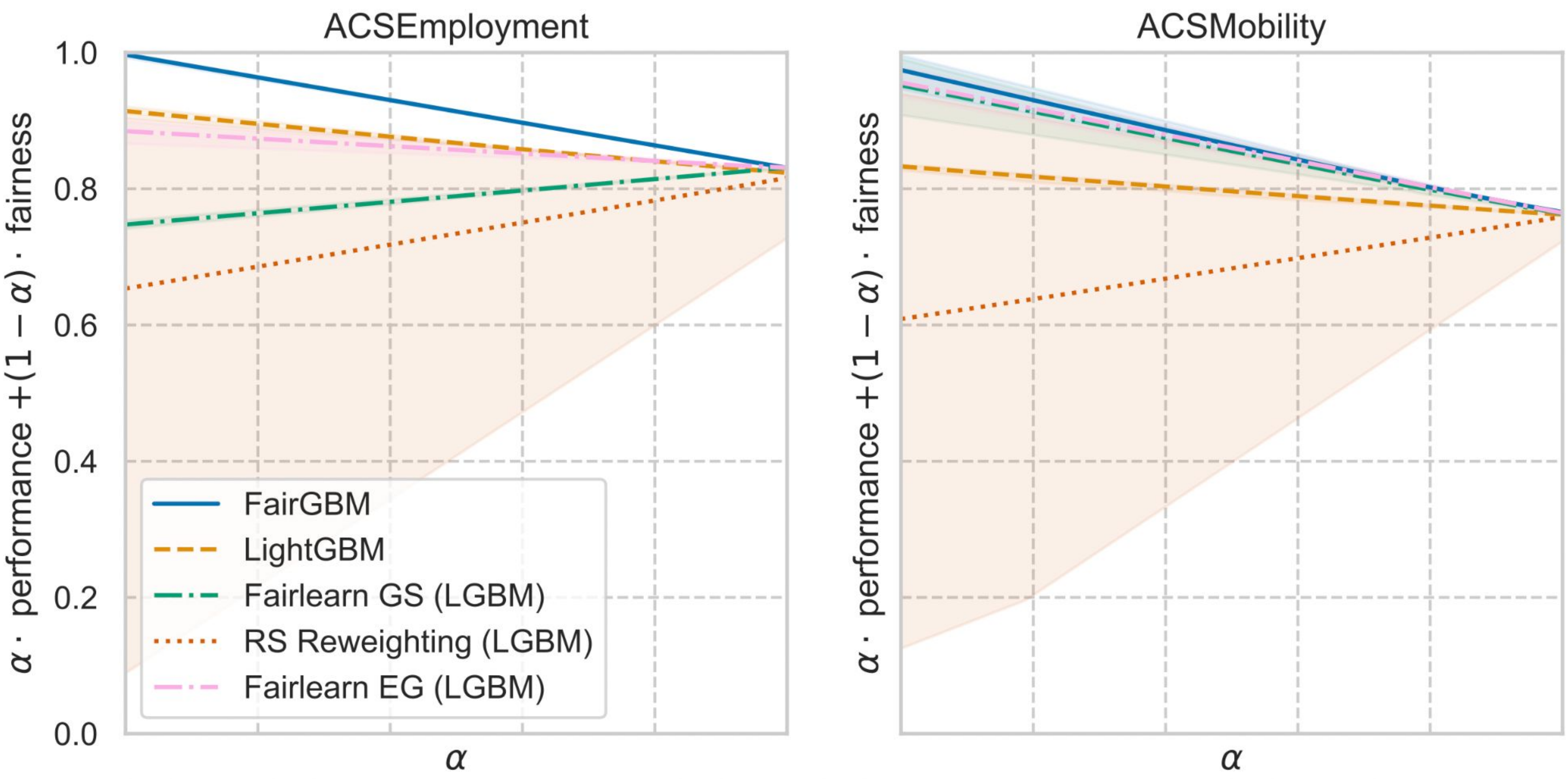




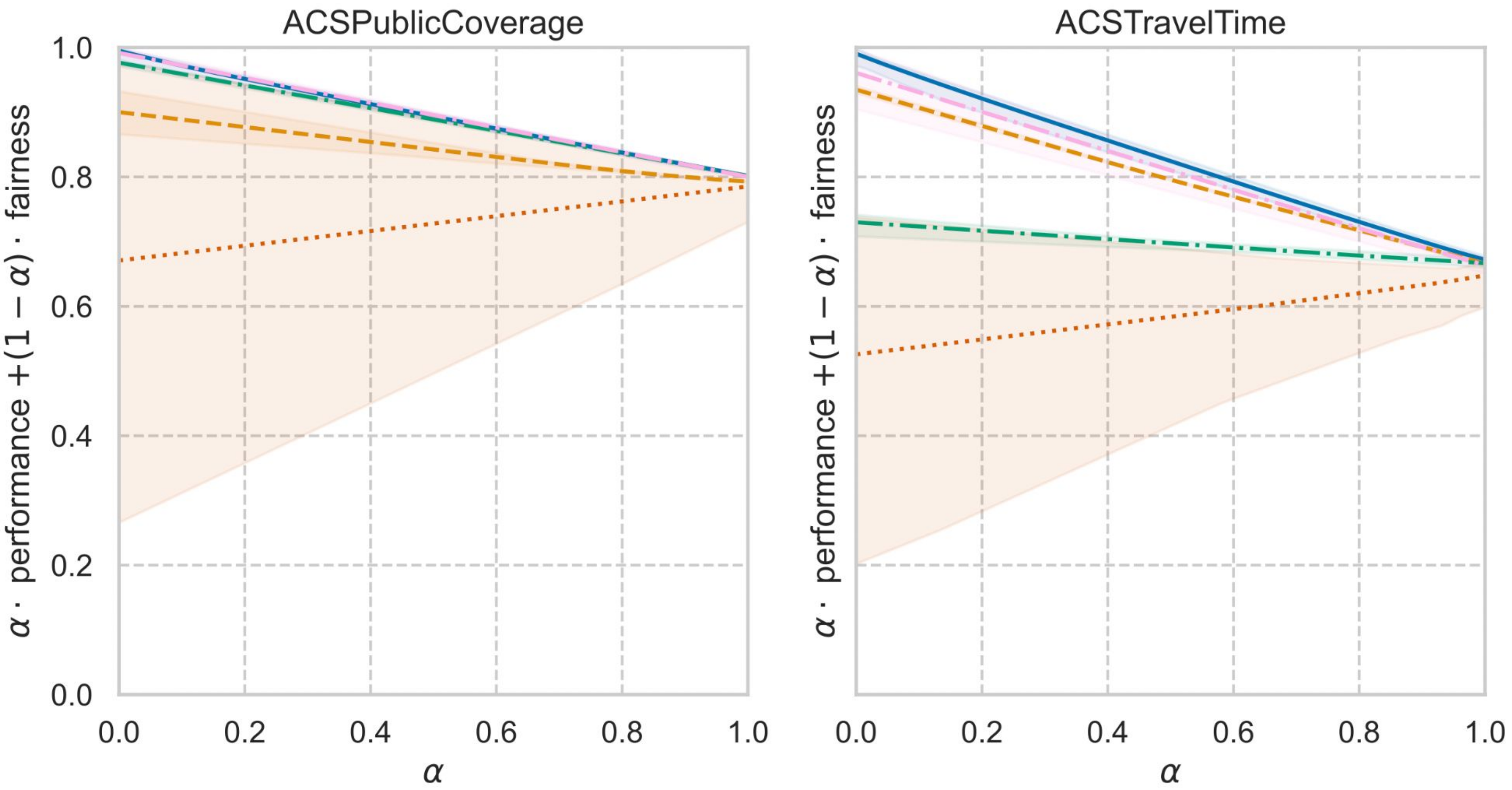
Other Datasets



Results on ACS datasets



Results on ACS datasets



Open-Source Repository

feedzai

- Open-source package available at

  https://github.com/feedzai/fairgbm/

- Try it out for yourself (on linux)

```
from fairgbm import FairGBMClassifier

# Instantiate
fairgbm_clf = FairGBMClassifier(
    constraint_type="FNR",    # constraint on equal group-wise TPR (equal opportunity)
    n_estimators=200,         # core parameters from vanilla LightGBM
    random_state=42,          # ...
)

# Train using features (X), labels (Y), and sensitive attributes (S)
fairgbm_clf.fit(X, Y, constraint_group=S)
# NOTE: labels (Y) and sensitive attributes (S) must be in numeric format

# Predict
Y_test_pred = fairgbm_clf.predict_proba(X_test)[:, -1]  # Compute continuous class probabilities (re
# Y_test_pred = fairgbm_clf.predict(X_test)              # Or compute discrete class predictions
```

Thank You

Check out the paper at https://arxiv.org/pdf/2209.07850

Check out the code at https://github.com/feedzai/fairgbm

feedzai