A. Details of the MPC Protocols

Secret sharing. A secret sharing scheme allows one to split a value x (the secret) among two parties, so that no party has unilateral access to x. In our setting, a user Alice will secret share a sensitive value, for example her race, among a modeler M and a regulator REG. Several secret sharing schemes exist, including *Shamir secret sharing*, *xor sharing*, *Yao sharing*, or *arithmetic multiplicative/additive sharing*. In this work we alternate between Yao sharing and additive sharing for efficiency. In the latter, the value x is represented in a finite domain $\mathbb{Z}_q$ with, for example $q = 2^{32}$. To share her race, Alice samples a value r from $\mathbb{Z}_q$ uniformly at random, and sends $x - r$ to M and r to REG. We call each of $x - r$ and r a *share*, and denote them as $\langle x \rangle_1$ and $\langle x \rangle_2$. Now M and REG can recover x by adding their shares, but each share on its own does not reveal anything about the value of x (other than that it is smaller than q). Note that the case where $q = 2$ corresponds to xor sharing.

Function evaluation. MPC can be classified in two groups depending on how f is represented: either as a Boolean or arithmetic circuit. All protocols proceed by having the parties jointly evaluate the circuit, processing it gate by gate. For each gate g for which the value for the input wires x, y is shared among the parties, the parties run a subprotocol to produce the value $z = g(x, y)$ of the output wire, again shared, without revealing any information in the process. In the setting where we use arithmetic additive sharing, the two parties M and REG hold shares, $\langle x \rangle_1$,$\langle y \rangle_1$ and $\langle x \rangle_2$,$\langle y \rangle_2$, respectively. In this case, f is represented as an arithmetic circuit, and hence each gate g in the circuit is either an addition or a multiplication. Note that if g is an addition gate, then a sharing of $z = g(x, y)$ can be obtained by having each party simply compute locally, i.e., without any interaction, $\langle z \rangle_i = \langle x \rangle_i + \langle y \rangle_i$, for $i \in \{1, 2\}$. If g is a multiplication, the subprotocol to compute shares of z is much more costly. Fortunately, it can be divided into an offline and an online phase.

The preprocessing model in MPC. In this model, two parties P_1, P_2 engage in an offline phase, which is data independent, and compute (and store) *shared multiplication triples* of the form (a, b, c), with $c = ab$. Here, $a, b \in \mathbb{F}_q$ are drawn uniformly at random, and each value a, b, c is shared among the parties as explained above. In the online phase, a multiplication gate $z = \texttt{mul}(x, y)$ on shared values x, y can be evaluated as follows: (1) each P_i sets $\langle e \rangle_i = \langle x \rangle_i - \langle a \rangle_i$ and $\langle f \rangle_i = \langle y \rangle_i - \langle b \rangle_i$, (2) the parties exchange their shares of e and f and reconstruct these values locally, and (3) each P_i computes $\langle z \rangle_i = (i-1)ef + f\langle a \rangle_i + e\langle b \rangle_i + \langle c \rangle_i$. The correctness of this protocol can be easily checked. Privacy relies on the uniform randomness of a, b, and hence $\langle e \rangle_i$ and $\langle f \rangle_i$ completely mask the values of $\langle x \rangle_i$ and $\langle y \rangle_i$, respectively. For a formal proof see (Demmler et al., 2015b).

Hence, for each multiplication in the function to be evaluated, the parties need to jointly generate a multiplication triple in advance. For computations with many multiplications (like in our case) this can be a costly process. However, this constraint is easy to accommodate in our architecture for private fair model training, as M and REG can run the offline phase once "overnight". Arithmetic multiplication via precomputed triples is a common technique, used in several popular MPC frameworks (Demmler et al., 2015b; Damgård et al., 2012). In this setting, several protocols for triple generation (which we did not describe) are available (Keller et al., 2018), and under continuous improvement. These protocols are often based on either Oblivious Transfer or Homomorphic Encryption.

The two-server model for multi-party learning. Due to a sequence of theoretical and engineering breakthroughs, in the last three decades MPC has gone from being a mathematical curiosity to a technology of practical interest with commercial applications. Several generic protocols for MPC exists, such as the ones based on arithmetic sharing (Damgård et al., 2012), garbled circuits (Yao, 1986), or GMW (Goldreich et al., 1987), with several available implementations (Demmler et al., 2015b; Zahur & Evans, 2015b). These protocols have different trade-offs in terms of the number of parties they support, network requirements, and scalability for different kinds of computations. In our work, we focus on the 2-party case, as the MPC computation is done by M and REG. The idea of privately outsourcing computation to two non-colluding parties in this way is recurrent in MPC, and often referred to as the two-server model (Mohassel & Zhang, 2017; Gascón et al., 2017; Nikolaenko et al., 2013b; Al-Rubaie et al., 2017).

While generic protocols exist, these do not yet scale to input sizes typically encountered in machine learning applications like ours. To circumvent this limitation, techniques tailored to specific applications have been proposed. Our protocols fall in this category, extending the SGD protocol from (Mohassel & Zhang, 2017), in which the following useful accelerating techniques are presented.

- Efficient rescaling: As our arithmetic shares represent fixed-point numbers, we need to rescale by the precision p after every multiplication. This involves dividing by 2^p, an expensive operation to do in MPC, and in particular in arithmetic sharing. Mohassel et al. show an elegant solution to this problem: the parties can rescale locally by dropping p bits of their shares. It is not hard to see that this might produce the wrong result. However, the parameters of the arithmetic secret sharing scheme can be set such that with a tunable arbitrarily large probability the error is at most ± 1. This trick can be used for any division by a power of two.

- Alternating sharing types: As already pointed out in previous work (Demmler et al., 2015b), alternating between secret sharing schemes can provide significant acceleration for some applications. Intuitively, arithmetic operations are fast in arithmetic shares, while comparisons are fast in schemes that represent functions as Boolean circuits. Examples of the latter are the GMW protocol and Yao's garbled circuits. In our implementation, we follow this recipe and implement matrix-vector multiplication using arithmetic sharing, while for evaluating our variant of sigmoid, we rely on the protocol from (Mohassel & Zhang, 2017) implemented with garbled circuits using the Obliv-C framework (Zahur & Evans, 2015b).

- Matrix multiplication triples: Another observation made by Mohassel et al. is that the idea described above for preprocessing multiplications over arithmetic shares can be reinterpreted at the level of matrices. This results in a faster online and offline phase (see (Mohassel & Zhang, 2017) for details).

How to prove that a protocol is secure. We did not provide a formal definition of security in this paper, and instead referred the reader to (Mohassel & Zhang, 2017). In MPC, privacy in the case of semi-honest adversaries is argued in the simulation paradigm (see (Goldreich, 2004) or (Lindell, 2016) for formal definitions and detailed proofs). Intuitively, in this paradigm one proves that every inference that a party—in our case either REG or M—could draw from observing the execution trace of the protocol could also be drawn from the output of the execution and the party's input. This is done by proving the existence of a *simulator* that can produce an execution trace that is indistinguishable from the actual execution trace of the protocol. A crucial point is that the simulator only has access to the input and output of the party being simulated.

B. Details of Fair Model Training

B.1. The Fair Training Algorithm

Algorithm 1 describes the computations M and REG have to perform for fair model training using the Lagrangian multiplier technique and the $p\%$-rule from eq. (9). In the next subsection we describe the parameter values. We implicitly assume all computations are performed jointly on additively shared secrets by M and REG as described in Section 3. This means that M and REG each receive a secret share of the protected attributes $\mathbf{Z}$. Following the protocols outlined in Section 3, they can then jointly evaluate the steps in Algorithm 1. This allows them to operate on the sensitive values within the MPC computation, while preventing unilateral access to them by M and REG. The result of these computations is the same as evaluating the algorithm as described with data in the clear.

BLOCKEDMULTSHIFTAVG stands for the blocked matrix multiplication to avoid overflow for fixed-point numbers described towards the end of Section 4. Note that it already contains the division by n. The averaging within the blocked matrix multiplications as well as over the results thereof are done by fast bit shifts instead of slow MPC division circuits. This is possible, because we chose all parameters such that divisions are always by powers of two.

We found the piecewise linear approximation of the sigmoid function introduced in (Mohassel & Zhang, 2017)

$$\textsc{SigmoidApprox}(x) := \begin{cases} 0 & \text{if } x \leq -\frac{1}{2}\,, \\ x + \frac{1}{2} & \text{if } -\frac{1}{2} < x < \frac{1}{2}\,, \\ 1 & \text{if } x \geq \frac{1}{2}\,. \end{cases}$$

to work best, see Figure 4.

Algorithm 1 Fair model training with private sensitive values using Lagrangian multipliers for $\mathbb{F}(\boldsymbol{\theta}) = {}^1\!/\!_n|\mathbf{Z}^\top X| - \mathbf{c}$.

Parameters: M, REG.
Input: (M) $\langle \mathbf{Z} \rangle_\mathbf{1} \in \mathbb{Z}_q^{n\times p}$
Input: (REG) $\mathbf{X} \in \mathbb{Z}_q^{n\times d}$, $\mathbf{y} \in \mathbb{Z}_q^n$, $\langle \mathbf{Z} \rangle_\mathbf{2} \in \mathbb{Z}_q^{n\times p}$
Input: (Public) Learning rates $\eta_{\boldsymbol{\theta}}, \eta_{\boldsymbol{\lambda}}$, number of training examples n, minibatch size 2^s, constraints $\mathbf{c} \in \mathbb{Z}_q^p$, number of epochs N_e.

```
1:  $\boldsymbol{\theta} \leftarrow \mathbf{0}, \boldsymbol{\lambda} \leftarrow \mathbf{0}$
2:  $\mathbf{A} \leftarrow \textsc{BlockedMultShiftAvg}(\mathbf{Z}^\top, \mathbf{X})$
3:  for all $j$ from 1 to $N_e$ do
4:    for all $i$ from 1 to ${}^n\!/\!_{2^s}$ do
5:      $(\mathbf{X}_i, \mathbf{y}_i) \leftarrow \textsc{SampleMinibatch}(\mathbf{X}, \mathbf{y})$
6:      $\mathbb{F} \leftarrow |\mathbf{A}\boldsymbol{\theta}| - c$
7:      $\nabla_{\boldsymbol{\lambda}} \leftarrow \max\{\mathbb{F}, \mathbf{0}\}$
8:      $\sigma \leftarrow \textsc{SigmoidApprox}(\mathbf{X}_i\boldsymbol{\theta})$
9:      $\nabla_{\boldsymbol{\theta}}^{\text{BCE}} \leftarrow \textsc{ShiftDivide}(\mathbf{X}_i^\top(\sigma - \mathbf{y}_i), 2^s)$
10:     $\nabla_{\boldsymbol{\theta}}^{\text{CON}} \leftarrow \begin{cases} \mathbf{A}^\top\boldsymbol{\lambda}, & \text{if } \mathbf{A} > \mathbf{0} \wedge \mathbb{F} > 0 \\ -\mathbf{A}^\top\boldsymbol{\lambda}, & \text{if } \mathbf{A} < \mathbf{0} \wedge \mathbb{F} > 0 \\ \mathbf{0}, & \text{if } \mathbb{F} \leq 0 \end{cases}$
11:     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_{\boldsymbol{\theta}}(\xi_j^{\text{BCE}}\nabla_{\boldsymbol{\theta}}^{\text{BCE}} + \xi_j^{\text{CON}}\nabla_{\boldsymbol{\theta}}^{\text{CON}})$
12:     $\boldsymbol{\lambda} \leftarrow \max\{\boldsymbol{\lambda} + \eta_{\boldsymbol{\lambda}}\nabla_{\boldsymbol{\lambda}}, \mathbf{0}\}$
13:   end for
14: end for
```

Output: Parameters $\boldsymbol{\theta}$

B.2. Description of Training Parameters

All our experiments use a batch size of 64, a fixed number of epochs scaling inversely with dataset size n (such that we always perform roughly 15 000 gradient updates), fixed learning rates of $\eta_{\boldsymbol{\theta}} = 10^{-4}, \eta_{\boldsymbol{\lambda}} = 0.05$, and an annealing

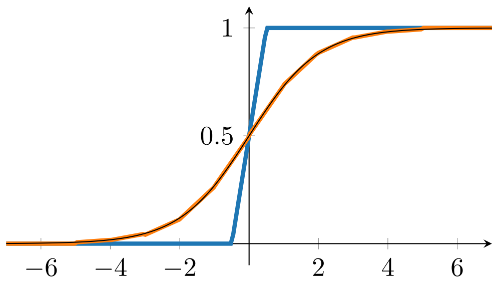

Figure 4. Piecewise linear approximations for the non-linear sigmoid function (in black) from Mohassel & Zhang (2017) in blue and from Faiedh et al. (2001) in orange.

schedule for $1/t$ in the interior point logarithmic barrier method as described by Boyd & Vandenberghe (2004). The weights for the gradients of the regular binary cross entropy loss (BCE) and the loss from the constraint terms (CON) follow the schedules

$$\xi_j^{\mathrm{BCE}} = \frac{N_e}{N_e + j}\,, \qquad \xi_j^{\mathrm{CON}} = \frac{N_e + 10j}{N_e}\,.$$

Weight decay, adaptive learning rate schedules. and momentum neither consistently improved nor impaired training. Therefore, all reported numbers were achieved with vanilla SGD, for fixed learning rates, and without any regularization. After extensive testing on all datasets, we converged to a fixed-point representation with 16 bits for the integer and fractional part respectively. The smaller the number of bits, the faster the MPC implementation and the higher the risk of loss of precision or over- and underflows. We found 16 bits to be the minimally needed precision for all our experiments to work.

C. Additional Experimental Results

C.1. Results on Remaining Datasets

Analogously to Figures 2 and Figure 3 we report the results on test accuracy as well as the mitigation of disparate impact for the Lagrangian multiplier method in Figure 5. In the Adult dataset we are able to mitigate disparate impact with slightly worse accuracy as compared to the baseline. Note that the German dataset contains only 512 training and 200 test examples, which explains the discrete jumps in accuracy in minimal steps of $1/200 = 0.005$. Hence, even though the Lagrangian multiplier technique here consistently removes disparate impact to a similar extent as the baseline, interpretations of results on such small datasets require great care. For the much larger stop, question and frisk dataset we again observe the curious initial increase in accuracy similar to our observations for the Bank dataset. In this dataset about 93% of all samples have positive labels, which explains the near optimal accuracy when collapsing to always predict 1, which happens for the baseline as well for our method at a similar rate as c decreases.

C.2. Disadvantages of Other Optimization Methods

In Section 5 we suggest the Lagrangian multiplier technique for fair model training using fixed-point numbers. Here we substantiate this suggestion with further empirical evidence. Figure 6 shows analogous results to Figure 3 and the second row of Figure 5. These plots reveal the shortcomings of the interior point logarithmic barrier and the projected gradient methods.

Interior Point Logarithmic Barrier method. While the interior point logarithmic barrier method does balance the fractions of people being assigned positive outcomes between the two different demographic groups when the constraint is tightened, it soon breaks down entirely due to overflow and underflow errors. The number of failed runs was substantially higher than for the Lagrangian multiplier technique. As explained in (Boyd & Vandenberghe, 2004), when we increase the parameter t of the interior point logarithmic barrier method during training, the barrier becomes steeper, approaching the function

$$I_-(x) = \begin{cases} 0 & \text{for } x \le 0\,, \\ \infty & \text{for } x > 0\,. \end{cases}$$

From this it becomes obvious that when facing tight constraints, the gradients might change from almost zero to extremely large values within a single update of the parameters $\boldsymbol{\theta}$. Moreover, iplb requires careful tuning and scheduling of t. Hence, the interior point logarithmic barrier method, while achieving good results over some domains, is not well suited for MPC.

Projected gradient method. In Figure 6, we observe that the projected gradient method seems to fail in most cases, since it does not actually balance the fractions of positive outcomes across the sensitive groups. There is a simple explanation why it can satisfy the constraint $\mathbb{F}(\boldsymbol{\theta}) \le 0$ for the $p\%$-rule even with small $\mathbf{c}$ and still retain near optimal accuracy. Note that the accuracy only depends on the direction of $\boldsymbol{\theta}$, i.e., it is invariant to arbitrary rescaling of $\boldsymbol{\theta}$. Since the constraint $\mathbb{F}(\boldsymbol{\theta}) = |\mathbf{A}\boldsymbol{\theta}| - \mathbf{c} \le 0$ is always satisfied for $\boldsymbol{\theta} = 0$, dividing any $\boldsymbol{\theta}$ by a large enough factor will result in a classifier that achieves equal accuracy and satisfies the constraint (by continuity). However, minimizing the loss in the original logistic regression optimization problem (or equivalently maximizing the likelihood), which is not invariant under rescaling of $\boldsymbol{\theta}$, counteracts shrinking $\boldsymbol{\theta}$ as it enforces high confidence of decisions, i.e., large $\boldsymbol{\theta}$. The projection method produces high accuracy classifiers with small weights that formally fulfill the fairness constraint, but do not properly mitigate disparate impact as measured

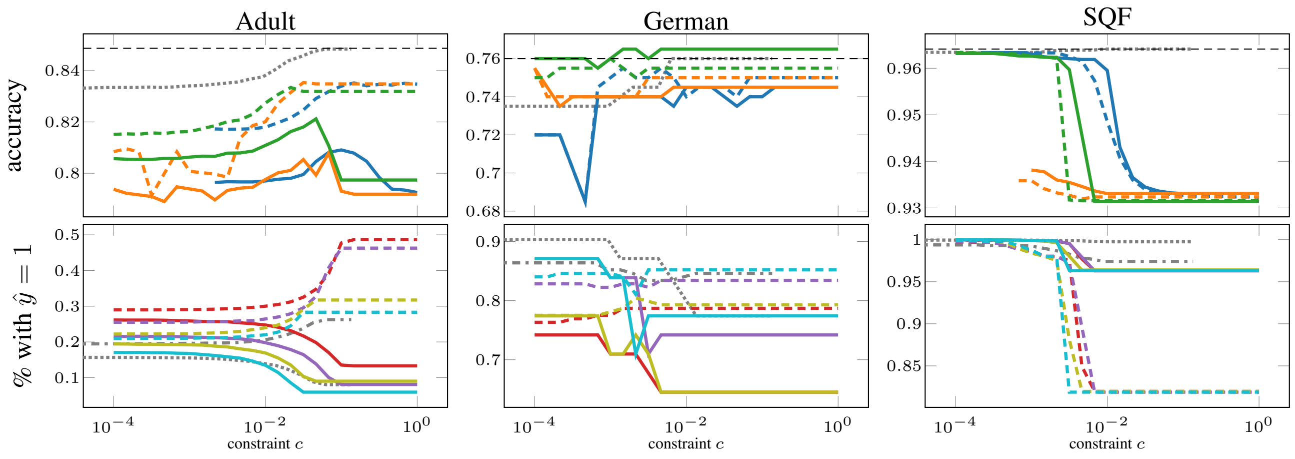

Figure 5. **First row:** The color code is blue: iplb, orange: projected, green: Lagrange with *continuous* lines for no approximation and *dashed* lines for piecewise linear approximation. The gray dotted line is the baseline and the dashed black line marks unconstrained logistic regression. **Second row:** *Continuous/dotted* lines correspond to $z = 0$ and *dashed/dash-dotted* lines to $z = 1$. The color code is (red: no approx. + float, purple: no approx. + fixed, yellow: pw linear + float, turquoise: pw linear + fixed, gray: baseline).

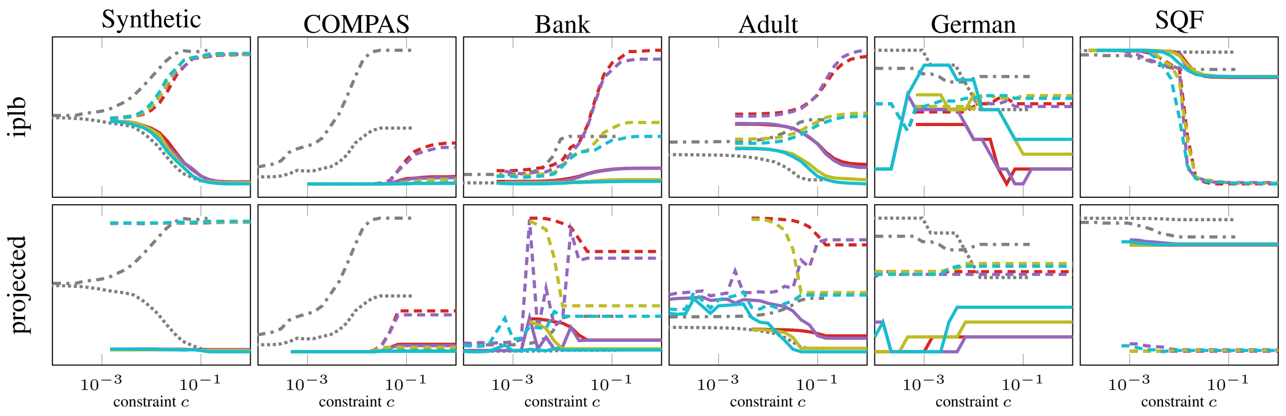

Figure 6. We plot the fraction of people with $z = 0$ (*continuous/dotted*) and with $z = 1$ (*dashed/dash-dotted*) who get assigned positive outcomes over the constraint c for 5 different datasets. The different colors correspond to (red: no approximation + floats, purple: no approximation + fixed-point, yellow: piecewise linear + floats, turquoise: piecewise linear + fixed-point, gray: baseline).

by the true $p\%$-rule instead of the computational proxy. It also often fails for small constraint values, as the projection matrix in eq. (10) turns out to become near singular producing over- and underflow errors.

D. Clarification of Privacy or Secrecy

In this work, privacy or secrecy constraints are separate from other theorized, setup-dependent attacks, e.g., model extraction (Tramèr et al., 2016) or inversion (Fredrikson et al., 2015). If relevant, modelers may need to consider these separately.