A SUPPLEMENTARY MATERIAL

A.1 Working example for generating counterfactuals

We present a simple working example for counterfactual generation. Given the assumptions we undertake for (1) plus the additional assumption of an additive noise model (ANM)—i.e., $\mathcal{S} = \{X_j \leftarrow f_j(X_{pa(j)}) + U_j\}_{j=1}^{p}$—the generating procedure is straightforward. The ANM assumption is also assumed in Section 4 for the classification scenarios. It is a common model specification assumption that allows to identify more easily the non-random parts of the equation. Suppose we have the following structural causal model $\mathcal{M}$ and corresponding directed acyclical graph $\mathcal{G}$:

$$\begin{tikzcd} & X_2 \arrow[dr] & \\ X_1 \arrow[ur] \arrow[rr] & & X_3 \end{tikzcd}$$

$$\mathcal{M} \begin{cases} X_1 & \leftarrow U_1 \\ X_2 & \leftarrow \alpha \cdot X_1 + U_2 \\ X_3 & \leftarrow \beta_1 \cdot X_1 + \beta_2 \cdot X_2 + U_3 \end{cases}$$

where U_1, U_2, U_3 represent the latent variables, X_1, X_2, X_3 the observed variables, and α, β_1, β_2 the coefficient for the causal effect of, respectively, $X_1 \longrightarrow X_2$, $X_1 \longrightarrow X_3$, and $X_2 \longrightarrow X_3$. Suppose we want to generate the counterfactual for X_3, i.e., X_3^{CF}, had X_1 been equal to $x_1 \in X_1$. In the *abduction* step, we estimate U_1, U_2, and U_3 given the evidence or what is observed under the specified structural equations:

$$\begin{aligned} \hat{U}_1 &= X_1 \\ \hat{U}_2 &= X_2 - \alpha \cdot X_1 \\ \hat{U}_3 &= X_3 - \beta_1 \cdot X_1 + \beta_2 \cdot X_2 \end{aligned}$$

We can generalize this step for (1) as $U_j = X_j - f_j(X_{pa(j)})\ \forall X_j \in X$. This step is an individual-level statement on the residual variation under SCM $\mathcal{M}$. It accounts for all that our assignment functions f_j, which are at the population level, cannot explain, or the *error terms*. In the *action* step, we intervene X_1 and set all of its instances equal to x_1 via $do(X_1 := x_1)$ and obtaining the intervened DAG $\mathcal{G}'$ and SCM $\mathcal{M}'$:

$$\begin{tikzcd} & X_2 \arrow[dr] & \\ do(x_1) & & X_3 \end{tikzcd}$$

$$\mathcal{M}' \begin{cases} X_1 & = x_1 \\ X_2 & \leftarrow \alpha \cdot x_1 + U_2 \\ X_3 & \leftarrow \beta_1 \cdot x_1 + \beta_2 \cdot X_2 + U_3 \end{cases}$$

where no edges come out from X_1 as it has been fixed to x_1. Finally, in the *prediction* step, we combine these two steps to calculate X_3^{CF} under the set of $\hat{U}$ and the intervened $\mathcal{M}'$:

$$\begin{aligned} X_3^{CF} &\leftarrow \beta_1 \cdot x_1 + \beta_2 \cdot X_2 + \hat{U}_3 \\ &\leftarrow \beta_1 \cdot x_1 + \beta_2 \cdot (\alpha \cdot x_1 + \hat{U}_2) + \hat{U}_3 \end{aligned}$$

which is done for all instances in X_3. This is what is done at a larger scale, for example, in [33] and [47], and also in this paper. The same three steps can apply to X_2 (also for X_1, though it would be trivial as it is a root note).

We can view this approach as a *frequentist*[10] one for generating counterfactuals, in particular, with regard to the Abduction step. A more *Bayesian* approach is what is done by [39] where they use a Monte Carlo Markov Chain (MCMC) to draw $\hat{U}$ by updating its prior distribution with the evidence X to then proceed with the other two steps. In

[10]This is not a formal distinction, but based on talks with other researchers in counterfactual generation. Such a distinction, to the best of our knowledge, remains an open question.

Section 4.2, we used both approaches for generating the counterfactuals and found no difference in the results. We only present in this paper the first approach as it is less computationally expensive.

A.2 Sketch of Proof for Proposition 3.6

Consider the factual tuple $(x_c, a_c = 1, \widehat{y}_c = 0)$ and assume the generated counterfactual is $(x_c^{CF}, a_c^{CF} = 0, \widehat{y}_c^{CF} = 0)$. Since $\widehat{y}_c = \widehat{y}_c^{CF}$, this is a case where counterfactual fairness holds. However, the decision boundary of the model $b()$ can be purposely set such that the k-nearest neighbors of x_c are all within the decision $\hat{Y} = 0$, and less than $1 - \tau$ fraction of the k-nearest neighbors of x_c^{CF} are within the decision $\hat{Y} = 0$. This leads to a $\Delta p > 1 - (1 - \tau) = \tau$, showing that there is individual discrimination. The other way can be shown similarly by assuming $\widehat{y}_c \neq \widehat{y}_c^{CF}$ but the sets of k-nearest neighbors have rates of negative decisions whose difference is lower than τ.

B ALGORITHMS FOR K-NN CST IMPLEMENTATION

We present the relevant algorithms for the k-NN CST implementation (Section 3.4). The algorithm 1 performs CST while algorithm 2 returns the indices of the top-k tuples with respect to the search centers based on the distance function d. Notice that the main difference in algorithm 1 when creating the neighborhoods is that the search centers are drawn from the factual dataset for the control group $\mathcal{D}$ and the counterfactual dataset $\mathcal{D}^{CF}$ for the test group. Further, notice that we use the same c (i.e., index) for both as these two data-frames have the same structure by construction.

Algorithm 1: run_CST

```
Input  : $\mathcal{D}$, $\mathcal{D}^{CF}$, $k$
Output: $[p_c - p_t]$
$prot\_condition \leftarrow \mathcal{D}[:, \ prot\_attribute] == prot\_value$
$\mathcal{D}_c \leftarrow \mathcal{D}[prot\_condition]$                        // get protected (control) search space
$\mathcal{D}_t \leftarrow \mathcal{D}[\neg \ prot\_condition]$                     // get non-protected (test) search space
$prot\_idx \leftarrow \mathcal{D}_c.index.to\_list(\ )$;                      // get idx for all complainants
$diff\_list = [\ ]$
for $c, \ row \in prot\_idx$ do
    $res\_1 \leftarrow get\_top\_k(\mathcal{D} \ [c, \ :], \mathcal{D}_c, k)$;         // idx of the top-k tuples for control group
    $res\_2 \leftarrow get\_top\_k(\mathcal{D}^{CF}[c, \ :], \mathcal{D}_t, k)$;          // idx of the top-k tuples for test group
    $p_c \leftarrow sum(\mathcal{D}[res_1, \ target\_attribute] == negative\_outcome) \ / \ len(res\_1)$
    $p_t \leftarrow sum(\mathcal{D}[res_2, \ target\_attribute] == negative\_outcome) \ / \ len(res\_2)$
    $diff\_list[c] \leftarrow p_c - p_t$
end
return $diff\_list$
```

Algorithm 2: get_top_k

```
Input  : $t, t\_set, k$
Output: $[indices]$
$(idx, dist) \leftarrow k\_NN(t, t\_set, k + 1)$;                    // run k-NN algorithm with $k + 1$
if without search centers then
    $remove(t, idx, dist)$;                                // remove the center t from idx
end
$idx' \leftarrow sort(idx, dist)$;                                 // sort idx by the distance
return $idx'$
```

Received 10 May 2023; revised 5 September 2023; accepted 20 September 2023