Counterfactual Explanations and the CARLA Library

Simon Frank
University of Tuebingen

David Ott
University of Tuebingen

***Abstract*—We implemented the Adversarially Robust Algorithmic Recourse (ARAR) algorithm from Dominguez-Olmedo et al. [1] in the Counterfactual And Recourse Library (CARLA) [2], including testing and a small demonstration. We discuss the notion of robust counterfactuals and examined the ARAR algorithm on its robustness in some experiments, comparing different hyperparameter settings and comparing it to the method described in Wachter et al. [3].**

I. Introduction

There is an increase in the use of machine learning (ML) classifiers for consequential decision-making in sensitive domains such as loan approval. Many of these ML classifiers behave like a black-box for the user, but algorithmic recourse methods have been developed to guarantee human agency, which wants to provide individuals with actionable recommendations to reverse unfavorable automated decisions. An emerging issue in the use of recourse methods is how to deal with unpredictable changes. Previous works argued that the decision-maker must reverse the unfavorable decision, despite unanticipated changes, to maintain trust [3, 4, 5]. In contrast, Dominguez-Olmedo et al. [1] argue that the recourse methods must be robust to plausible uncertainties arising in the recourse process.

We will illustrate the notion of a robust recourse action with a concrete example. Suppose a bank has denied a customer's loan, and counterfactual recourse has shown that he would be approved a loan if his savings were higher. The customer has now increased his savings. However, due to unforeseen circumstances, his weekly working hours and thus his salary is slightly reduced and now the decision-making classifier will not approve the loan. If the loan is approved anyway, to shield the recourse action against uncertainty *ex-post*, this could have serious consequences for both the bank (e.g., monetary loss) and the client (e.g., bankruptcy and inability to secure future loans). If the recourse promise is broken, this would lead to a loss of confidence in the decision-maker. Therefore it is necessary to ensure that recourse recommendations are *ex-ante* robust to uncertainty. This example motivates the notion of robustness in counterfactuals. A counterfactual generated by a recourse method should still be classified as the desired class, even when small uncertain perturbations are applied to it.

While the previous recourse algorithms (e.g. Wachter et al. [3]) are providing recourse actions that are only valid for the individual x, the recommendations provided by the adapted algorithm of Dominguez-Olmedo et al. [1] remain valid for *all* plausible individuals in the uncertainty set $B(x)$, as illustrated in Figure 1.

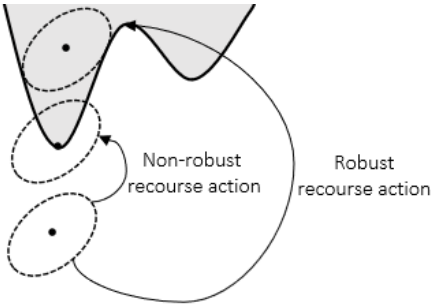

Fig. 1. An adversarially robust recourse actions must ensure a positive classification outcome for all individuals in the uncertainty set around the individual x. The grey area in the figure represents the portion of the input space that is classified as the target class, while the white area is classified as the negative class.

In this work the Adversarially Robust Algorithmic Recourse (ARAR) algorithm [1] is integrated into the CARLA library [2], a software project, in which many state-of-the-art counterfactual and recourse algorithms are brought together.

II. Implementation

The uncertainty surrounding an observation x is modelled here by an ϵ -ball of uncertainty $B(x) = \{x + \Delta | \|\Delta\| \leq \epsilon\}$ around the observed x. The norm $\|.\|$ characterizes some relevant notion of magnitude for perturbations θ to the observation x and ϵ specifies the amount of uncertainty under consideration [6][7].

A recourse action a is then adversarially robust for an individual $x \in X$ and classifier $h : \mathcal{X} \to [0, 1]$, if and only if it is valid for all individuals x' in the uncertainty set $B(x)$, i.e.

$$h(x' + a) > 0.5 \quad \forall x' \in B(x) \tag{1}$$

For a differential classifier h the adversarially robust recourse problem is equivalent to the following unconstrained problem, where l is the binary cross-entropy loss and c the cost function, which models the effort required by an individual $x \in X$ to implement some recourse action a, and $F(x)$ is the feasibility set:

$$\min_{a \in F(x)} \max_{\lambda \geq 0} c(x, a) + \lambda \Big(\max_{x' \in B(x)} l\big(h(x' + a), 1\big) \Big) \tag{2}$$

Algorithm 1 Generating adversarially robust recourse for a differentiable classifier h

Require: Factual individual x, size of the uncertainty ball ϵ, weight of the cost $\lambda > 0$, decay rate $\gamma < 1$, actions $\theta = 0$, learning rate α, differentiable classifier h

Output: θ adversarially robust counterfactual actions

1: $pertb_x \leftarrow 0$
2: **while** $N \leq N_{max}$ **do**
3: **while** $M \leq M_{max}$ **do**
4: $pertb \leftarrow \nabla_{pertb_x} l(h(x + \theta + pertb_x), 1)$
5: $pertb \leftarrow \frac{g}{\|g\|_2} \cdot \epsilon$
6: **if** use PGD **then**
7: **for** 10 iterations **do**
8: $grad \leftarrow \nabla_{pertb} l(h(x + \theta + pertb), 1)$
9: $pertb \leftarrow g + \alpha \cdot grad$
10: $pertb \leftarrow \frac{g}{\|g\|_2} \cdot \epsilon$
11: **end for**
12: **end if**
13: $x^* \leftarrow x + pertb$
14: Reconstruct constraints on x^*
15: **if** $h(x^* + \theta) > 0.5$ **then**
16: **return** θ
17: **end if**
18: $g \leftarrow \nabla_\theta l(h(x^* + \theta), 1) + \lambda \|\theta\|_1$
19: $\theta \leftarrow Proj_{F(x)}(\theta - \alpha g)$
20: **end while**
21: $\lambda \leftarrow \gamma\lambda$
22: **end while**

Our implementation to solve this problem is described in Algorithm 1. In the following we will discuss some details of this implementation.

The inner maximization over the ball $B(x)$ in general is a non-convex intractable optimization problem. However it can approximately be solved using a first-order approximation (line 4) and normalizing the calculated perturbations $pertb$ to the size provided by the user ϵ (line 5), so that the user can define the magnitude of uncertainty. Even more robust, if chosen by the user, the first-order approximation is used as a starting point for Projected Gradient Descent (PGD) using gradient ascent to find the maximum (lines 6-12). It has to be stressed, that the local maximum x^* (line 13) found by the algorithm may not be the global maximum in $B(x)$, since this inner maximization problem is in general non-convex. Then the exit condition $h(x^* + \theta) > 0.5$ does not imply that $h(x' + \theta) > 0.5 \ \forall x' \in B(x)$. In the case of a linear model h, this optimization does find the global optimum, so the aforementioned implication holds. However, as discussed in Sections III. Experiments and IV. Results, we find that our implementation is effective in generating robust recourse actions, even for a non-linear model h. Without any uncertainty in the features of the individual x, that is $B(x) = \{x\}$ or equivalently $\epsilon = 0$, the optimization procedure is exactly the same as the one describe in Wachter et al. [3] since the inner maximization has the trivial solution $x^* = x$.

To ensure that the local maximum x^* is a valid factual, the constraints given by the dataset have to be reconstructed on x^* (line 14). In code this amounts to setting the floating point results of the optimization procedure to the closest value for the categorical variable.

To solve the outer minimax optimization problem we use projected gradient ascent over the recourse action θ and feasibility set $F(x)$. The maximization over λ in Equation 2 is intuitively similar to a loosening of the cost constraints throughout the counterfactual search procedure. In practice this is realized by decreasing a λ which is multiplied to the cost, which is equivalent (lines 18 - 21). We used the L1-norm in our implementation as the cost function. The decrease of λ thereby places growing emphasis in crossing the classifier's decision boundary, while loosening the cost constraints.

In Algorithm 1 ARAR is described for the target class 1, however, in our implementation, the user can choose between the target class 0 or 1. Since the CARLA library is written in python and is using pytorch as a backend, we implemented the algorithm in python using pytorch as well.

Algorithm 1 describes how to find an adversarially robust counterfactual for one factual individual, however, it is easily parallelizable. In our implementation the counterfactuals for all factuals are calculated in one batch in parallel. As soon as a counterfactual is found the corresponding factual is masked and thereby not used for further calculations. Through this technique, we were able to achieve a massive speed improvement, when dealing with big data sets compared to the Wachter implementation in CARLA.

A. Testing

We implemented some tests to ensure valid results for our ARAR implemenation in the CARLA library. The first test is checking the structure of the provided output. It checks whether the output is a panda dataframe, the order of the columns, and the size of the output and input panda dataframe are the same. The second test checks if the label is flipped for all the returned counterfactuals, so it checks that the technique does in fact return valid counterfactuals. We are testing for both target classes 0 and 1, and for the Linear Regression (Linear) and Artificial Neural Network (ANN) model on the Adult dataset [8] provided in CARLA.

B. Working with CARLA

In the following we will discuss some challenges and bottlenecks that we encountered when integrating our code into CARLA in a concise form:

- The provided installation instructions (shown in the Appendix A) did not reliably work on Windows. For example, installing with these commands did not work due to a broken wheel for pytorch. We settled to fully manually install all packages (specific commands are shown in the Appendix A).
- There are some problems from CARLA using packages in a non up-to-date version. For example installing the torch wheel on windows is not possible for this reason.

Further, on windows most tests do not work due to the model loading for tensorflow models not working because of h5py not working. This is probably due to it not being up-to-date, as the tests of the h5py library fail (h5py.run_tests()) but they do not when installing only a current version of h5py. Similarly, on linux, torch does not work, because the old version of torch does not support a current version of CUDA. In general the dependencies should be updated to their current versions.
- The examples provided in the tutorial notebooks in the documentation (see the specific links in the Appendix B) did not fully run due to API changes. Some minor changes were necessary for all these examples to run them. This was particularily challenging when getting into the structure of the library.
- In the README.md the link to the Plotting tutorial notebook is broken.
- Most recourse methods require hyperparameters, which are not readily provided. It would be useful if each recourse method had a set of default hyperparameters, which would allow the user to initialize them with hyperparameters=None to improve ease of use.
- The default logging behavior for some methods (e.g. Wachter) is very verbose. The default logging level should probably not be set to INFO, but to WARNING. Alternatively the logging convention of the library should be explained relatively early in the tutorials.
- The provided ANN models for the Give me Some Credit (GMSC) [9] and HELOC [10] dataset with the pytorch backend do not appear to work. They both return a constant value, regardless of the input factual.

The general structuring of CARLA into a Dataset Catalog (OnlineCatalog), MLModelCatalog and recourse_catalog is sensible. Implementing the recourse method in this structure was very useful, since as soon as everything was CARLA-API-compliant, it was easy to run experiments. This is especially the case, as in our situation all the data handling and model training, not specifically part of the recourse method, was already solved. Similarly we would expect it to be easy to implement and train a different ML model, and then have recourse methods readily available.

III. Experiments

To examine the behavior of our implementation of the ARAR algorithm for different hyperparameters and to compare it to the Wachter algorithm [3] we conducted some experiments.

We conducted all our experiments for the ANN and the Linear model already implemented and pretrained in the CARLA library [2], for all datasets provided: Adult [8], COMPAS [11], Give Me Some Credit (GMSC) [9] and HELOC [10]. For our evaluation we omitted the ANN model on the GMSC and HELOC dataset, as the model only returned a constant value for any input on these datasets.

First of all we generated factuals in the datasets with the provided models that were classified as the negative class. Then we generated counterfactuals for each method and hyperparameter setting discussed here. Further we tested, whether these counterfactuals were actually classified as the target class and report the proportion of found counterfactuals relative to the total number of factuals that were provided.

To examine the robustness of the generated counterfactuals, we conducted two types of experiments:

1) Finding perturbations in the ϵ-ball via gradient ascent: We try to find a perturbation in the $\epsilon = 0.1$-ball around the counterfactual which is still classified as the negative class. This is done by optimizing a perturbation with its norm smaller than $\epsilon = 0.1$ which is added to the counterfactual via gradient ascent, such that it maximizes the models score for the negative class. We report the proportion of classifications that are still classified as the target class, even after adding the perturbation found with the gradient.

2) Generating random perturbations in the ϵ-ball: We generate multiple random perturbations ($N = 1000$) in the $\epsilon = 0.1$-ball and add them to the counterfactuals. Then we examine the model classfications for these randomly perturbed counterfactuals. We report the proportion of classifications that are still the target class, after random perturbation.

For ARAR we conducted the experiments on the whole dataset for all conditions. For Wachter we only examined a maximum of 500 factuals for each dataset due to computational/runtime constraints (see Section IV-A2). We report the hyperparemeters that were used for the experiments in Appendix C

IV. Results

First of all we can observe that both, ARAR and Wachter are capable of generating counterfactuals that are classified as the target class (see Fig. 2). For the Linear model counterfactuals are found for almost all factuals by all methods, which is unsurprising, as both methods use the gradient to obtain the counterfactuals and in the case of the Linear model this is a convex problem. So if there exists some sample in the input space that is classified as the target class, then it will be found by the gradient using both methods. For the ANN model however, not for all factuals a counterfactual could be found with these methods. This is due to the non-convex nature of the problem, where the optimization gets stuck in a local optimum.

We can further observe, that the ARAR methods with a larger ϵ find fewer counterfactuals for the ANN model than the ARAR methods with a smaller ϵ. This is to be expected, as by introducing the constraint of having the model classify as the target even in the ϵ-ball around the counterfactual, the problem becomes much harder or even impossible to solve.

A. Comparing ARAR and Wachter

We can verify the key expected finding, which is, that ARAR with $\epsilon > 0$ is more robust than Wachter. This can be seen in Figures 3 and 4, as the proportions of counterfactuals

	Model: ANN		Model: Linear			
	Adult	COMPAS	Adult	COMPAS	GMSC	HELOC
ARAR, $\epsilon = 0.1, \alpha = 0.1$	0.007	0.005	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.1$, with pgd, $\alpha = 0.1$	0.007	0.005	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.05, \alpha = 0.1$	0.179	0.398	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.05$, with pgd, $\alpha = 0.1$	0.179	0.398	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.0, \alpha = 0.1$	0.387	0.658	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.1, \alpha = 0.01$	0.007	0.005	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.1$, with pgd, $\alpha = 0.01$	0.007	0.005	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.05, \alpha = 0.01$	0.179	0.398	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.05$, with pgd, $\alpha = 0.01$	0.179	0.398	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.0, \alpha = 0.01$	0.387	0.658	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.0, \alpha = 0.01$, like Wachter	0.387	0.658	1.0	1.0	1.0	1.0
Wachter	0.406	0.644	0.998	1.0	1.0	1.0

Fig. 2. Proportion of counterfactuals found from the total number of factuals provided for each method, i.e. $\frac{\text{\# counterfactuals found}}{\text{total number of factuals}}$.

	Model: ANN		Model: Linear			
	Adult	COMPAS	Adult	COMPAS	GMSC	HELOC
ARAR, $\epsilon = 0.1, \alpha = 0.1$	1.0	1.0	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.1$, with pgd, $\alpha = 0.1$	1.0	1.0	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.05, \alpha = 0.1$	0.955	1.0	0.379	0.577	0.755	0.917
ARAR, $\epsilon = 0.05$, with pgd, $\alpha = 0.1$	0.955	1.0	0.380	0.577	0.754	0.916
ARAR, $\epsilon = 0.0, \alpha = 0.1$	0.502	0.899	0.201	0.577	0.739	0.911
ARAR, $\epsilon = 0.1, \alpha = 0.01$	1.0	1.0	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.1$, with pgd, $\alpha = 0.01$	1.0	1.0	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.05, \alpha = 0.01$	0.0	0.0	0.0	0.0	0.0	0.0
ARAR, $\epsilon = 0.05$, with pgd, $\alpha = 0.01$	0.0	0.0	0.0	0.0	0.0	0.0
ARAR, $\epsilon = 0.0, \alpha = 0.01$	0.0	0.0	0.0	0.0	0.0	0.0
ARAR, $\epsilon = 0.0, \alpha = 0.01$, like Wachter	0.0	0.0	0.0	0.0	0.0	0.0
Wachter	0.0	0.0	0.0	0.0	0.0	0.0

Fig. 3. Proportion of predictions classified as target after addition of a perturbation adversarially generated with the gradient in the $\epsilon = 0.1$-ball.

	Model: ANN		Model: Linear			
	Adult	COMPAS	Adult	COMPAS	GMSC	HELOC
ARAR, $\epsilon = 0.1, \alpha = 0.1$	1.0	1.0	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.1$, with pgd, $\alpha = 0.1$	1.0	1.0	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.05, \alpha = 0.1$	0.999	1.0	0.994	0.973	0.997	0.999
ARAR, $\epsilon = 0.05$, with pgd, $\alpha = 0.1$	0.999	1.0	0.994	0.973	0.997	0.999
ARAR, $\epsilon = 0.0, \alpha = 0.1$	0.991	0.999	0.846	0.899	0.959	0.995
ARAR, $\epsilon = 0.1, \alpha = 0.01$	1.0	1.0	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.1$, with pgd, $\alpha = 0.01$	1.0	1.0	1.0	1.0	1.0	1.0
ARAR, $\epsilon = 0.05, \alpha = 0.01$	0.975	0.902	0.974	0.895	0.950	0.993
ARAR, $\epsilon = 0.05$, with pgd, $\alpha = 0.01$	0.975	0.903	0.974	0.896	0.950	0.993
ARAR, $\epsilon = 0.0, \alpha = 0.01$	0.552	0.537	0.569	0.519	0.546	0.565
ARAR, $\epsilon = 0.0, \alpha = 0.01$, like Wachter	0.606	0.590	0.636	0.594	0.640	0.786
Wachter	0.597	0.592	0.615	0.594	0.644	0.783

Fig. 4. Proportion of classifications to target after addition of random perturbations in the $\epsilon = 0.1$-ball.

perturbed in the ϵ-ball with the gradient or randomly, that are still classified as the positive class, are both higher for ARAR with $\epsilon > 0$ than for Wachter, on all datasets and models.

Further, as we examine the robustness in the $\epsilon = 0.1$-ball, we can see that ARAR with $\epsilon = 0.1$ is fully robust here, as all perturbed counterfactuals are still classified as the target class (denoted by a 1.0 in the Figures 3 and 4). This is somewhat surprising as we have no theoretical guarantee on the robustness of the generated counterfactuals for the ANN model, due to the non-convexity of the inner maximization problem. However, experimentally we observe that the method is fully robust. Further we can see that ARAR with $\epsilon = 0.1$ is more robust than ARAR with $\epsilon = 0.05$, which is more robust than ARAR with $\epsilon = 0.0$, which was to be expected.

When examining Figure 4 it should be noted, that one would expect a value of around 0.5 for the proportions as a baseline for counterfactuals that are perfectly on decision boundary of the model, assuming the decision boundary can be roughly approximated linearly. If this is the case, then one would expect points randomly sampled on the ϵ-ball around these counterfactuals to fall inside the decision boundary in roughly 50% of the cases. Finding such counterfactuals on the decision boundary should be the case for ARAR with $\epsilon = 0$ and for Wachter. Surprisingly, both obtain proportions slightly better than 0.5, this is probably due to the gradient procedure finding slightly more optimal solutions. Even more surprising, ARAR with $\epsilon = 0.0$ and a larger learning rate of 0.1 is much more robust, this is further discussed in section IV-C

1) Relation of ARAR with $\epsilon = 0$ and Wachter: Regarding hyperparameters, the default settings of Wachter in CARLA and of ARAR in [1] are a little different. When adjusting the settings of ARAR to match those of Wachter (setting the $\alpha = 0.01$ and using no l1-regularization on the perturbation by setting $\lambda = 0$), we can observe that ARAR with $\epsilon = 0$ behaves the same as Wachter (see Figures 2, 3 and 4). All remaining differences are due to only using a portion of the factuals in the evaluation of Wachter and randomness in the sampling of the perturbations. This experimentally confirms the theoretical finding stated in [1], that ARAR with $\epsilon = 0$ is equal to Wachter. As this is the case, if using CARLA one should use ARAR with hyperparameters $\epsilon = 0$, $\lambda = 0$ and $\alpha = 0.01$ instead of Wachter due to speed.

2) Runtime: Our implementation of ARAR is orders of magnitude faster than the implementation of Wachter in CARLA. This is due to ours being implemented in a parallel fashion for all provided factuals, while the implementation of Wachter generates counterfactuals for each factual in succession. For this reason, the runtime for our ARAR implementation for a specific model is almost constant, regardless of the number of factuals/counterfactuals to be generated, as long as the computations fit in memory. For example for the ANN model on the Adult dataset, our ARAR implementation needs about 1:40 minutes to generate counterfactuals for the whole dataset (or about 10 minutes when using PGD for the inner maximization), while the Wachter implementation would need multiple days, with default hyperparameter settings. Note that this is heavily dependant on the maximum computation time set for each counterfactual.

B. Effect of using PGD

Further we compare the use of PGD against naive gradient ascent for the inner maximization in ARAR. As can be seen in Figures 2, 3 and 4, ARAR with and without PGD for the inner maximization produces almost the same results. So at least for our models and datasets there is no advantage in using PGD, in particular as it has a larger compute demand. However, this could be due to the simplicity of the ANN model. In general, PGD is more robust for finding adversarial examples [12], so one could expect this also holds for finding adversarial examples in the ϵ-ball around the counterfactuals for more complicated models.

C. Effect of the learning rate α

One interesting observation that can be made is, that some of the robustness can be obtained by using a higher learning rate. To see this one can compare ARAR with learning rate 0.1 to ARAR with learning rate 0.01 for the different hyperparameters within Figure 3 and Figure 4 respectively. A larger portion of the counterfactuals generated with ARAR with the learning rate $\alpha = 0.1$ is still classified as the target class, even after perturbation generated with the gradient and random perturbation. For counterfactuals generated with $\epsilon < 0.1$ and learning rate $\alpha = 0.01$ however, perturbations that are classified as the negative class can be found in all cases with the gradient (the reported proportion is 0.0 in Figure 3.

The robustness from the learning rate even holds for ARAR with $\epsilon = 0$, which naively should not be robust at all. This is probably due to gradient descent with the higher learning rate overshooting the models decision boundary, and thereby obtaining robustness as a side product.

V. Conclusion

We made a substantial contribution to the CARLA library by implementing the ARAR algorithm and discussed some of the problems and advantages arising during the implementation by using CARLA. Further we were able to experimentally verify the robustness of ARAR, in particular in comparison to Wachter [3]. We conducted experiments to further examine the effect of different hyperparameters to understand the inner workings of our ARAR implementation but also of the ARAR algorithm in general. We expect that this *ex-ante* approach to robustness will be important in the future of recourse methods. Further there are some interesting intricacies, e.g. regarding the learning rate α, so these methods have to be investigated further.

REFERENCES

[1] R. Dominguez-Olmedo, A. H. Karimi, and B. Schölkopf, "On the adversarial robustness of causal algorithmic recourse," in *International Conference on Machine Learning*. PMLR, 2022, pp. 5324–5342.

[2] M. Pawelczyk, S. Bielawski, J. van den Heuvel, T. Richter, and G. Kasneci, "Carla: A python library to benchmark algorithmic recourse and counterfactual explanation algorithms," 2021.

[3] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual explanations without opening the black box: Automated decisions and the gdpr," *Harv. JL & Tech.*, vol. 31, p. 841, 2017.

[4] S. Venkatasubramanian and M. Alfano, "The philosophical basis of algorithmic recourse," in *Proceedings of the 2020 conference on fairness, accountability, and transparency*, 2020, pp. 284–293.

[5] A.-H. Karimi, G. Barthe, B. Schölkopf, and I. Valera, "A survey of algorithmic recourse: definitions, formulations, solutions, and prospects," *arXiv preprint arXiv:2010.04050*, 2020.

[6] D. Bertsimas, J. Dunn, C. Pawlowski, and Y. D. Zhuo, "Robust classification," *INFORMS Journal on Optimization*, vol. 1, no. 1, pp. 2–34, 2019.

[7] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *stat*, vol. 1050, p. 9, 2017.

[8] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[9] W. C. Credit Fusion, "Give me some credit," 2011. [Online]. Available: https://kaggle.com/competitions/GiveMeSomeCredit

[10] FICO, "FICO xML Challenge," 2018. [Online]. Available: https://community.fico.com/s/explainable-machine-learning-challenge

[11] J. Angwin, J. Larson, S. Mattu, and L. Kirchner, "Machine Bias: There's software used across the country to predict future criminals. And it's biased against blacks." *ProPublica*, 2016. [Online]. Available: https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

[12] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

APPENDIX A
INSTALLING AN ENVIRONMENT WITH ALL PACKAGES REQUIRED FOR CARLA

Installing the packages required by CARLA with the following commands did not work on Windows:

```
pip install -U pip setuptools wheel
pip install -e .
```

Therefore installed an environment with these commands:

```
conda create -n "CARLA-dev" python==3.7 pre-commit==2.9.2 pytest==6.1.2 \
    sphinx==4.0.2 numpydoc==1.1.0 imageio==2.9.0 ipython==7.22.0 jinja2==2.11.3 \
    networkx==2.5.1 scipy==1.6.2 markupsafe==2.0.1 protobuf<=3.21 lime==0.2.0.1 \
    numpy==1.19.4 pandas==1.1.4 scikit-learn==0.23.2 tensorflow==1.14.0 \
    pytorch==1.7.0 torchvision==0.8.1 h5py==2.10.0 xgboost==1.4.2 ipykernel \
    pylint jupyter -c conda-forge -c pytorch
conda activate CARLA-dev
pip install sphinx-rtd-theme==0.5.2 sphinx_autodoc_typehints==1.12.0 mip==1.12.0 \
    recourse==1.0.0 dice-ml==0.5 causalgraphicalmodels==0.0.4 keras==2.3.0
```

APPENDIX B
LINKS TO EXAMPLE NOTEBOOKS THAT DID NOT WORK WITHOUT MODIFICATIONS

- https://carla-counterfactual-and-recourse-library.readthedocs.io/en/latest/notebooks/how_to_use_carla.html
- https://carla-counterfactual-and-recourse-library.readthedocs.io/en/latest/notebooks/how_to_use_carla_causal.html
- https://carla-counterfactual-and-recourse-library.readthedocs.io/en/latest/notebooks/plotting_example.html
- https://carla-counterfactual-and-recourse-library.readthedocs.io/en/latest/notebooks/benchmark_example.html
- https://carla-counterfactual-and-recourse-library.readthedocs.io/en/latest/examples.html

APPENDIX C
HYPERPARAMTER CHOICES IN THE EXPERIMENTS

For ARAR we used the same hyperparameters as in Dominguez-Olmedo et al. [1] if not otherwise specified. These are:

- $\alpha = 0.1$
- $\lambda_{\text{init}} = 1.0$
- decay rate $= 0.9$
- outer iterations $= 100$
- inner iterations $= 50$
- inner maximization with PGD = False
- early stopping = False

For ARAR we varied the $\epsilon \in [0.0, 0.05, 0.1]$, the learning rate $\alpha \in [0.01, 0.1]$ and whether the inner maximization was done with or without PGD.

For Wachter we used the default hyperparameters specified in the CARLA library:

- $\alpha = 0.01$
- $\lambda = 0.01$
- number of iterations $= 1000$.

However, we limited the computation time per counterfactual to 0.01 minutes due to computational constraints. This resulted in no noticeable performance reduction in preliminary experiments.