RobustX: Robust Counterfactual Explanations Made Easy

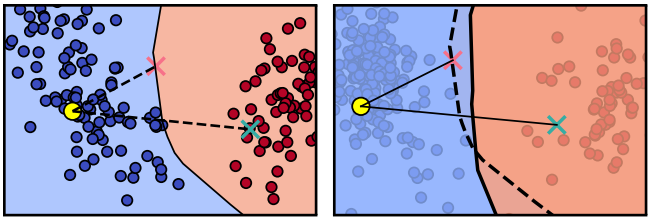
Junqi Jiang[1] , **Luca Marzari**[2] , **Aaryan Purohit**[1] and **Francesco Leofante**[1]
[1] Department of Computing, Imperial College London, United Kingdom
[2]University of Verona, Department of Computer Science, Italy
{junqi.jiang20, francesco.leofante}@imperial.ac.uk, luca.marzari@univr.it

Abstract

The increasing use of Machine Learning (ML) models to aid decision-making in high-stakes industries demands explainability to facilitate trust. Counterfactual Explanations (CEs) are ideally suited for this, as they can offer insights into the predictions of an ML model by illustrating how changes in its input data may lead to different outcomes. However, for CEs to realise their explanatory potential, significant challenges remain in ensuring their robustness under slight changes in the scenario being explained. Despite the widespread recognition of CEs' robustness as a fundamental requirement, a lack of standardised tools and benchmarks hinders a comprehensive and effective comparison of robust CE generation methods. In this paper, we introduce RobustX, an open-source Python library implementing a collection of CE generation and evaluation methods, with a focus on the robustness property. RobustX provides interfaces to several existing methods from the literature, enabling streamlined access to state-of-the-art techniques. The library is also easily extensible, allowing fast prototyping of novel robust CE generation and evaluation methods.

1 Introduction

With the increasing use of Machine Learning (ML) models to aid decision-making in high-stakes fields such as healthcare [Shaheen, 2021] and finance [Cao, 2022], there is a growing need for better explainability of these models. Counterfactual Explanations (CEs) [Guidotti, 2024] are often leveraged in Explainable AI (XAI) to this end due to their intelligibility and alignment with human reasoning [Miller, 2019; Byrne, 2019]. In particular, CEs can offer insights into the predictions produced by an ML model by showing how small changes in its input may lead to different (often more desirable) outcomes. To see what benefits CEs can bring, consider an illustration of a loan application with features *27* years of age, *low* credit rating, and *15K* loan amount. Assume a bank's ML model classifies the application as not creditworthy. A CE for this outcome could be an altered input where a *medium* credit rating (with the other features unchanged) would result in the application being classified as creditworthy, thus giving the applicant an idea of what is required to have their loan approved.



(a) (b)

Figure 1: A lack of robustness may invalidate CEs, here demonstrated on a neural network classifier trained to solve a binary classification task. An input (yellow circle) receives an initial classification, and two counterfactuals (red and green crosses) are generated for it (Figure 1a). After a fine-tuning step occurs (Figure 1b), the decision boundary slightly changes (from dashed to full black line), and previously generated CEs may be invalidated (red cross).

Despite their potential, current approaches to generating CEs often fall short in generating *robust explanations*. Consequently, these methods may produce explanations whose validity is compromised by slight changes in the scenario being explained. For instance, recent work [Upadhyay *et al.*, 2021; Jiang *et al.*, 2023b; Hamman *et al.*, 2023] has highlighted that even small alterations in the parameters of an ML model, e.g. following fine-tuning, may invalidate previously generated CEs. An example of this scenario is captured in Figure 1, where a lack of robustness is demonstrated on a model trained for binary classification tasks. In Figure 1a, an input (yellow circle) receives an initial classification (blue class), and two counterfactuals are generated for it: one (red cross) laying exactly on the decision boundary (full black line) and one deeper inside the counterfactual class (green cross). In Figure 1b, we observe that the decision boundary of the model undergoes slight changes, induced by fine-tuning on a slightly shifted input distribution. As a result, previously generated CEs may cease to be valid if no precautions are taken to ensure robustness. For instance, we observe that the CE corresponding to the red cross is now classified as belonging to the blue class and is thus invalid. Now consider the consequences of these behaviours in our loan example: after fine-

tuning, the applicant changing their credit rating to medium no longer ensures the success of the loan application, as the CE was not robust. When this happens, the CE previously generated by the bank is invalidated, and the bank may be liable for inconsistent statements made to customers regarding loan terms.

This and many other forms of robustness of CEs have been the subject of intense research efforts recently, and numerous algorithms to evaluate the robustness of CEs have been proposed (a recent survey identified about 40 methods [Jiang *et al.*, 2024c]). However, the current state of robust CE research is fragmented, with various methods developed independently and implemented in different, often incompatible, ways. This lack of standardisation has resulted in challenges for the broader research community, as comparing the effectiveness of robust CE generation methods is impractical.

We fill this gap in this paper and introduce RobustX, an open-source Python library to standardise and streamline the generation, evaluation, and benchmarking of robust CEs. RobustX provides flexible, extensible, and customisable tools to implement custom CE methods. Differently from existing frameworks, e.g. [Pawelczyk *et al.*, 2021; Agarwal *et al.*, 2022], our library focuses on providing a consistent framework for testing robustness and systematically comparing various methods, ensuring fair and reliable evaluations. RobustX addresses key limitations on library tools in the current landscape ([Keane *et al.*, 2021; Jiang *et al.*, 2024c]) by offering a standardised approach to robust CE development while also promoting extensibility, allowing users to integrate new datasets and explanation algorithms as needed. The library, including documentation and tutorials, is publicly available at the following link:

https://github.com/RobustCounterfactualX/RobustX

The reminder of this paper is organised as follows. Section 2 presents the main components of the library, providing details about their functionalities. Section 3 demonstrates how easy it is to use RobustX to benchmark existing CE generation algorithms and compare them using different metrics. Finally, Section 4 offers some concluding remarks and pointers for future work.

2 Overview

RobustX implements a complete pipeline for robust CE generation and evaluation (Figure 2) with three major components: **Task**, **CE generator**, and **CE evaluator**. Each has an abstract class template for easy customisation. Users start by creating a task which defines the model and inputs to be explained. Then, the user can choose whether to use RobustX to generate CEs, or directly evaluate the robustness of previously (externally) generated CEs.

Task objects, providing functionality for interactions between models and datasets, are the basic class passed into the CE generation and evaluation pipelines. Our current implementation assumes a `ClassificationTask` by default, as this is the most commonly considered use case in the literature; however, users can also implement customised `Task` objects for learning problems other than classification. RobustX natively supports models trained using sklearn [Pedregosa *et al.*, 2011], Keras [Chollet and others, 2015] and PyTorch [Paszke, 2019]. Models trained using other frameworks can also be used by instantiating a `BaseModel` wrapper class. As far as datasets are concerned, RobustX offers a selection of pre-loaded example datasets that can be readily loaded using the `DatasetLoader` class. Additionally, this class also allows the uploading of custom datasets if needed via *.csv* files.

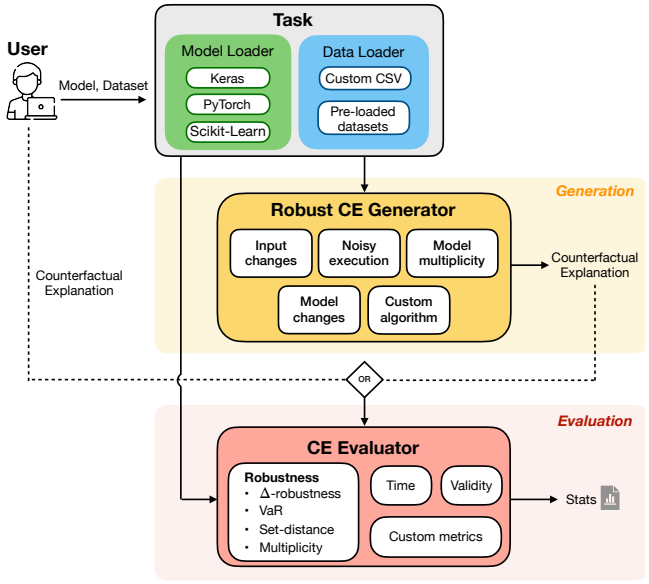


Figure 2: Internal work-flow of RobustX. Users can choose whether to generate robust CEs using our library or evaluate the robustness of externally generated CEs in our RobustX's evaluation facilities.

RobustX currently implements nine **robust CE generation methods** across different robustness use cases reported in the yellow box in Figure 2: `AP`Δ`S` [Marzari *et al.*, 2024], `ArgEnsembling` [Jiang *et al.*, 2024b], `DiverseRobustCE` [Leofante and Potyka, 2024], `MCER` [Jiang *et al.*, 2023b], `ModelMultiplicityMILP` [Leofante *et al.*, 2023], `PROPLACE` [Jiang *et al.*, 2023a], `RNCE` [Jiang *et al.*, 2024a], `ROAR` [Upadhyay *et al.*, 2021], `STCE` [Dutta *et al.*, 2022; Hamman *et al.*, 2023]. It also provides four popular non-robust methods that can be used as baselines to crease new generation methods: `BLS` [Leofante and Potyka, 2024], `MCE` [Mohammadi *et al.*, 2021], `KDTreeNNCE` [Brughmans *et al.*, 2024], and the seminal work by [Wachter *et al.*, 2017]. All methods inherit from the abstract class `CEGenerator` and implement the `_generation_method()` function, providing an easy interface to other components in the pipeline.

CE evaluation methods can take in CEs, either generated within or outside RobustX, and benchmark their robustness along with other common properties identified in the literature. Currently, RobustX provides a `CEEvaluator` class to evaluate the validity and proximity of CEs [Wachter *et al.*, 2017], as well as five classes specifically focusing on robustness evaluation metrics: `VaRRobustnessEvaluator` to assess the validity of CEs after retraining [Dutta *et al.*, 2022], `DeltaRobustnessEvaluator` [Jiang *et al.*, 2023b] to

```
# first prepare a task
from robustx.datasets.ExampleDatasets import get_example_dataset
from robustx.lib.models.pytorch_models.SimpleNNModel import SimpleNNModel
from robustx.lib.tasks.ClassificationTask import ClassificationTask

data = get_example_dataset("ionosphere")
data.default_preprocess()
model = SimpleNNModel(34, [8], 1)
model.train(data.X, data.y)
task = ClassificationTask(model, data)

# specify the names of the methods and evaluations we want to use, run benchmarking
# This will find CEs for all instances predicted with the undesirable class (0) and compare
from robustx.lib.DefaultBenchmark import default_benchmark

methods = ["KDTreeNNCE", "MCE", "MCER", "RNCE", "STCE", "PROPLACE"]
evaluations = ["Validity", "Distance", "Delta-robustness"]

default_benchmark(task, methods, evaluations, neg_value=0, column_name="target", delta=0.005)
```

Figure 3: Code snippet exemplifying how RobustX can be used to easily benchmark CE methods. Table 1 lists the benchmarking results.

assess the robustness of CEs under plausible model changes, `ApproximateDeltaRobustnessEvaluator` [Marzari *et al.*, 2024] assessing probabilistic robustness to plausible model changes, `SetDistanceRobustnessEvaluator` [Leofante and Potyka, 2024] for assessing stability of CEs when the input is perturbed, and `MultiplicityValidityRobustnessEvaluator` [Leofante *et al.*, 2023] for checking CE robustness under model multiplicity. When needed, additional robustness evaluators can be easily added through the extensible interface provided by RobustX.

3 RobustX in Action

In this section we provide an example on how to use RobustX in practice; additional examples are available online. Figure 3 shows how to run and compare six methods supported by RobustX. In this example we focus on robustness against model changes [Upadhyay *et al.*, 2021] and perform a comparison between four robust methods and two non-robust baselines. To this end, we first import the (pre-loaded) `ionosphere` dataset for binary classification (line 6) and apply standard pre-processing. We then create and train a simple three-layer neural network model (line 8). A task object is then created from the dataset and model. Then, we specify the CE generation and evaluation methods of interest (line 16 and 17) and run the benchmarking procedure (line 19). The default benchmark function in this example runs each method with its default hyperparameters, although customised hyperparameters can be configured. It then generates CEs for all instances in the dataset which are predicted with an undesirable class (here 102 points with neg_value=0), and runs the specified evaluation methods. In this example, we evaluate CEs along three metrics: validity, proximity [Wachter *et al.*, 2017] and Δ-robustness [Jiang *et al.*, 2023b]. The results along the selected evaluation metrics are then printed in a structured table, so that we can easily compare how each method performs.

Table 1 shows the results obtained, and reports the computation time, the percentage of valid CEs, average proximity (L2 distance to the input), and the percentage of CEs that are Δ-robust. Observing this table, users of RobustX will be able to identify that robustness performance improvements from the non-robust baselines (NNCE, MCE) to the robust methods (MCER, RNCE, STCE, PROPLACE) are notable, although MCER fails to achieve 100% robustness. Based on these results, users might conclude that `RNCE` is the optimal CE generator for this task, balancing between computation time, proximity, and robustness.

Method	**Time (s)**	**Validity** (%)	**Proximity**	**Rob.** (%)
KDTreeNNCE	0.2	100	5.76	51.6
MCE	3.4	100	2.95	0
MCER	137.6	100	4.84	64.8
RNCE	3.9	100	6.03	100
STCE	39.7	100	7.30	100
PROPLACE	12.9	100	6.02	100

Table 1: Example benchmarking of six CE generation methods.

4 Conclusion

We presented RobustX, a Python framework to generate, evaluate and compare robust CE for ML models. Our library fills a major gap in the existing literature on robust CEs, providing an easy-to-use and extensible platform to benchmark existing algorithms for robust CEs. Building upon extensive research in the area, RobustX provides a unified platform to run and compare existing approaches, as well as implementing new ones, reducing the need to re-implement software from scratch. Work is underway to further expand the list of available generation algorithms, evaluation methods and additional software facilities for testing and validation to ensure the correctness of the implementations. We believe RobustX will streamline research efforts in robust CEs, accelerating the development of innovative solutions and fostering collaboration within this rapidly growing field.

References

[Agarwal *et al.*, 2022] Chirag Agarwal, Satyapriya Krishna, Eshika Saxena, Martin Pawelczyk, Nari Johnson, Isha Puri, Marinka Zitnik, and Himabindu Lakkaraju. Openxai: Towards a transparent evaluation of model explanations. In *NeurIPS*, 2022.

[Brughmans *et al.*, 2024] Dieter Brughmans, Pieter Leyman, and David Martens. NICE: an algorithm for nearest instance counterfactual explanations. *Data Min. Knowl. Discov.*, 38(5):2665–2703, 2024.

[Byrne, 2019] Ruth M. J. Byrne. Counterfactuals in explainable artificial intelligence (XAI): evidence from human reasoning. In *IJCAI*, pages 6276–6282, 2019.

[Cao, 2022] Longbing Cao. AI in finance: challenges, techniques, and opportunities. *ACM Computing Surveys*, 55(3):1–38, 2022.

[Chollet and others, 2015] François Chollet et al. Keras. https://keras.io, 2015.

[Dutta *et al.*, 2022] Sanghamitra Dutta, Jason Long, Saumitra Mishra, Cecilia Tilli, and Daniele Magazzeni. Robust counterfactual explanations for tree-based ensembles. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 5742–5756. PMLR, 2022.

[Guidotti, 2024] Riccardo Guidotti. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery*, 38(5):2770–2824, 2024.

[Hamman *et al.*, 2023] Faisal Hamman, Erfaun Noorani, Saumitra Mishra, Daniele Magazzeni, and Sanghamitra Dutta. Robust counterfactual explanations for neural networks with probabilistic guarantees. In *ICML*, volume 202, pages 12351–12367. PMLR, 2023.

[Jiang *et al.*, 2023a] Junqi Jiang, Jianglin Lan, Francesco Leofante, Antonio Rago, and Francesca Toni. Provably robust and plausible counterfactual explanations for neural networks via robust optimisation. In *ACML*, volume 222 of *Proceedings of Machine Learning Research*, pages 582–597. PMLR, 2023.

[Jiang *et al.*, 2023b] Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. Formalising the robustness of counterfactual explanations for neural networks. In *AAAI*, pages 14901–14909, 2023.

[Jiang *et al.*, 2024a] Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. Interval abstractions for robust counterfactual explanations. *Artificial Intelligence*, 336:104218, 2024.

[Jiang *et al.*, 2024b] Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. Recourse under model multiplicity via argumentative ensembling. In *AAMAS*, pages 954–963. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2024.

[Jiang *et al.*, 2024c] Junqi Jiang, Francesco Leofante, Antonio Rago, and Francesca Toni. Robust counterfactual explanations in machine learning: A survey. In *IJCAI*, pages 8086–8094, 2024. Survey Track.

[Keane *et al.*, 2021] Mark T. Keane, Eoin M. Kenny, Eoin Delaney, and Barry Smyth. If only we had better counterfactual explanations: Five key deficits to rectify in the evaluation of counterfactual XAI techniques. In *IJCAI*, pages 4466–4474, 2021.

[Leofante and Potyka, 2024] Francesco Leofante and Nico Potyka. Promoting counterfactual robustness through diversity. In *AAAI*, pages 21322–21330, 2024.

[Leofante *et al.*, 2023] Francesco Leofante, Elena Botoeva, and Vineet Rajani. Counterfactual explanations and model multiplicity: a relational verification view. In *KR*, pages 763–768, 2023.

[Marzari *et al.*, 2024] Luca Marzari, Francesco Leofante, Ferdinando Cicalese, and Alessandro Farinelli. Rigorous probabilistic guarantees for robust counterfactual explanations. In *ECAI*, volume 392 of *Frontiers in Artificial Intelligence and Applications*, pages 1059–1066. IOS Press, 2024.

[Miller, 2019] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019.

[Mohammadi *et al.*, 2021] Kiarash Mohammadi, Amir-Hossein Karimi, Gilles Barthe, and Isabel Valera. Scaling guarantees for nearest counterfactual explanations. In *AIES*, pages 177–187, 2021.

[Paszke, 2019] Paszke. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035. Curran Associates, Inc., 2019.

[Pawelczyk *et al.*, 2021] Martin Pawelczyk, Sascha Bielawski, Johannes van den Heuvel, Tobias Richter, and Gjergji Kasneci. CARLA: A python library to benchmark algorithmic recourse and counterfactual explanation algorithms. In *NeurIPS Datasets and Benchmarks*, 2021.

[Pedregosa *et al.*, 2011] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[Shaheen, 2021] Mohammed Yousef Shaheen. Applications of artificial intelligence (ai) in healthcare: A review. *ScienceOpen Preprints*, 2021.

[Upadhyay *et al.*, 2021] Sohini Upadhyay, Shalmali Joshi, and Himabindu Lakkaraju. Towards robust and reliable algorithmic recourse. In *NeurIPS*, pages 16926–16937, 2021.

[Wachter *et al.*, 2017] Sandra Wachter, Brent D. Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.*, 31:841, 2017.