Project Plan

Group 17B

Problem analysis

Introduction

While machine learning algorithms are becoming more and more important in data-driven decision-making systems, they are also becoming bigger and thus more opaque. As a result, human operators in charge of these systems rely on them blindly, and the individuals subject to the decisions made have no way of challenging an undesirable outcome. The Counterfactual Explanations package attempts to solve this problem: it provides a way to explain how a system's inputs have to change for it to produce different decisions. The package takes as input a pre-trained model that was fitted to data, the input that was given to the pre-trained model, the actual decision outputted by that model, a target decision that the user wants to move the output towards, and the appropriate counterfactual generator for the problem. The specified counterfactual generator then shows the user how the input features need to change in order for the model to make the target decision instead of the actual decision. For example, if the user was denied a loan by a model and they specify that the target class is being accepted for the loan, the chosen counterfactual generator will show which of the user features (for example, their age, credit score, education, etc.) would have had to have been different for the model to accept them.

Problem statement

CounterfactualExplanations.jl is an open source package developed by Patrick Altmeyer, a PhD candidate at TU Delft. The package is aimed at individuals or organizations who want to use machine learning packages in Julia programming language. The package is designed to generate Counterfactual Explanations and Algorithmic Recourse for black-box algorithms. It is a tool that provides insights into how inputs into a model need to change for it to produce different outputs.

This package is useful in various domains where black-box algorithms are commonly employed, such as finance, healthcare, and criminal justice. In finance, it can be used to provide algorithmic recourse to help individuals who face adverse outcomes, such as getting rejected for a loan or paying higher insurance. In healthcare, it can be used for medical diagnosis and treatment planning. In criminal justice, it can be used to ensure that decision-making is transparent and accountable.

The current implementation of CounterfactualExplanations.jl is compatible with machine learning models trained in Julia programming language. However, the client is seeking assistance in scaling up the package by increasing the scope of predictive models that are compatible with it. Specifically, they are interested in adding support for models trained in Python and R, as well as other machine learning models trained in Julia. This extension is worth pursuing because it will increase the package's applicability to a wider range of machine learning models.

Details

Stakeholders

- Client: Patrick Altmeyer, a PhD candidate at TU Delft who developed the CounterfactualExplanations.jl package.
- Client assistance: Antony Bartlett, a PHD candidate at TU Delft, specialist in testing.
- Project supervisors: Matej Havelka (TA) and Bart Gerritsen (coach) responsible for communication between the students and the client.
- Project team: a group of 5 students who will be contributing to the package's development.
- End-users: individuals or organizations who want to use machine learning packages in Julia programming language.

Use Cases

CounterfactualExplanations.jl can be used in various domains where black-box algorithms are commonly employed, such as finance, healthcare, and criminal justice. Specifically, it can be used to provide insights into how inputs into a model need to change for it to produce different outputs. For example, in finance, the package can be used to generate explanations for why an individual was denied a loan or was given a high interest rate. In healthcare, the package can be used to generate explanations for why a particular treatment was recommended for a patient. In criminal justice, the package can be used to generate explanations for why a certain individual was denied bail or was given a higher sentence. This can be used for algorithmic recourse to help individuals who face adverse outcomes, as well as for transparency and accountability in AI decision-making.

Existing Technologies and Products Analysis

There are already existing products/technologies that do similar things to what we need. We conducted research and found a Python package called CARLA (Counterfactual And Recourse Library for Assessments), which provides a reference implementation for counterfactual explanations and recourse for machine learning models. However, there are some differences in the counterfactual generator between CARLA and CounterfactualExplanations.jl, so we cannot fully incorporate or reuse it. Nonetheless, we can learn from the approaches used in CARLA and use them as inspiration for our own implementation. We also found a couple of datasets and benchmarking tools that we can utilize in our work (e.g. ‘Statlog (German Credit Data) Data Set’ or ‘Fashion-MNIST’).

Users and Experts

The client has suggested that we can meet with Martin Pawelczyk, one of the authors of the CARLA package. We can reach out to him to talk about how to solve particular problems they have already solved when they created the CARLA package.

Feasibility Study

Given the constraints on time and resources, it appears that the project is feasible but may require adjustments depending on the progress made during each phase. The project has goals that are a part of open-ended research, which means that their difficulty is hard to gauge. We have agreed with the client that in the case where one of our predefined goals turns out to be infeasible, we will consider the goal completed upon delivery of a written report explaining the reason for the goal not being possible to complete.

Regarding the availability of technologies, frameworks, and data, there are existing resources like the CARLA library and the ONNX.jl framework that can be utilized, although the latter is currently under reconstruction. The project can also benefit from incorporating similar approaches and datasets to improve its functionality.
We have agreed to be flexible with the frameworks we use, as their effectiveness is difficult to predict.

Risk analysis:

1. Client Availability: While Patrick may have limited availability during the initial weeks, he has assured that he will be more involved after May 17th. Regular meetings can be scheduled to ensure project progress.
2. Hardware and Data Requirements: The project does not require anything besides access to the current github repository of Taija, which has been granted.
3. Legal Issues: We are operating under the MIT license and every team member is responsible for making sure the work they commit does not violate any laws or have copyright issues.
4. Possible blockages due to insufficient prior knowledge:
   Testing question - Refer to Antony, he has strong experience in testing AI systems.
   General question - As the project requires strong domain knowledge, we can expect to become blocked. We have discussed this with our client Patrick and he will be fully available from May 17th onwards, where he will solely focus on our project and answer questions we have about the blockages we will have been facing.
5. Since many of the issues we will be solving require research into the methods and their compatibility with the client’s package, it is uncertain whether all of

them will turn out to be compatible with the package and solvable within the scope of the software project. To anticipate these kinds of problems, we have done the following:

   a. We have agreed with the client on the following Definition of Done: a requirement is fulfilled either once it’s successfully implemented and integrated into the package, or once we have documented with valid and thorough reasoning why that particular task could not be completed within the scope of the software project.
   b. We have gone over all of our issues together with the client and marked all issues that could involve this kind of a risk with an asterisk (*) in our MoSCoW model below.

6. Many of the counterfactual generators fall into the category described under bullet point 5 - we might encounter roadblocks that make them infeasible to implement within the scope of the project. To have fallbacks in this situation, we have two generators specified as Could Haves in the MoSCoW analysis below that we can implement if implementing some of the generators under Must or Should Haves turns out to be impossible. If we encounter such roadblocks, we will make sure that the implementation really is infeasible by discussing the problems we have encountered with our client, who has extensive experience in the field and has already implemented various generators as part of the same package. If we can organize a meeting with Martin Pawelczyk, the author of the CARLA package, that will also help us understand the theoretical details behind the generators better: he has implemented some of the generators we will implement, such as Growing Spheres and Feature Tweak, as part of his own package in Python, and is also a co-author of the paper introducing the PROBE generator.
7. In the case where a generator ends up taking too long to implement (over one week of total development time), then we will cease to spend time on it. The whole process will also be documented, explaining why adding the generator has proven to be challenging and highlighting the approaches that have been tried.
8. There are no risks about availability and compatibility of additional datasets that we will need to add: we have already identified compatible existing datasets in conversation with the client.

Functional Requirements

Must Haves

1. Add counterfactual generator PROBE, based on the approach introduced in Pawelczyk et al. [1]. *
2. Add counterfactual generator Feature Tweak, based on the approach introduced in Tolomei et al. [4]. *
3. Add FashionMNIST from MLDatasets.jl, proposed by Han Xiao et al. [6], as a benchmark dataset.
4. Add Statlog German credit data as a benchmark dataset.
5. Sort out exports (make sure that the default exports of the package are methods that are relevant for the end-user).
6. Investigate ONNX.jl to make the package compatible with Python deep learning models. *
7. Investigate ONNX.jl to make the package compatible with R deep learning models. *
8. Interface to Supervised MLJ models. *
9. Identify and document models which have been marked to have medium or high maturity in the Julia native MLJ model list [7] to figure out which of them are differentiable and can be subjected to counterfactual explanations.
10. Investigate and document whether a one-size-fits-all class would be feasible to handle all compatible MLJ models. If it is, implement it as the subtype of AbstractFittedModel called MLJModel.
11. If Must Have number 10 is not feasible, make a class that handles the conversion for evotrees. *
12. Once the implementation has been finished, create an interactive notebook or a blog post that summarizes our work.

Should Haves

1. Add counterfactual generator Growing Spheres, based on the approach introduced in Laugel et al. [3]. *

2. Add the Adult dataset from the UCI Machine Learning Repository as a benchmark dataset.
3. Add CIFAR10 from MLDatasets.jl as a benchmark dataset.
4. If Must Have number 10 is feasible, implement it. *

Could Haves

1. Perform benchmarking experiments on 1 synthetic and 1 real-world dataset to determine the optimal penalty strength.
2. Develop a module to allow users to customize penalty strength.
3. Integrate automatic penalty strength tuning using machine learning techniques.
4. Investigate using more informed choices for default penalty strength. If feasible, implement.
5. Create an interactive Pluto.jl notebook for exploring different penalty strengths.
6. Investigate using compression techniques such as UMAP [8] for visualizing high-dimensional data.
7. As a fallback option if multiple Must Haves turn out to be infeasible, add the counterfactual generator MINT, based on the approach introduced in Karimi et al. [2]. *
8. As another fallback option, add the counterfactual generator CLUE, based on the approach introduced in Antorán et al. [9]. *
9. Add one more dataset from MLDatasets.jl as a benchmark dataset.
10. Move plotting into a separate package and/or contribute them as Plots.jl recipes.
11. Move the GenerativeModels module into a separate package.

Won't Haves

1. Adding counterfactual generator ROAR, based on the approach introduced in Lakkaraju et al. [5], as this counterfactual generator is inferior to PROBE which we will implement as a must have.
2. Performance profiling as the client has done this themselves.
3. Work on macros.

Non-Functional Requirements

Must haves

1. All functionality will be implemented using the Julia language.
2. All code will be thoroughly tested to at least 80% coverage.
3. The development process will be thoroughly documented.

As noted in the risk sections above: since the project includes many open-ended research tasks, we have agreed with the client that if we document with valid and thorough reasoning why a task could not be completed within the scope of the software project, that also counts as fulfilling the corresponding requirement. Particularly risky issues have been marked with an **asterisk (*)**.

Project approach

Week 3-5

In order to provide the end users with an implementation of Counterfactual Explanations in Julia, we plan to first implement a number of counterfactual generators, as these are the backbone of the package. The generators in question are the PROBE generator and the Feature Tweak generator, as these are important and modern generators that will provide an important starting point. Should this go smoothly, we will then implement the Growing Spheres generator as well. Otherwise we will implement this at a later point. We will also choose what to export for the package around the same time. Additionally, we plan to add some databases to the project for benchmarking purposes. The exact databases we plan to make use of can be found in the requirements section of this document.

Week 5-8

In order to make use of Python and R deep learning models, we will investigate the ONNX.jl package. Currently the package uses RCall/PythonCall which is prone to

errors, so we plan to move towards using ONNX.jl in order to employ a broader range of models. Furthermore, we will attempt to implement an interface for *<: Supervised* MLJ models. We will figure out what models from MLJ are differentiable and consider if it is possible to construct a class which can handle all these models. If this is not feasible, we will implement a class to handle conversion of evotrees instead.
Should we have time left at this point in the project, we will then focus on visualizing higher dimensional data and implementing the MINT counterfactual generator. Furthermore, we will investigate interactivity related to penalty strength, as this is a part of the package end users may be interested in.

Week 8-9

Near the end of the project, we will also provide the client with a blog post or notebook to accompany our work. In these last couple of weeks, we will also revise our code once more to properly evaluate if we met the client's needs and are able to leave it in a finished state.

Of course, everything in the project will be written in Julia as requested by the client. The code we write will always be accompanied by extensive tests that cover at least our solution to the problem. We will also document our process, mostly in the form of comments in the source code.

Development Methodology

We are starting with a development approach that is a modified version of agile. In the beginning, we only have daily standup meetings to keep track of our progress. We are starting light and will adapt accordingly to our needs.

Definition of Done

1. Every new feature is reviewed and manually tested by at least 2 people.
2. Above 80% test coverage. The crux of a solution should be thoroughly tested. This is due diligence of the reviewers.

3. A mirror pull request in GitHub so that our work is synced with the original repository.
4. A generator is considered implemented when it can generate plausible explanations for the example models in the package.
5. The documentation should be up to par with the existing documentation.

Workflow

Our workflow is centered around GitLab, GitHub and VSCode. We might integrate more tools if need be. Other tools should be used at your own discretion.

Communication

We have a Discord channel for miscellaneous information and longer discussions. For short messages, we use a WhatsApp group.

References

[1] Martin Pawelczyk, Teresa Datta, Johannes van-den Heuvel, Gjergji Kasneci, and Himabindu Lakkaraju. 2022. Probabilistically Robust Recourse: Navigating the Trade-offs between Costs and Robustness in Algorithmic Recourse. https://arxiv.org/abs/2203.06768
[2] Amir-Hossein Karimi, Bernhard Schölkopf, and Isabel Valera. 2021. Algorithmic Recourse: From Counterfactual Explanations to Interventions. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT '21)*.
[3] Laugel Thibault, Lesot Marie-Jeanne, Marsala Christophe, Renard Xavier, and Detyniecki Marcin. 2017. Inverse classification for comparison-based interpretability in machine learning.
[4] Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. 2017. Interpretable Predictions of Tree-Based Ensembles via Actionable Feature Tweaking. In International Conference on Knowledge Discovery and Data Mining (KDD) (KDD '17). https://arxiv.org/abs/1712.08443

[5] Sohini Upadhyay, Shalmali Joshi, and Himabindu Lakkaraju. 2021. Towards Robust and Reliable Algorithmic Recourse. arXiv:cs.LG/2102.13620
[6] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. https://arxiv.org/abs/1708.07747
[7] https://alan-turing-institute.github.io/MLJ.jl/dev/list_of_supported_models/
[8] Leland McInnes, John Healy, and James Melville. 2018. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. https://arxiv.org/abs/1802.03426
[9] Javier Antoran, Umang Bhatt, Tameem Adel, Adrian Weller, and José Miguel Hernández-Lobato. 2021. Getting a {CLUE}: A Method for Explaining Uncertainty Estimates. In *International Conference on Learning Representations*.