juliacon

Explaining black-box algorithms using CounterfactualExplanations.jl

Patrick Altmeyer[1] and Cynthia Liem[1]

[1]Delft University of Technology

ABSTRACT

Machine learning models like deep neural networks have become so complex and opaque over recent years that they are generally considered as black boxes. Nonetheless such models play a key role in modern automated decision-making systems. Counterfactual explanations (CE) can help programmers make sense of the systems they build: they explain how inputs into a system need to change for it to produce different decisions. Explanations that involve realistic and actionable changes can be used for the purpose of algorithmic recourse (AR): they offer individuals subject to algorithms a way to turn a negative decision into positive one. In this article we discuss the usefulness of counterfactual explanations for interpretable machine learning and demonstrate its implementation in Julia using the `CounterfactualExplanations` package.

Keywords

Julia, Interpretable Machine Learning, Counterfactual Explanations, Algorithmic Recourse

1. Introduction

In section Section 1

2. Methodological background

Counterfactual search happens in the feature space: we are interested in understanding how we need to change individual attributes in order to change the model output to a desired value or label ([8]). Typically the underlying methodology is presented in the context of binary classification: $M : \mathcal{X} \mapsto y$ where and $y \in \{0, 1\}$. Let $t = 1$ be the target class and let $\overline{x}$ denote the factual feature vector of some individual outside of the target class, so $\overline{y} = M(\overline{x}) = 0$. We follow this convention here, though it should be noted that the ideas presented here also carry over to multi-class problems and regression ([8]).

2.1 Generic framework

Then the counterfactual search objective originally proposed by [16] is as follows

$$\min_{\underline{x}\in\mathcal{X}} h(\underline{x}) \quad \text{s. t.} \quad M(\underline{x}) = t \tag{1}$$

where $h(\cdot)$ quantifies how complex or costly it is to go from the factual $\overline{x}$ to the counterfactual $\underline{x}$. To simplify things we can restate this constrained objective (Equation 1) as the following unconstrained and differentiable problem:

$$\underline{x} = \arg\min_{\underline{x}} \ell(M(\underline{x}), t) + \lambda h(\underline{x}) \tag{2}$$

Here ℓ denotes some loss function targeting the deviation between the target label and the predicted label and λ governs the stength of the complexity penalty. Provided we have gradient access for the black-box model M the solution to this problem (Equation 2) can be found through gradient descent. This generic framework lays the foundation for most state-of-the-art approaches to counterfactual search and is also used as the baseline approach - `GenericGenerator` - in our package. The hyperparameter λ is typically tuned through grid search. Conventional choices for ℓ include margin-based losses like cross-entropy loss and hinge loss. It is worth pointing out that the loss function is typically computed with respect to logits rather than predicted probabilities, a convetion that we have chosen to follow.[1]

Numerous - and in some cases competing - extensions to this simple approach have been developed since counterfactual explanations were first proposed in 2017 (see [15] and [4] for surveys). The various approaches largely differ in how they define the complexity penalty. In [16], for example, $h(\cdot)$ is defined in terms of the Manhattan distance between factual and counterfactual feature values. While this is an intuitive choice, it is too simple to address many of the desirable properties of effective counterfactual explanations that have been set out. These desiderata include: **closeness** - the average distance between factual and counterfactual features should be small ([16]); **actionability** - the proposed feature perturbation should actually be actionable ([14], [11]); **plausibility** - the counterfactual explanation should be plausible to a human ([3]); **unambiguity** - a human should have no trouble assigning a label to the counterfactual ([12]); **sparsity** - the counterfactual explanation should involve as few individual feature changes as possible ([12]); **robustness** - the counterfactual explanation should be robust to domain and model shifts ([13]); **diversity** - ideally multiple diverse counterfactual explanations should be provided ([9]); and **causality** - counterfactual explanations reflect the structual causal model underlying the data generating process ([6],[5]).

2.2 Counterfactuals for Bayesian models

For what follows it is worth elaborating on the approach proposed in [12]. The authors demonstrate that many of the above-

[1] While the rationale for this convention is not entirely obvious, implementations of loss functions with respect to logits are often numerically more stable. For example, the `logitbinarycrossentropy(ŷ, y)` implementation in `Flux.Losses` (used here) is more stable than the mathematically equivalent `binarycrossentropy(ŷ, y)`.

mentioned desiderata can be addressed very easily, if the classifier M is Bayesian. In particular, they show that close, realistic, sparse and unambigous counterfactuals can be generated by implicitly minimizing the classifier's predictive uncertainty through a greedy counterfactual search. Formally, they define $h(\cdot)$ as the predictive entropy of the classifier, which captures both **epistemic** and **aleatoric** uncertainty: the former is high on points far away from the training data while the latter is high in regions of the input space that are inherently noisy. Both are regions we want to steer clear off in our counterfactual search and hence predictive entropy is an intuitive choice for a complexity penalty. The authors further point out that any solution that minimizes cross-entropy loss (Equation 2) also minimizes predictive entropy: $\arg\min_{\underline{x}} \ell(M(\underline{x}),t) \in \arg\min_{\underline{x}} h(\underline{x})$. Let $\widetilde{\mathcal{M}}$ denote the class of binary classifiers that incorporate predictive uncertainty, then the previous observation implies that the optimal solution to counterfactual search (Equation 2) can be restated as follows:

$$\underline{x} = \arg\min_{\underline{x}} \ell(M(\underline{x}),t) \ \ , \ \ \forall M \in \widetilde{\mathcal{M}} \tag{3}$$

We can drop the complexity penalty altogether and still generate effective counterfactual explanations. As we will see below, even a fast and greedy counterfactual search proposed in [12] yields good results in this setting. The approach has been implemented as `GreedyGenerator` in our package and should only be used with classifiers of type $\widetilde{\mathcal{M}}$. It is worth noting that the findings in [12] are not mutually exclusive of many of the other methodologies that have been put foward. On the contrary, we believe that they are complementary: the generic counterfactual search proposed in [16], for example, can be shown to produce more plausible counterfactuals in the Bayesian setting. Similarly, there is no obvious reason why recent work on diversity ([9]), robustness ([13]) and causality ([6],[5]) could not be complemented by the findings in [12]. For this reason we are highlighting [12] here and have prioritized it in the development of `CounterfactualExplanations`. While there is no free lunch and $M \in \widetilde{\mathcal{M}}$ may seem like a hard constraint, recent advances in probabilistic machine learning have shown that the computational cost involved in Bayesian model averaging is lower than we may have thought ([2], [7], [1], [10]).

3. Using `CounterfactualExplanations`

The package is built around two modules that are designed to be as scalable as possible through multiple dispatch: 1) `Models` is concerned with making any arbitrary model compatible with the package; 2) `Generators` is used to implement arbitrary counterfactual search algorithms.[2] The core function of the package `generate_counterfactual` uses an instance of type `T <: FittedModel` produced by the `Models` module (Figure 1) and an instance of type `T <: Generator` produced by the `Generators` module (Figure 2). Relating this back the methodology outlined in Section 2, the former instance corresponds to the model M while the latter defines the rules for the counterfactual search (Equation 2 and Equation 3). In the following we will demonstrate how to use and extend the package architecture through a few examples.

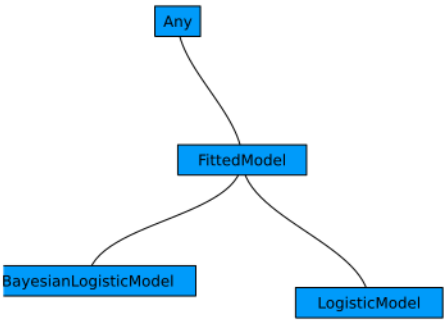

Fig. 1. Schematic overview of the `FittedModel` base type and its descendants.

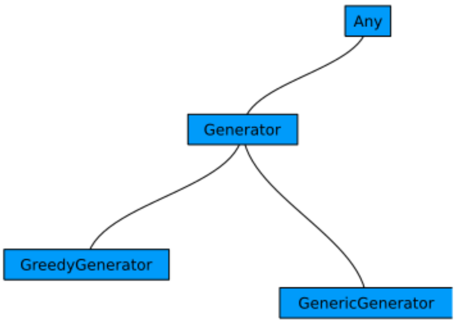

Fig. 2. Schematic overview of the `Generator` base type and its descendants.

3.1 Getting started

The code below provides a complete example demonstrating how the framework presented in Section 2 can be implemeted in Julia using the `CounterfactualExplantions` package: using a synthetic data set with linearly separable samples we firstly define our model and then generate a counterfactual for a randomly selected sample. Figure 3 shows the resulting counterfactual path in the two-dimensional feature space: features go through iterative perturbations until the desired confidence level is reached as illustrated by the contour in the background, which indicates the classifier's predicted probability that the label is equal to 1.

It may help to go through the relevants parts of the code in some more detail starting from the part involving the model. For illustrative purposes the `Models` module ships with a constructor for a logistic regression model: `LogisticModel(W::Matrix,b::AbstractArray) <: FittedModel`. This constructors does not fit the regression model, but rather takes its underlying parameters as given. In

[2] We have made an effort to keep the code base a flexible and scalable as possible, but cannot guarantee at this point that really any counterfactual generator can be implemented without further adaptation.

other words, it is generally assumed that the user has already estimated a model. Based on the provided estimates two functions are already implemented that compute logits and probabilities for the model, respectively. Below we will see how users can use multiple dispatch to extend these functions for use with arbitrary models. For now it is enough to note that those methods define how the model makes its predictions $M(x)$ and hence they form an integral part of the counterfactual search.

With the model M defined in the code below we go on to set up the counterfactual search as follows: 1) choose a random sample `x_factual`; 2) compute its factual label `y_factual` as predicted by the model ($M(\overline{x}) = 0$); and 3) specify the other class as our `target` label ($t = 1$) along with a desired level of `confidence` in the final prediction $M(\underline{x}) = t$.

The last two lines of the code below define the counterfactual generator and finally run the counterfactual search. The first three fields of the `GenericGenerator` are reserved for hyperparameters governing the strength of the complexity penalty, the step size for gradient descent and the tolerance for convergence. The fourth field accepts a `Symbol` defining the type of loss function ℓ to be used. Since we are dealing with a binary classification problem logistic binary cross-entropy is an appropriate choice.[3] The fifth and last field can be used to define mutability constraints for the features.

```julia
# Data:
using CounterfactualExplanations, Random
Random.seed!(1234);
N = 100 # number of data points
using CounterfactualExplanations.Data
x, y = toy_data_linear(N)

# Model:
using CounterfactualExplanations.Models
w = [1.0 1.0]# true coefficients
b = 0
M = LogisticModel(w, [b])

# Setup:
x_factual = x[rand(1:length(x))]
y_factual = round(probs(M, x_factual)[1])
target = ifelse(y_factual==1.0,0.0,1.0)
confidence = 0.75

# Counterfactual search:
generator = GenericGenerator(
    0.1,0.1,1e-5,:logitbinarycrossentropy,nothing)
counterfactual = generate_counterfactual(
    generator, x_factual, M, target, confidence)
```

In this simple example the generic generator produces an effective counterfactual: the decision boundary is crossed (i.e. the counterfactual explanation is valid) and upon visual inspection the counterfactual seems plausible (Figure 3). Still, the example also illustrates that things may well go wrong: since the underlying model produces high-confidence predictions in regions free of any data, it is easy to think of scenarios that involve valid but unrealistic or ambiguous counterfactuals. Consider, for example, the scenario illustrated in Figure 4, which involves the same logisitic classifier albeit massively overfitted. In this case generic search may yield an unrealistic counterfactual that is well into the yellow region and yet far away from all other samples (red marker) or an ambiguous counterfactual near the decision boundary (black marker).

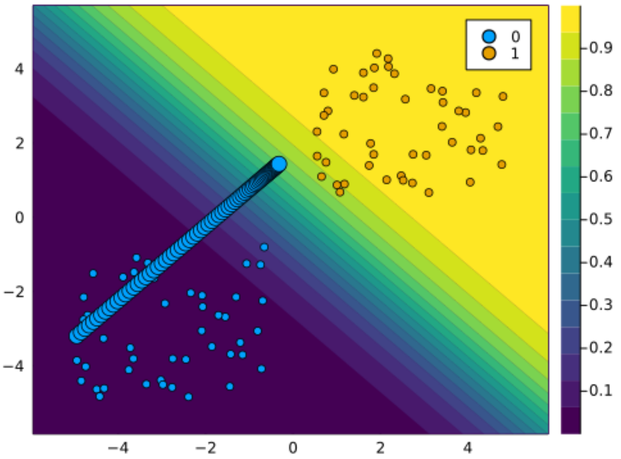

Fig. 3. Counterfactual path using generic counterfactual generator for conventional binary classifier.

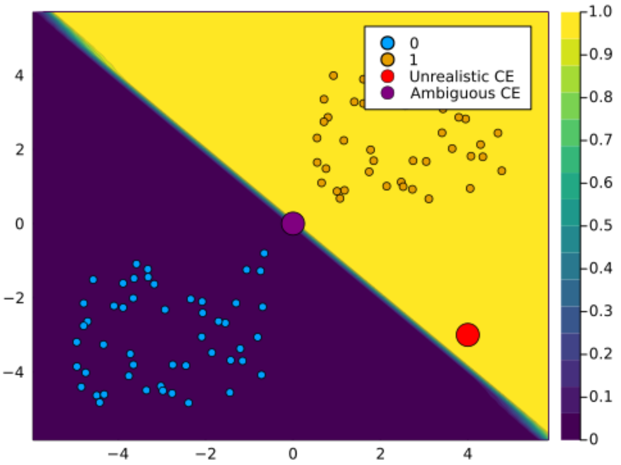

Fig. 4. Unrealistic and ambiguous counterfactuals that may be produced by generic counterfactual search for an overfitted conventional binary classifier.

Among the different approaches that have recently been put forward to deal with such issues is the greedy generator for Bayesian models proposed by [12]. For reasons discussed in Section 2, we have chosen to prioritize this approach in the development of `CounterfactualExplanations`. The code below shows how this approach can be implemented.

```julia
using LinearAlgebra
I = UniformScaling(1)
cov = Symmetric(reshape(randn(9),3,3).*0.01 + I)
w = [1 1]
params = hcat(b, w)
M = BayesianLogisticModel(params, cov);
generator = GreedyGenerator(
    0.25,20,:logitbinarycrossentropy,nothing)
counterfactual = generate_counterfactual(
    generator, x_factual, M, target, confidence)
```

[3] As mentioned earlier, the loss function is computed with respect to logits and hence it is important to use logistic binary cross-entropy loss as opposed to just binary cross-entropy.

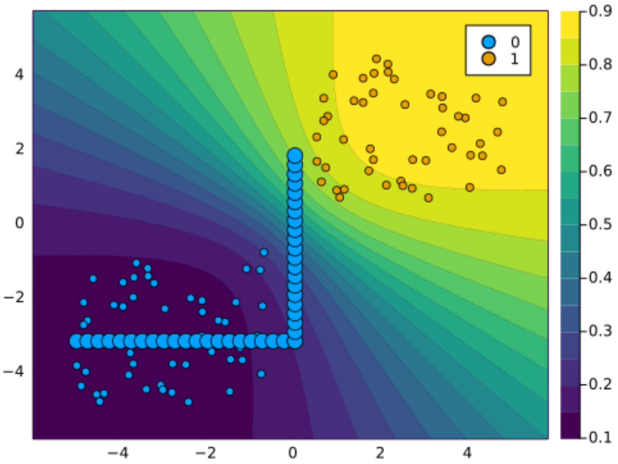

Fig. 5. Counterfactual path using greedy counterfactual generator for Bayesian binary classifier.

3.2 Custom models

4. Empirical example

5. Related and future work

6. References

[1] Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux-effortless bayesian deep learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[2] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[3] Shalmali Joshi, Oluwasanmi Koyejo, Warut Vijitbenjaronk, Been Kim, and Joydeep Ghosh. Towards realistic individual recourse and actionable explanations in black-box decision making systems. *arXiv preprint arXiv:1907.09615*, 2019.

[4] Amir-Hossein Karimi, Gilles Barthe, Bernhard Schölkopf, and Isabel Valera. A survey of algorithmic recourse: definitions, formulations, solutions, and prospects. *arXiv preprint arXiv:2010.04050*, 2020.

[5] Amir-Hossein Karimi, Bernhard Schölkopf, and Isabel Valera. Algorithmic recourse: from counterfactual explanations to interventions. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 353–362, 2021.

[6] Amir-Hossein Karimi, Julius Von Kügelgen, Bernhard Schölkopf, and Isabel Valera. Algorithmic recourse under imperfect causal knowledge: a probabilistic approach. *arXiv preprint arXiv:2006.06831*, 2020.

[7] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[8] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[9] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 607–617, 2020.

[10] Kevin P Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[11] Rafael Poyiadzi, Kacper Sokol, Raul Santos-Rodriguez, Tijl De Bie, and Peter Flach. Face: Feasible and actionable counterfactual explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 344–350, 2020.

[12] Lisa Schut, Oscar Key, Rory Mc Grath, Luca Costabello, Bogdan Sacaleanu, Yarin Gal, et al. Generating interpretable counterfactual explanations by implicit minimisation of epistemic and aleatoric uncertainties. In *International Conference on Artificial Intelligence and Statistics*, pages 1756–1764. PMLR, 2021.

[13] Sohini Upadhyay, Shalmali Joshi, and Himabindu Lakkaraju. Towards robust and reliable algorithmic recourse. *arXiv preprint arXiv:2102.13620*, 2021.

[14] Berk Ustun, Alexander Spangher, and Yang Liu. Actionable recourse in linear classification. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 10–19, 2019.

[15] Sahil Verma, John Dickerson, and Keegan Hines. Counterfactual explanations for machine learning: A review. *arXiv preprint arXiv:2010.10596*, 2020.

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