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 often play a key role in modern automated decision-making systems. Counterfactual explanations 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: 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

Advances in technology have typically gone hand in hand with an outsourcing of labour from humans to machines: the printing press succeeded human scribes centuries ago, ATMs replaced bank tellers decades ago and today robots are swarming factory floors. While these transitions involved a subsitution of manual or repetitive tasks, recent advances in computing and artificial intelligence (AI) have accelerated a new type of transformation: from human to data-driven decision-making. Today, it is more likely than not that your digital loan or employment application will be handled by an algorithm, at least in the first instance. This can in theory be beneficial to you: automation typically leads to increased efficiency and has the potential to remove human bias and error. In reality though, state-of-the-art algorithms are often instable ([6]), encode existing biases ([2]) and learn representations that are surprising or even counter-intuitive form a human perspective (**REFERENES?** []).
This is made more problematic by the fact that many modern machine learning algorithms tend to be so complex and underspecified in the data, that they are essentially black boxes. While this is a known issue, such models are still used to guide decision-making and research in industry as well as academia. At the time of writing, the largest artificial neural networks are made up of several hundreds of billion neurons. In the context of high-stake decision-making systems, black-box models create undesirable dynamics: the human operators in charge of the system have to rely on it blindy, while those indviduals subject to it generally have no way to challenge an outcome. If your digital loan or employment application gets rejected, for example, that is typically the end of the story.

> "You cannot appeal to (algorithms). They do not listen. Nor do they bend."
> — Cathy O'Neil in *Weapons of Math Destruction*, 2016

While the inappropriate abuse of such technologies is arguably the biggest issue, we should also be concerned about missed opportunities. The lack of trustworthiness in machine learning prevents it from being adopted in other fields of research, which might actually benefit from its adoption. Economics and financial markets, for example, are full of complexities and non-linearities that machine learning algorithms are well-equipped to model. But financial practitioners and policy makers are understandably wary of using tools they cannot fully understand ([19],[7]).
In light of all this, a quickly growing body of literature on explainable artificial intelligence has emerged. Counterfactual explanations (CE) and algorithmic recourse (AR) fall into this broader category. Counterfactual explanations 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. Through the `CounterfactualExplanations` package we aim to contribute a scalable and verstile implementation of CE and AR to the Julia community. The remainder of this article is structured as follows: Section 2 presents related work on explainable AI, Section 3 provides a brief overview of the methodological framework, Section 4 presents the package functionality and Section 5 concludes.

2. Related work

2.1 Literature on explainable AI

The field of explainable artificial intelligence (xAI) is still relatively young and made up of a variety of subdomains, definitions, concepts and taxonomies. Covering all of these is beyond the scope of this article, so we will focus only on high-level concepts. The following literature surveys provide more detail: [1] provide a broad overview of xAI; [4] focus on explainability in the context of deep learning; and finally, [10] offer a detailed review of the literature on counterfactual explanations and algorithmic recourse.[1].

[1]Readers who prefer a text-book approach may also want to consider [16] and [27]

The first broad distinction we want to make here is between **interpretable** and **explainable** AI. These terms are often used interchangeably, but this can lead to confusion. We find the distinction made in [22] useful: interpretable AI involves models that are inherently interpretable such as general additive models (GAM), decision trees and rule-based models; explainable AI involves models that are not inherently interpretable, but require additional tools to be explainable to humans. Examples of the latter include ensembles, support vector machines and deep neural networks. Some would argue that we best avoid the second category of models [[22]] and instead focus solely on interpretable AI. While we agree that initial efforts should always be geared towards interpretable models, avoiding black boxes altogether would entail missed opportunities and anyway is probably not very realistic at this point. For that reason, we expect the need for explainable AI to persist in the near future. Explainable AI can further be broadly divided into **global** and **local** explainability: the former is concerned with explaining the average behaviour of a model, while the latter involves explanations for individual predictions [16]. Tools for global explainability include partial dependence plots (PDP), which involves the computation of marginal effects through Monte Carlo, and global surrogates. A surrogate model is an interpretable model that is trained to explain the predictions of a black-box model.
Counterfactual explanations fall into the category of local methods: they explain how individual predictions change in response to individual feature perturbations. Among the most popular alternatives to counterfactual explanations are local surrogate explainers including local interpretable model-agnostic explanations (LIME) and Shapley additive explanations (SHAP). They are among the most widely used xAI tools today, potentially because they are easily understood, relatively fast and implemented in popular programming languages. Proponents of surrogate explainers also commonly mention that there is a straight-forward way to assess their reliability: a surrogate model that generates predictions in line with those produced by the black-box model is said to have high fidelity. As intuitive as this notion may be, it also points to an obvious shortfall of surrogate explainers: even a highly fidel surrogate model that produces the same predictions as the black-box model 99 percent of the time is useless and potentially misleading for every 1 out 100 individual predictions. In fact, a recent study has shown that even experienced data scientists tend to put too much trust in explanations produced by LIME and SHAP [13]. Another recent work has shown that both LIME and SHAP can be easily fooled: both methods depend on random input perturbations, a property that be abused by adverse agents to essentially whitewash strongly biased black-box models [**?**]. In a related work the same authors find that while gradient-based counterfactual explanations can also be manipulated, there is a straight-forward way to this in practice [24]. In the context of quality assessment, it is also worth noting that - contrary to surrogate explainers - counterfactual explanations always achieve full fidelity by construction: counterfactuals are search with respect to the black-box classifier, not some approximation of it. That being said, counterfactual explanations should also be used with care and research around them is still at its early stages. We shall discuss this in more detail in Section 3.

2.2 Existing software

To the best of our knowledge the `CounterfactualExplanations.jl` package provides the first implementation of counterfactual explanations in Julia and therefore represents a novel contribution to the community. As for other programming languages, we are only aware of one other unifiying framework: CARLA is Python library that was recently introduced ([20]). In addition to that, there exists open-source code for some specific approaches to counterfactual explanations that have been proposed in recent years. The approach-specific implementations that we have been able to find are generally well documented, but exclusively in Python. For example, a PyTorch implementation of a greedy generator for Bayesian models proposed in [23] can be found here. The popular InterpretML library includes an implementation of a diverse counterfactual generator proposed by [17].
Generally speaking though, software development in the space of xAI has largely focused on various global methods and surrogate explainers: implementations of PDP, LIME and SHAP are available for both Python (e.g. `lime`, `shap`) and R (e.g. `lime`, `iml`, `shapper`, `fastshap`). In the Julia space, we have only been able to identify one package that falls into the broader scope of xAI, namely `ShapML.jl`](https://github.com/nredell/ShapML.jl) which provides an implementation of SHAP. We also should not fail to mention the comprehensive Interpretable AI infrastructure, which focuses exclusively on interpretable models. Arguably the current availability of tools for explaining black-box models in Julia is limited, but it appears that the community is invested in changing that. The team behind `MLJ.jl`, for example, is currently recruiting contributors for a project about both interpretable and explainable AI. With our work on counterfactual expanations we hope to contribute to these efforts. We think that because of its unique transperancy the Julia language naturally lends itself towards establishing a greater degree of trust in machine learning and artificial intelligence.

3. 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 ([16]). 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 ([16]).

3.1 Generic framework

Then the counterfactual search objective originally proposed by [29] 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 convention that we have chosen to follow.[2]

Numerous - and in some cases competing - extensions to this simple approach have been developed since counterfactual explanations were first proposed in 2017 (see [28] and [10] for surveys). The various approaches largely differ in how they define the complexity penalty. In [29], 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 ([29]); **actionability** - the proposed feature perturbation should actually be actionable ([26], [21]); **plausibility** - the counterfactual explanation should be plausible to a human ([9]); **unambiguity** - a human should have no trouble assigning a label to the counterfactual ([23]); **sparsity** - the counterfactual explanation should involve as few individual feature changes as possible ([23]); **robustness** - the counterfactual explanation should be robust to domain and model shifts ([25]); **diversity** - ideally multiple diverse counterfactual explanations should be provided ([17]); and **causality** - counterfactual explanations reflect the structual causal model underlying the data generating process ([12],[11]).

3.2 Counterfactuals for Bayesian models

For what follows it is worth elaborating on the approach proposed in [23]. The authors demonstrate that many of the abovementioned 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 [23] yields

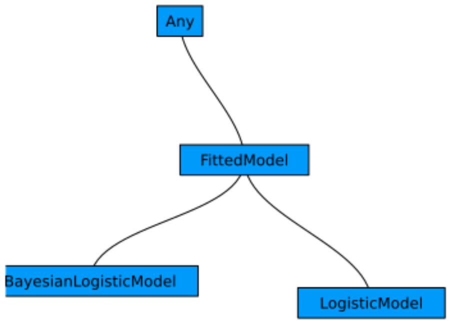

Fig. 1. Schematic overview of the `FittedModel` base type and its descendants.

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 [23] 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 [29], 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 ([17]), robustness ([25]) and causality ([12],[11]) could not be complemented by the findings in [23]. For this reason we are highlighting [23] 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 ([5], [14], [3], [18]).

4. 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.[3] 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 3, 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.

4.1 Getting started

The code below provides a complete example demonstrating how the framework presented in Section 3 can be implemeted in Julia

[2]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)`.

[3]We have made an effort to keep the code base a flexible and scalable as possible, but camodelot guarantee at this point that really any counterfactual generator can be implemented without further adaptation.

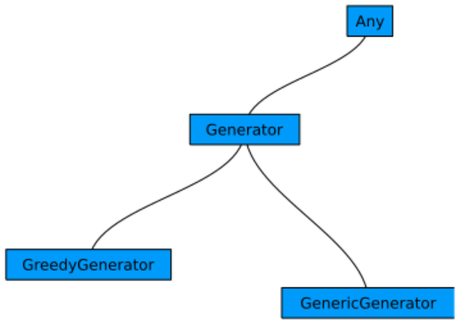

Fig. 2. Schematic overview of the `Generator` base type and its descendants.

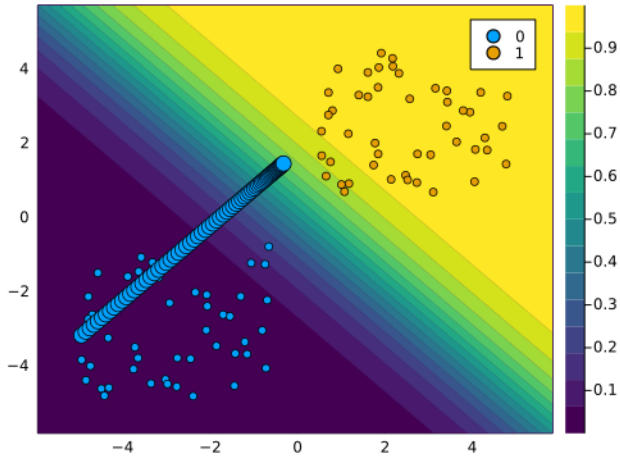

Fig. 3. Counterfactual path using generic counterfactual generator for conventional binary classifier.

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 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 logit binary cross-entropy is an appropriate choice.[4] The fifth and last field can be used to define mutability constraints for the features.

```
# 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

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

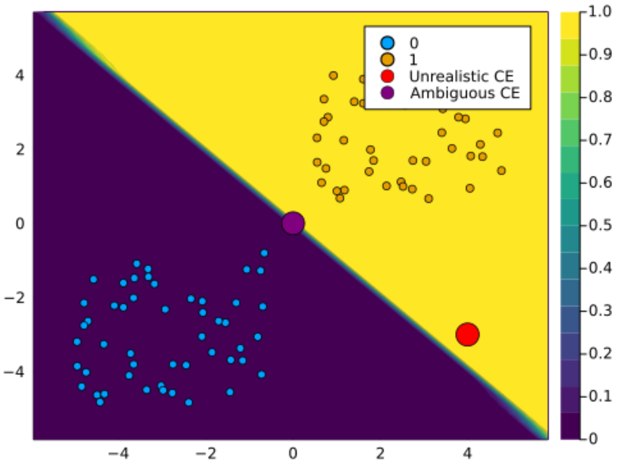

Fig. 4. Unrealistic and ambiguous counterfactuals that may be produced by generic counterfactual search for an overfitted conventional binary classifier.

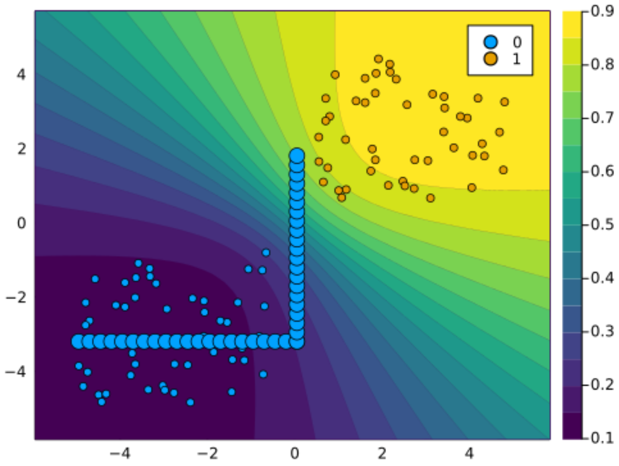

Fig. 5. Counterfactual path using greedy counterfactual generator for Bayesian binary classifier.

yet far away from all other samples (red marker) or an ambiguous counterfactual near the decision boundary (black marker).

Among the different approaches that have recently been put forward to deal with such issues is the greedy generator for Bayesian models proposed by [23]. For reasons discussed in Section 3, we have chosen to prioritize this approach in the development of `CounterfactualExplanations`. The code below shows how this approach can be implemented. Figure 5 shows the resulting counterfactual path through the feature space along with the predicted probabilities from the Bayesian classifier.

Once again it is worth dwelling on the code for a moment. We have used the same synthetic toy data as before, but this time we use assume that we have fit a Bayesian logistic regression model through Laplace approximation. This approximation uses the fact the second-order Taylor expansion of the logit binary cross-entropy function evaluated at the maximum-a-posteriori (MAP) estimate amounts to a multivariate Gaussian distribution ([18]).[5] The `BayesianLogisticModel <: FittedModel` constructor takes the two moments defining that distribution as its arguments: firstly, the MAP esitmate, i.e. the vector of parameters $\hat{\mu}$ including the constant term and, secondly, the corresponding covariance matrix $\hat{\Sigma}$. As with logisitic regression above, the package ships with methods to compute predictions from instances of type `BayesianLogisticModel`.[6] Contrary to the simple logisitic regression model above, predictions from the Bayesian logistic model incorporate uncertainty and hence predicted probabilities fan out in regions free of any training data (Figure 5).

For the counterfactual search we use a greedy approach following [23]. The approach is greedy in the sense that in each iteration it selects the most salient feature with respect to our objective (Equation 3) and perturbs it by some predetermined step size δ. Since the gradient $\nabla_{\underline{x}}\ell(M(\underline{x},t))$ is proportional to the MAP estimate $\hat{\mu}$, the same feature is chosen until a predefined maximum number of perturbations n has been exhausted. Those two hyperparameters, δ and n, are defined in the first two fields of `GreedyGenerator <: Generator` in the code below. The third and fourth field are reserved for the loss function and mutability constraints. Since we are making use of multiple dispatch, the final command that actually runs the counterfactual search is the same as before.

```
# Model:
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)

# Counterfactual search:
generator = GreedyGenerator(
    0.25,20,:logitbinarycrossentropy,nothing)
counterfactual = generate_counterfactual(
    generator, x_factual, M, target, confidence)
```

The counterfactual in Figure 5 is not only valid, but also realistic and unambiguous. In this case it is more difficult to imagine adverse scenarios like in Figure 4. Evidently it is easier to avoid pitfalls when generating counterfactual explanations for models that incorporate predictive uncertainty.

4.2 Custom models

One of our priorities has been to make `CounterfactualExplanations` scalable and versatile. In the long term we aim to add support for more default models and counterfactual generators. In the short term it is designed to allow users to integrate models and generators themselves. Ideally, these community efforts will facilitate our long-term goals. Only two steps are necessary to make any supervised-learning model compatible with our package[7]:

To demonstrate how this can be done in practice we will now consider another synthetic example. Once again samples are two-dimensional for illustration purposes, but this time they are grouped

[5] See also this blog post for a gentle introduction and implementation in Julia.

[6] Predictions are computed using a probit approximation.

[7] In order for the model to be compatible with the gradient-based default generators presented in Section 4.1 gradient access is also necessary, but any model can also be complemented with a custom generator.

into four different classes and not linearly separable. To predict class labels based on features we use a simple deep-learning model trained in Flux.jl ([8]). The code below shows the simple model architecture. Note how outputs from the final layer are note passed through a softmax activation function, since counterfactual loss is evaluated with respect to logits as we discussed earlier. The model is trained with dropout for ten training epochs.

```
n_hidden = 32
output_dim = length(unique(y))
input_dim = 2
model = Chain(
    Dense(input_dim, n_hidden, activation),
    Dropout(0.1),
    Dense(n_hidden, output_dim)
)
```

The code below implements the two steps that are necessary to make the trained neural network compatible with the package: subtyping and multiple dispatch. Computing logits amounts to just calling the Flux.jl model on inputs. Predicted probabilities for labels can than be computed through softmax.

```
# Step 1)
struct NeuralNetwork <: Models.FittedModel
    model::Any
end

# Step 2)
# import functions in order to extend
import CounterfactualExplanations.Models: logits
import CounterfactualExplanations.Models: probs
logits(M::NeuralNetwork, X::AbstractArray) =
M.model(X)
probs(M::NeuralNetwork, X::AbstractArray) =
softmax(logits(M, X))
M = NeuralNetwork(model)
```

Finally, the code below draws a random sample and generates a counterfactual in a different target class through generic search. The code very much resembles the earlier examples, with the only notable difference that for the counterfactual loss function we are now using the multi-class logit cross-entropy loss. The resulting counterfactual path is shown in Figure 6. In this case the contour shows the predicted probability that the input is in the target class ($t = 4$). Generic search yields a valid, realistic and unambiguous counterfactual.

```
# Randomly selected factual:
using Random
Random.seed!(42)
x_factual = x[rand(1:length(x))]
y_factual = Flux.onecold(
    probs(M, x_factual),unique(y))
target = rand(unique(y)[1:end .!= y_factual])
confidence = 0.75

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

As before we will also look at the Bayesian setting. Using Laplace approximation (LA) much in the same way as above we can recover a Bayesian representation of our neural network in a post-hoc fashion ([3]). Alternatively, we could have considered using a deep ensemble ([14]), Monte Carlo dropout ([5]) or variational inference. Using the greedy generator yields the counterfactual path in Figure 7. The code that produces these results follows below.

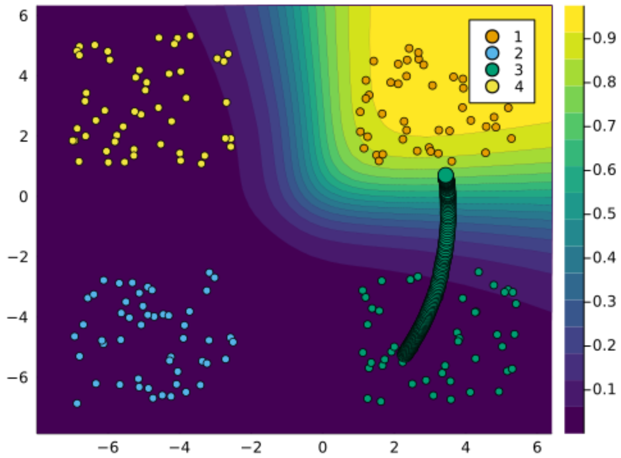

Fig. 6. Counterfactual path using generic counterfactual generator for multi-class classifier.

Contrary to the example involving binary classification above, it is less clear that counterfactuals for the Bayesian classifier are more effective in this case. While predictions from this simple Bayesian neural network are overall more conservative, the model fails to only produce high-confidence predictions in regions that are abundant with training samples. This illustrates that the quality of counterfactual explanations may ultimately depend to some degree on the quality of the classifier. Put differently, if the quality of the classifier is poor, we may expect this to come through in the counterfactual explanation.

```
# Fitting the Laplace approximation:
using BayesLaplace
la = laplace(model)
fit!(la, data)

# Model:
# Step 1)
struct LaplaceNeuralNetwork <: Models.FittedModel
    la::BayesLaplace.LaplaceRedux
end

# Step 2)
logits(M::LaplaceNeuralNetwork, X::AbstractArray) =
M.la.model(X)
probs(M::LaplaceNeuralNetwork, X::AbstractArray) =
BayesLaplace.predict(M.la, X)
M = LaplaceNeuralNetwork(la)

# Counterfactual search:
generator = GreedyGenerator(
    0.25,30,:logitcrossentropy,nothing)
counterfactual = generate_counterfactual(
    generator, x_factual, M, target, confidence)
```

4.3 Empirical example

Now that we have explained the basic functionality of `CounterfactualExplanations` through a few illustrative toy ex-

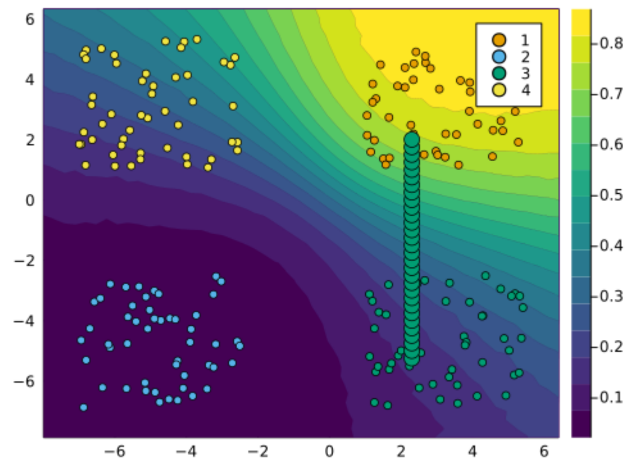

Fig. 7. Counterfactual path using generic counterfactual generator for multi-class classifier with Laplace approximation.

amples, it is time to consider some real data. The MNIST dataset contains 60,000 training samples of handwritten digits in the form of 28x28 pixel grey-scale images ([15]). Each image is associated with a label indicating the digit (0-9) that the image represents. The data makes for an interesting case-study of counterfactual explanations, because humans have a good idea of what realistic counterfactuals of digits look like. For example, if you were asked to pick up an eraser and turn the digit in Figure 8 into a four (4) you would know exactly what to do: just erase the top part. In [23] leverage this idea to illustrate to the reader that their methodolgy produces effective counterfactuals. In what follows we replicate some of their findings. You as the reader are therefore the perfect judge to evaluate the quality of the counterfactual explanations presented here.
On the model side we will use two pre-trained classifiers[8]: firstly, a simple multi-layer perceptron (MLP) and, secondly, a deep ensemble composed of five such MLPs following [23]. Deep ensembles are approximate Bayesian model averages that have been shown to yield high-quality esimtates of predictve uncertainty for neural networks ([**?**], [14])). In the previous section we already created the necessary subtype and methods to make the multi-output MLP compatible with our package. The code below implements the two necessary steps for the deep ensemble.

```
using Flux: stack
# Step 1)
struct FittedEnsemble <: Models.FittedModel
    ensemble::AbstractArray
end
# Step 2)
using Statistics
logits(M::FittedEnsemble, X::AbstractArray) =
mean(
    stack([m(X) for m in M.ensemble],3),
    dims=3)
probs(M::FittedEnsemble, X::AbstractArray) = mean(
    stack([softmax(m(X)) for m in M.ensemble],3),
    dims=3)
M_ensemble = FittedEnsemble(ensemble)
```

[8]The pre-trained models were stored as package artifacts and loaded through helper functions.

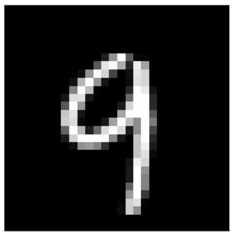

Fig. 8. A handwritten nine (9) randomly drawn from the MNIST dataset.

For the counterfactual search we will use four different combinations of classifiers and generators: firstly, the generic approach for the MLP; secondly, the greedy approach for the MLP; thirdly, the generic approach for the deep ensemble; and finally, the greedy approach for the deep ensemble.
TBD

5. Concluding remarks

6. References

[1] Alejandro Barredo Arrieta, Natalia Diaz-Rodriguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

[2] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018.

[3] 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.

[4] Fenglei Fan, Jinjun Xiong, and Ge Wang. On interpretability of artificial neural networks. *Preprint at https://arxiv.org/abs/2001.02522*, 2020.

[5] 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.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[7] Kristian Bondo Hansen. The virtue of simplicity: On machine learning models in algorithmic trading. *Big Data & Society*, 7(1):2053951720926558, 2020.

[8] Mike Innes. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 3(25):602, 2018.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Harmanpreet Kaur, Harsha Nori, Samuel Jenkins, Rich Caruana, Hanna Wallach, and Jennifer Wortman Vaughan. Interpreting interpretability: understanding data scientists' use of interpretability tools for machine learning. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pages 1–14, 2020.

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

[15] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

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

[17] 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.

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

[19] OECD. Artificial intelligence, machine learning and big data in finance: Opportunities, challenges and implications for policy makers, 2021.

[20] 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. *arXiv preprint arXiv:2108.00783*, 2021.

[21] 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.

[22] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[23] 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.

[24] Dylan Slack, Anna Hilgard, Himabindu Lakkaraju, and Sameer Singh. Counterfactual explanations can be manipulated. *Advances in Neural Information Processing Systems*, 34, 2021.

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

[26] 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.

[27] Kush R. Varshney. *Trustworthy Machine Learning*. Independently Published, Chappaqua, NY, USA, 2022.

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

[29] 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.