Tuneable Counterfactuals

Joshan Parmar, Pietro Lio and Soumya Banerjee

November 2023

Abstract

By answering the question "What must I change in the input to change the output of a model", counterfactuals aim to explain the outputs of machine learning systems.

In this paper, we propose a novel methodology for generating such counterfactuals. We focus on "sequential counterfactuals", which give the "changes" as a series of "steps" that an end-user should take. Our "human-in-the-loop" methodology borrows ideas from research in Human-Computer Interaction to allow domain experts and end-users to tune the chosen counterfactuals for different use cases.

We discuss how well our methodology performs in several applications: financial (where many institutions use automated systems to decide whether individuals are entitled to loans), medical (where AI systems aid clinical professionals in decision-making) and image classification.

We exploit the "modularity" of our proposed method to examine possible variations on our approach and how well it performs when the model's outputs are "noisy".

1 Introduction

1.1 Explainable AI

Automated "artificially intelligent" systems are increasingly intertwined with our everyday lives — from medical diagnosis to finance and image recognition. While some situations will use directly interpretable models — such as logistics regression [Molnar, 2023] — others will be "black-box" [Castelvecchi, 2016]. For consumers to trust these "black-box" models, they often want some "explanation" as to what this automated system is doing — an answer to why it made the decision it did. Explainable AI (XAI) is a field of research that aims to develop tools and techniques to answer these questions. XAI can also be used for debugging models by finding where the model has learnt spurious relationships in the training data and providing algorithmic accountability.

1.2 Counterfactuals

One way of providing these explanations is through counterfactuals. The classic example of a counterfactual explanation of a model involves a situation where we have trained a model to decide whether a bank should provide an individual with a loan. A helpful way to explain the predictions this model makes is to say what a user must modify to change the prediction made by a model. For example, "You were denied this loan, but if your income was £10,000 higher, you would have been granted this loan", or "You were granted this loan, but you would not have been granted a loan of a higher value". Similarly, we could say "This image was classified as a 1, but if we filled in these pixels, the image would be classified as a 0."

Formally, given a data point x and a classification model f, we define a counterfactual as a different data point x' such that $f(x) \neq f(x')$. We want our counterfactual point to be, by some metric, close to the original point, so we can solve this through an optimization problem.

1.2.1 Existing methodologies for Generating Counterfactuals

To examine the effectiveness and utility of this novel methodology, we compare it against some existing methodologies for generating counterfactuals. While we give a broader overview of the current research in our Literature Review, we begin by giving a detailed mathematical explanation of our two choices of comparisons. We chose these options as there are publicly available Python packages [Van Rossum and Drake Jr, 1995, Klaise et al., 2021] implementing these. As we also make a Python implementation of our methodology publicly available, these are reasonable points of comparison — though it is not exhaustive.

AlibiExplain, developed by Klaise et al. [2021] implements the method described by Wachter et al. [2017]. This methodology works by performing the following minimization:

$$\arg\min_{x'} \left(\max_{\lambda} \left(\lambda \left(f_w \left(x' \right) - y_i \right)^2 + d \left(x_i, x' \right) \right) \right) \tag{1}$$

In this case, (x_i, y_i) represents the point we are trying to find a counterfactual for, f_w is the model and d is some distance metric. Wachter et al. propose using the ADAM distance metric [Kingma and Ba, 2017].

Klaise et al. also implement a more complex methodology that utilizes "prototypes", as introduced by Van Looveren et al. [2021]. This methodology also minimizes a loss function $\mathcal{L}$ over some D-dimensional feature space. The loss function they use is

$$\mathcal{L} = c \cdot L_{pred} + \beta \cdot L_1 + L_2 + L_{AE}^{\gamma} + L_{proto}^{\theta} \tag{2}$$

$$L_{pred} = \max\left(\underbrace{[f(x_0+\delta)]_i}_{\text{Original class}} - \underbrace{\max_{t\neq i}([f(x_0+\delta)]_i), -\kappa}_{\text{New class}}\right) \tag{3}$$

L_{pred} encourages the class to change, L_{AE}^{γ} penalizes (by a factor of γ) out of distribution counterfactual instances (by using the L_2 reconstruction error from an auto-encoder[1] [Jordan, 2018] trained on the training set), L_{proto} acts by a factor of θ to both speed up the search process and guide the perturbation towards an interpretable counterfactual, and L_1 and L_2 correspond to the usual norms. Mathematically, L_{proto} works by first using an auto-encoder $\text{Enc} : \mathbb{R}^N \to \mathbb{R}^E$ to project x onto an E-dimensional latent space $\mathbb{R}^E$. Then, for each class, i, they apply this encoder to all training set instances belonging to that class. Then take the k-nearest instances in this latent space and define:

$$\text{proto}_i := \frac{1}{k}\sum_{t=1}^{k} ENC(x_t^i) \tag{4}$$

and j as the "closest" other class in $\mathbb{R}^E$ latent-space.,

$$j = \underset{i\neq\text{original}}{\arg\min} \left|ENC(x_0) - \text{proto}_i\right|_2 \tag{5}$$

Finally, the prototype loss term is defined as the distance of the proposed counterfactual to the average representation of the new class in $\mathbb{R}^E$ latent space.

$$L_{\text{proto}} = \theta \cdot \left|ENC(x_0+\delta) - \text{proto}_j\right|_2^2 \tag{6}$$

In effect, this guides perturbations towards the "nearest prototype". We do not provide a specific encoder, so class representations are built using k-d trees [Bentley, 1975].

1.2.2 Practical uses of Counterfactual Explanations

As with all forms of explanation for automated systems, they can increase a user's trust in an automated system [Inam et al., 2021]. By attempting to approximate the "reasoning" that these automated systems have employed, users can assess whether they agree with this reasoning. These counterfactual explanations can also be tuned to be actionable — this means that an end user can take actions based on their received explanations. For example, if an explainer suggests that an applicant was denied for a loan but would have been accepted for a smaller loan, the applicant can respond by applying for this smaller loan. Similarly, in a medical situation, if the explainer suggests that a patient who exercised more would be less likely to have severe complications from a disease,

[1] An Auto-encoder is a neural network trained to copy its input to its output. The hidden layer acts as a bottleneck in a lower-dimensional space. It can be, in some senses, be considered a compression algorithm designed to work well on the distribution in the training set.

the patient could respond by doing this. Of course, not all changes are possible for all situations, so a "tuneable", "human-in-the-loop" methodology is beneficial.

Similarly, as discussed by Blackwell [2022], users of an automated system are entitled to an explanation of why such a system made the decision it did. Counterfactual explanations can be one part of meeting this legal requirement [Goodman and Flaxman, 2017]. Counterfactuals can also help data scientists understand the model's internal structure better and correct for unintentional biases in the training data. For example, suppose counterfactuals generated for a financial model suggest that "protected characteristics" should be changed to change the output of the model. In that case, it is likely that the model has learnt unintended biases and that further work is needed to de-bias the training data before deploying the model in practice.

When domain experts are presented with such "counterintuitive" explanations, they can be prompted to examine the training data set for these biases, leading to fairer, more accurate models. Ultimately, this also can be a part of legal requirements to conduct impact assessments for model's "bias [and] effectiveness" [Office of Ron Wyden, 2022].

2 Methods

2.1 Methodology for Generating Counterfactuals

We can summarize our general methodology as "chaining together" simple local models to find counterfactual explanations in a highly tuneable way.

We begin by fitting simple surrogate models (Section 2.1.1) to the variation of each relevant feature in the model. We then use a tuneable set of scoring rules, as discussed in Section 2.1.2, to assign each of these a score. If any of these models suggest changing that variable is sufficient to cross the decision boundary, we return the "best" such model (in terms of this score). Otherwise, we are able to iterate from the the extrema points of these model (the feature value that takes us closest to crossing the decision boundary). Various strategies are possible to implement this "iterative search", which we discuss in detail in Section 2.1.3.

2.1.1 Surrogate Models

These surrogate models, which we will often refer to as single variable explainers (SVEs), attempt to explain the dependence of the model's output on the variation of one parameter locally.

For a particular variable, we generate a set of input data points by replacing that variable with other values that the variable takes in the training data set. Either we do this through Monte-Carlo-esque sampling — i.e. replacing with randomly chosen other values in the data set — or by selecting uniformly placed points across some interval, such as $[\mu \pm \sigma]$ (where μ is the variable's mean value in the training data and σ is the standard deviation) or defined "quantiles" of the values of that feature in the training data. We then apply the (black-box) model to these to get a series of sampling variables and class probabilities pairs.

We then fit a simple (explainable) model to this new data set. We usually use either a Gaussian Process or sometimes a linear model. Our code is sufficiently general to allow for any SciKit-Learn regression model to be used.

2.1.2 Scoring Rules

A range of rules can be used to score these models, by default, we consider the following:

$$S_{\text{basic}}(x_i^{\text{new}}) = \text{abs}\left(\text{Underlying Model}\left(\mathbf{x}^{\text{original}}\right) - \text{SVE}\left(x_i\right)\right) \tag{7}$$

In some cases, our model to recommend smaller changes to features so we can also consider the following:

$$S_{\text{masked}}(x_i^{\text{new}}) = \exp\left(-\frac{1}{2}\left(\frac{x_i^{\text{new}} - x_i^{\text{original}}}{\sigma_i^{\text{training}} \times f(w_i)}\right)^2\right) \times S_{\text{basic}}(x_i^{\text{new}}) \tag{8}$$

Finally, each score is weighted by a user-provided "feature changeability score" that quantifies how easily a feature can be changed. $S_{\text{Final}}\left(x_i^{\text{new}}\right) = w_i \cdot S\left(x_i^{\text{new}}\right)$ Borrowing ideas from the field of "Human Computer Interaction", this allows the user to "tune" the methodology to their use case.

This is visualized in Figure 1.

2.1.3 Search Rules

Our search system can be considered a modified form of "depth-first" search. We initially train surrogate models for each "changeable" feature of the example. We select the best (using the rules described in Section 2.1.2) and "change" that feature to the value that takes us closest to it's best value. From this new point, we retrain surrogate models and repeat this process until we cross our decision boundary. If we "run out" of features to change, we return one layer up the "tree" and change the next best feature. We continue this process until we cross the decision boundary. This process is illustrated in Figure 2.

We note that we have built our system such that it is "modular", so that we can easily change the search process. For example, we can change the order in

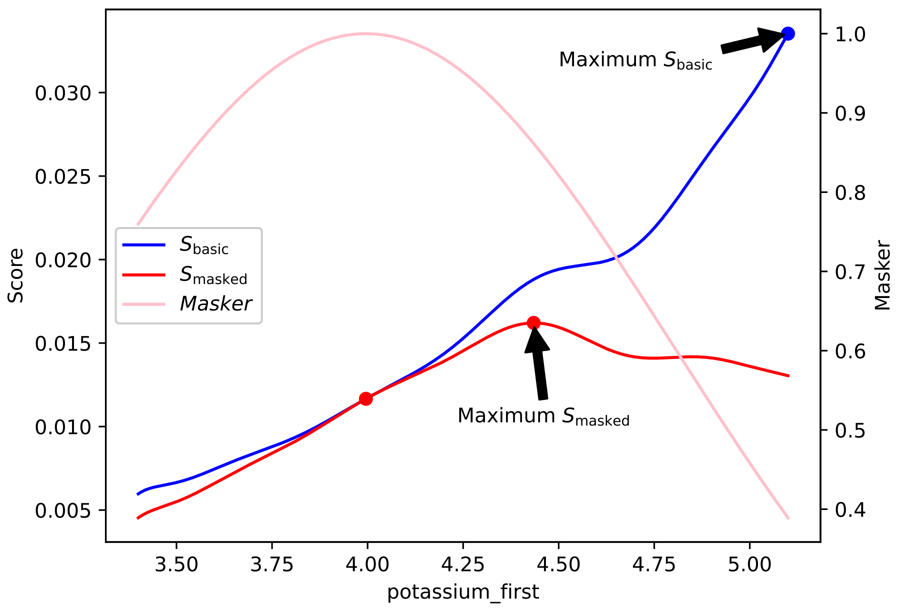


Figure 1: Example of how scoring rules work. The "basic" scoring rule is shown in blue, while the "masked" scoring rule is shown in red. The "masked" scoring rule is "masked" by a bell curve, shown in pink.

which we change features. We can also change the "stopping criteria" — i.e. when we stop changing features and decide there does not exist a counterfactual subject to the given criteria.

In terms of technical implementation, this requires the definition of a simple "searcher" class that can be passed to our "counterfactual" class. This searcher class only requires two functions. Firstly, `get_next_evaluation_set` should, given the current "node", select which nodes to evaluate next. Similarly, `get_next_node` should, given the scores of the previous evaluation, select the next node to evaluate. This allows for a wide range of search strategies to be implemented.

For this paper we use only S_{basic}, as defined in Equation 7.

2.2 Evaluation Methodology

We will evaluate our method by training simple models on simple data-sets and evaluating how the counterfactuals generated by our model compare to those generated by some existing methodologies for generating counterfactuals. While this is not an exhaustive comparison, we have chosen some methodologies with existing publicly available implementations as valuable points of comparison.

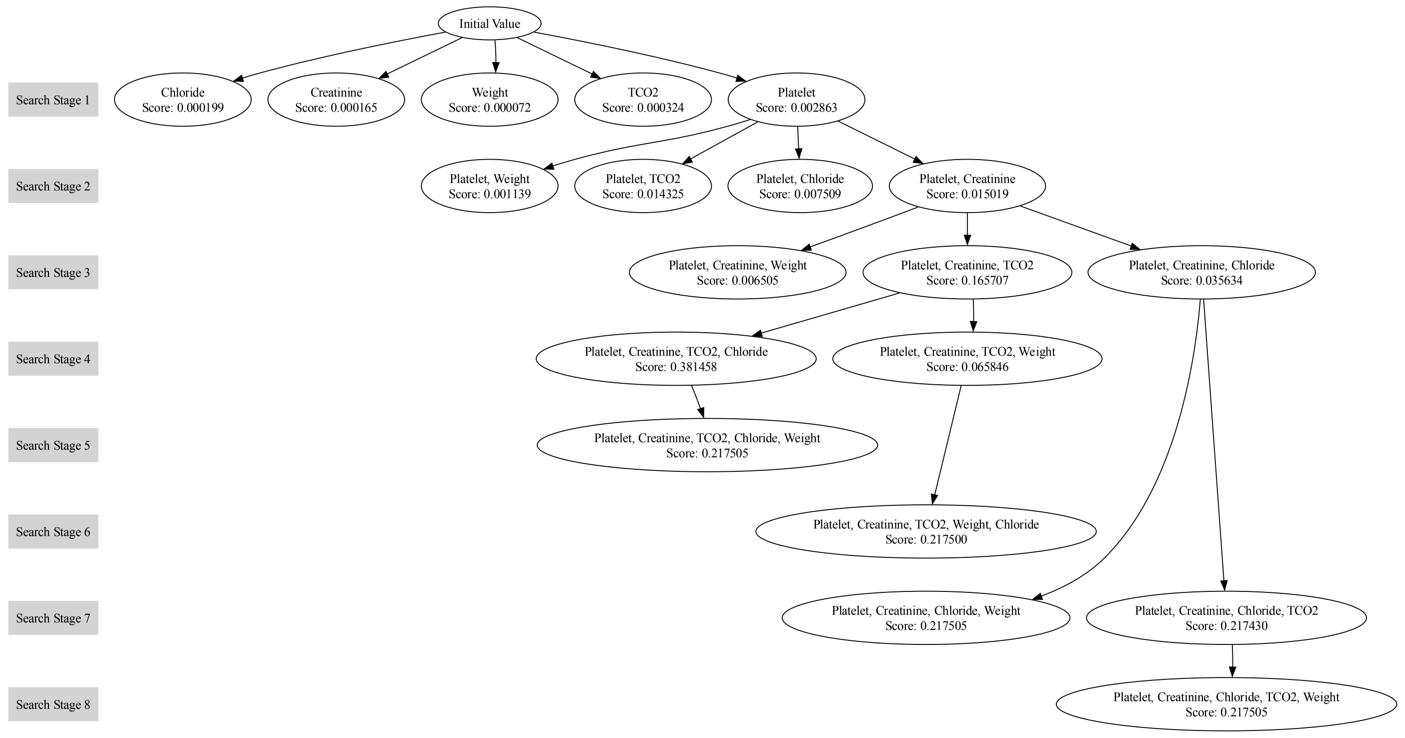

Initial Value
Search Stage 1
Chloride Score: 0.000199
Creatinine Score: 0.000165
Weight Score: 0.000072
TCO2 Score: 0.000324
Platelet Score: 0.002863
Search Stage 2
Platelet, Weight Score: 0.001139
Platelet, TCO2 Score: 0.014325
Platelet, Chloride Score: 0.007509
Platelet, Creatinine Score: 0.015019
Search Stage 3
Platelet, Creatinine, Weight Score: 0.006505
Platelet, Creatinine, TCO2 Score: 0.165707
Platelet, Creatinine, Chloride Score: 0.035634
Search Stage 4
Platelet, Creatinine, TCO2, Chloride Score: 0.381458
Platelet, Creatinine, TCO2, Weight Score: 0.065846
Search Stage 5
Platelet, Creatinine, TCO2, Chloride, Weight Score: 0.217505
Search Stage 6
Platelet, Creatinine, TCO2, Weight, Chloride Score: 0.217500
Search Stage 7
Platelet, Creatinine, Chloride, Weight Score: 0.217505
Platelet, Creatinine, Chloride, TCO2 Score: 0.217430
Search Stage 8
Platelet, Creatinine, Chloride, TCO2, Weight Score: 0.217505

Figure 2: Example of the default search process we have implemented.

As our framework is available as an easily installable Python package, this accurately compares what other options domain experts will have at their disposal to explain machine learning models. We will use qualitative and quantitative metrics to evaluate our counterfactuals compared to those generated by other models.

2.2.1 Evaluation Metrics

In terms of quantitative metrics, we first examine speed. An ideal algorithm to generate counterfactuals will be able to generate them quickly. Of course, this is not a perfect metric — as different technical implementations of the algorithm will lead to different run times. Similarly, different hardware may allow the algorithm to perform at different speeds. However, studying this gives us a rough estimate of the algorithm's efficiency. We conducted all our experiments on a 2022 M2 MacBook Pro with 24 GB of RAM.

Additionally, as we want our counterfactual to be "close" to the original data point, we compare the distance to the original data point in terms of both the L_1 and L_2 norms. For these distance metrics, we normalize them relative to the variation of that feature present in the training data. We define the standardized L_k distances as:

$$L_k^{\text{Standardized}} = \sqrt[k]{\sum_{i=1}^{N} \left(\left(\frac{\left| x_i^{\text{new}} - x_i^{\text{old}} \right|}{\sigma_i^{\text{Training Set}}} \right)^k \right)} \tag{9}$$

Similarly, we want our counterfactuals to change as few features as possible, so we also compare the number of feature changes. Furthermore, we often want our counterfactual to be "plausible" — coming from a distribution similar to the original training data set — so we shall also measure the standardized L_2 distance to the nearest training data point.

Some of our qualitative metrics will include the ones used by Schut et al. [2021] . We want our counterfactuals to be both "unambiguous" and "realistic". However, we also want them to reflect the model accurately. We also qualitatively examine what we "learn" about the model from the counterfactuals, and how "useful" such a counterfactual may be to a user.

2.2.2 Evaluation Data-sets

As we discussed in Section 1.2.2, these counterfactual explanations can be useful in financial and medical situations, so we evaluate our methodology on these. Since our methodology can be easily visualized on image datasets, we also evaluate it on this.

MIMIC The MIMIC (Multi-parameter Intelligent Monitoring in Intensive Care) II data set [Saeed et al., 2011] contains physiological signals and vital signs time series captured from patient monitors, along with comprehensive clinical data for tens of thousands of Intensive Care Unit (ICU) patients. We use a "pre-processed", smaller version of this data set produced by Raffa [2016] , which contains information about the patient, such as age, gender, weight and BMI, information as to if and when the patient entered ICU, information about the patient's clinical and medical history.

MNIST The MNIST data set is a large data set of labelled handwritten digits. It contains 70000 individually labelled 28 by 28 images with pixel values being integers from 0 to 255. This handwritten digit data set is standard across research in this field Verma et al. [2022] , presenting a valuable point of comparison.

2.2.3 Training the simple models

For the medical datasets, we train "Multi-layer Perceptron" (MLP) Classifiers[2] with two hidden layers of size 20 to evaluate our explanation methodology on. We select a subset of useful columns in the dataset for this purpose.

Similarly, for our image dataset, we use a MLP with hidden layers of size 100 and 200.

3 Results and Discussion

3.1 Medical Data-set

By running our methodology using "gaussian process" surrogate models, we can example how our proposed method works on medical data. Consider the patient described in Table 1. Figure 3 shows the counterfactuals generated by our methodology. These counterfactuals are "plausible", i.e. taking "realistic" values for the variables present[3]. They are also "unambiguous" — they clearly show a change in classification, increasing the probability of death from 0.0 to 0.99.

This presentation of our results highlights a key benefit of our methodology, that it creates "sequential" counterfactuals. This means that it gives a series of "steps" that a user can take to change the model's output. This is particularly useful in medical situations, as these can often be actionable where a user can take these steps to improve the patient's health. We can use changeability scores to "tailor" our results here, by assigning higher scores to the features we want

[2]A multi-layer perceptron is a supervised learning algorithm made up of a feed-forward neural network with three types of layers — input, hidden and output. These layers are usually fully connected. Each layer is usually followed by some non-linear activation function to allow the method to model non-linear relationships.

[3]We discuss possible correlation within the variables making this not "realistic" in our Future Work section.

Feature	**Value**
age	71.317250
bmi	28.058252
weight_first	43.600000
icu_los_day	1.910000
hospital_los_day	22.000000
day_icu_intime_num	7.000000
hour_icu_intime	5.000000
map_1st	87.000000
hr_1st	71.000000
temp_1st	97.400002
spo2_1st	100.000000
abg_count	2.000000
wbc_first	7.200000
hgb_first	11.700000
platelet_first	209.000000
sodium_first	139.000000
potassium_first	2.900000
tco2_first	26.000000
chloride_first	101.000000
bun_first	6.000000
creatinine_first	1.000000
day_28_flg	**0.000000**

Table 1: (Processed) MIMIC Patient data for a patient who did not die within 28 days of entering ICU.

to change. For example, we can assign a higher score to the "weight" feature, as this is something that is easier to change than the "age" feature. As these scores would be different for each patient, our method allows for a domain expert to tune this "human-in-the-loop" methodology.

3.2 Image Data-set

By running our methodology with "linear" surrogate models, and allowing features to take values between the 10% and 90% quantiles of the training data, we can generate counterfactuals for the MNIST data set. We can see some examples of these in Figure 4.

These counterfactuals are giving us an insight into how the model is making its classification decisions. In some sense it begins to "fill-in" the 0 to make it look more like a 1.

Similarly, running the previously described existing methodologies on this ex-

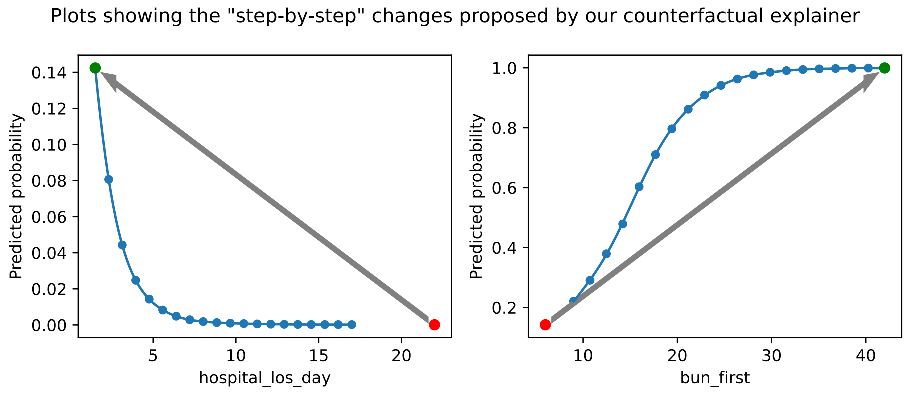


Figure 3: Example counterfactual generated with our proposed methodology. Each image shows one of the "steps" suggested by our methodology.

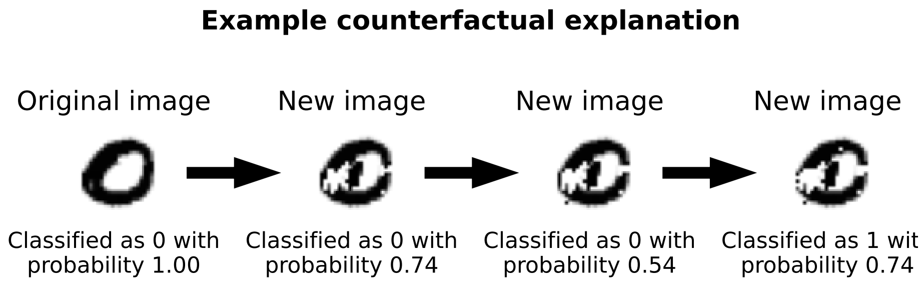

Example counterfactual explanation
Original image
New image
New image
New image
Classified as 0 with probability 1.00
Classified as 0 with probability 0.74
Classified as 0 with probability 0.54
Classified as 1 wit probability 0.74

Figure 4: Example Example counterfactual generated with our proposed methodology. The first picture shows the original image, with the final showing the counterfactual. The middle images show some of the "steps" taken to get from the original image to the counterfactual.

ample, lead to different, almost "advserarial" counterfactuals. These are shown in Figure 5[4].

We can see a clear qualitative difference in types of counterfactual generated between our methodology and the two "comparison" methods. In a sense, our method works more to "transform" the image from a 0 into a 1, while the other methods work to "perturb" the image from a 0 into a 1. Since our method "chains together" single pixel changes, there is an "inbuilt" incentive for our method to change fewer pixels (as each change is "costed" by the scoring function). This is not the case for the other methods, which can change many pixels at once. This can lead to our counterfactuals being more human-interpretable.

Similarly, our use of changeability scores can be used as an alternative method to guide our counterfactuals to be more "interpretable". For example, when generating our MNIST counterfactuals, we want to "change" pixels "close" to the digit. We can do this by giving these pixels a higher changeability score. We do this by "blurring" the image, and applying a threshold of 0.05 to these scores. This is shown in Figure 6. In this case, we did not apply any advanced "feature bounds" rules — we simply allowed the features to take any value between 0 and 255. This continues to generate counterfactuals that are "close" to the original image, but also "interpretable", demonstrating the flexibility of our methodology. Similarly, our counterfactuals are "unambiguous" — they demonstrate a clear "change in classification".

3.3 Quantitative Evaluations

Figures 7 and 8 contains violin plots[5] of the various quantitative metrics defined in Section 2.2.1.

As shown in these figures, compared to comparable methods, our proposed methodology changes fewer features, but changes them by a larger amount (both in terms of L_1 and L_2 distance). In a sense, the counterfactuals are "more interpretable" as they change fewer features.

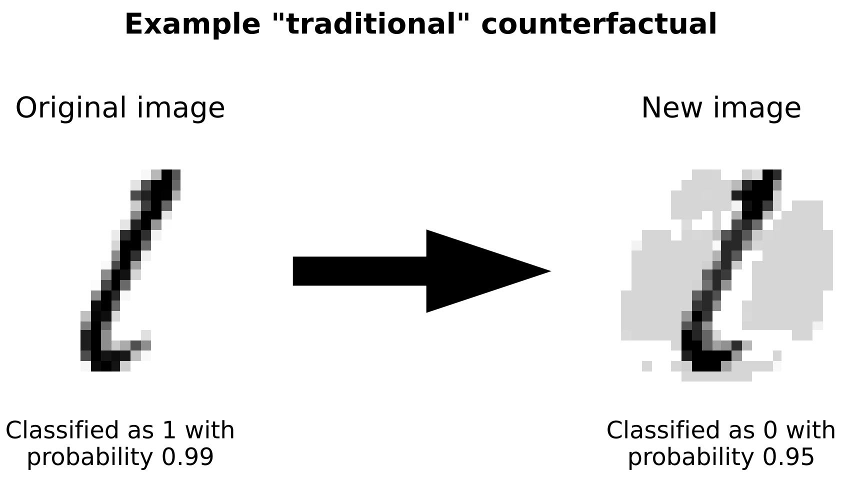
4 Conclusions

4.1 Key Contributions

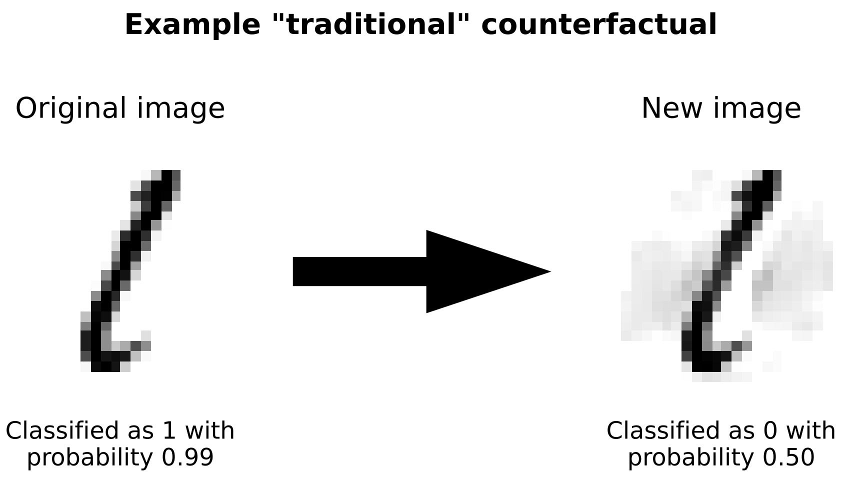
In the project, we proposed a modular methodology to generate counterfactuals to explain black box models. This method borrows ideas from Human-Computer Interaction (HCI) research to develop a "tuneable", "human-in-the-loop" explanation method for machine learning models. We evaluated the model's performance on a range of data sets and examined variations of our method.

[4]These were also generated with "quantile" feature bounds, but rather using 5% and 95% bounds as theses methods were unable to find any counterfactuals in the $10\% - 95\%$ range.

[5]Violin plots are similar to box plots, but also show the distribution of the data


Example "traditional" counterfactual
Original image
New image
Classified as 1 with probability 0.99
Classified as 0 with probability 0.95

(a) Example counterfactual generated with the methodology of Wachter et al. [2017]


Example "traditional" counterfactual
Original image
New image
Classified as 1 with probability 0.99
Classified as 0 with probability 0.50

(b) Example counterfactual generated with the methodology of Van Looveren et al. [2021]

Figure 5: Example counterfactuals generated with other methodologies. The first picture shows the original image, with the final showing the counterfactual.

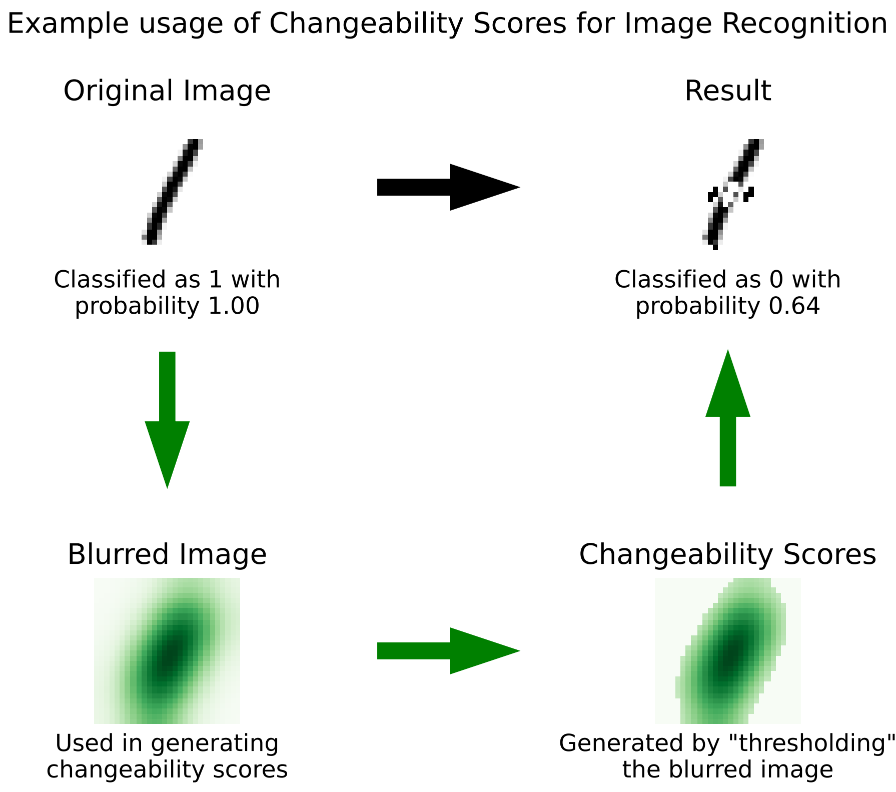

Example usage of Changeability Scores for Image Recognition
Original Image
Classified as 1 with probability 1.00
Result
Classified as 0 with probability 0.64
Blurred Image
Used in generating changeability scores
Changeability Scores
Generated by "thresholding" the blurred image

Figure 6: Example Example counterfactual generated with our proposed methodology. The first picture shows the original image, with the final showing the counterfactual. The middle images show some of the “steps” taken to get from the original image to the counterfactual.

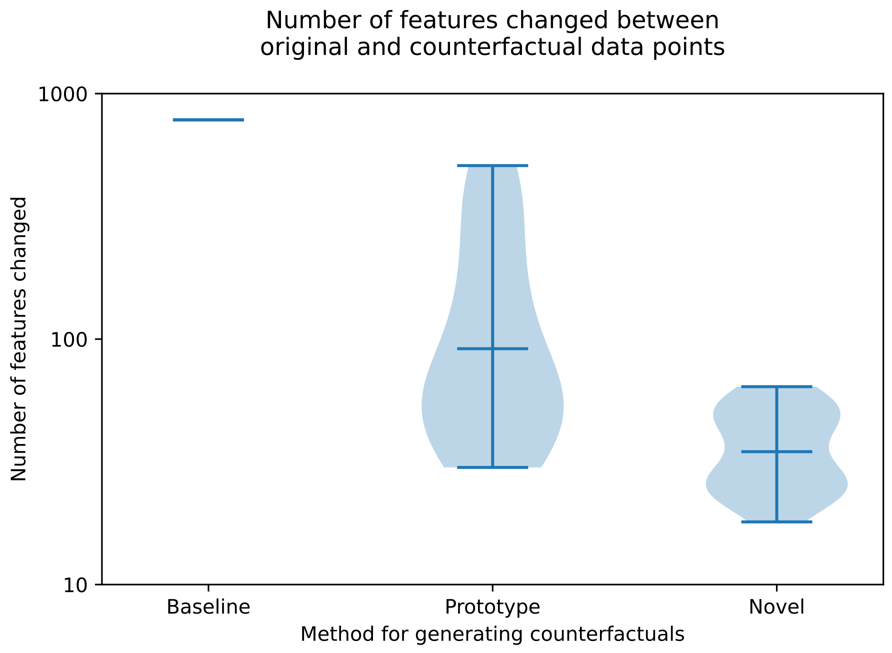


Figure 7: Violin plots of the number of feature changes for each method. The "traditional" method is that of Wachter et al. [2017], while the "prototype" method is that of Van Looveren et al. [2021].

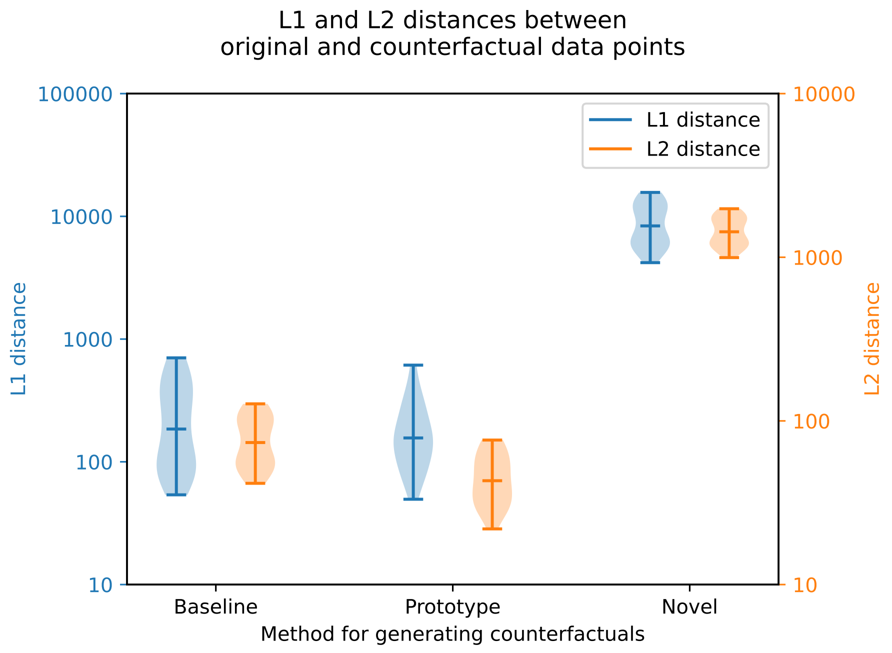


Figure 8: Violin plots of the L_1 and L_2 distances to the original data point for each method. The “traditional” method is that of Wachter et al. [2017], while the “prototype” method is that of Van Looveren et al. [2021].

References

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975. ISSN 0001-0782, 1557-7317. doi: 10.1145/361002.361007. URL https://dl.acm.org/doi/10.1145/361002.361007.

Alan Blackwell. Bias and Fairness, November 2022. URL https://www.cl.cam.ac.uk/teaching/2223/IML/IWML-2022-fairness-bias.pdf. Place: University of Cambridge.

Davide Castelvecchi. Can we open the black box of AI? *Nature News*, 538 (7623):20, October 2016. doi: 10.1038/538020a. URL http://www.nature.com/news/can-we-open-the-black-box-of-ai-1.20731.

Bryce Goodman and Seth Flaxman. European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation". *AI Magazine*, 38(3):50–57, October 2017. doi: 10.1609/aimag.v38i3.2741. URL https://doi.org/10.1609\%2Faimag.v38i3.2741. Publisher: Wiley.

Rafia Inam, Ahmad Terra, Elena Fersman, and Aneta Vulgarakis Feljan. Explainable AI: How humans can trust Artificial Intelligence, April 2021. URL https://www.ericsson.com/en/reports-and-papers/white-papers/explainable-ai--how-humans-can-trust-ai.

Jeremy Jordan. Introduction to autoencoders., March 2018. URL https://www.jeremyjordan.me/autoencoders/.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980.

Janis Klaise, Arnaud Van Looveren, Giovanni Vacanti, and Alexandru Coca. Alibi Explain: Algorithms for Explaining Machine Learning Models, June 2021. URL http://jmlr.org/papers/v22/21-0017.html.

Christoph Molnar. *Interpretable Machine Learning*. Independently published, March 2023. URL https://christophm.github.io/interpretable-ml-book/logistic.html.

Office of Ron Wyden. Algorithmic Accountability Act of 2022 One-Pager. Technical report, Office of Ron Wyden, February 2022.

Jesse Raffa. Clinical data from the MIMIC-II database for a case study on indwelling arterial catheters:, 2016. URL https://physionet.org/content/mimic2-iaccd/1.0/.

Mohammed Saeed, Mauricio Villarroel, Andrew Reisner, Gari Clifford, Li-wei Lehman, George Moody, Thomas Heldt, Tin Kyaw, Benjamin Moody, and Roger Mark. MIMIC-II Clinical Database, 2011. URL https://physionet.org/content/mimic-ii/2.6.0/.

Lisa Schut, Oscar Key, Rory McGrath, Luca Costabello, Bogdan Sacaleanu, Medb Corcoran, and Yarin Gal. Generating Interpretable Counterfactual Explanations By Implicit Minimisation of Epistemic and Aleatoric Uncertainties, March 2021. URL http://arxiv.org/abs/2103.08951.

Arnaud Van Looveren, Janis Klaise, Giovanni Vacanti, and Oliver Cobb. Conditional Generative Models for Counterfactual Explanations, January 2021. URL http://arxiv.org/abs/2101.10123.

Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

Sahil Verma, Varich Boonsanong, Minh Hoang, Keegan E. Hines, John P. Dickerson, and Chirag Shah. Counterfactual Explanations and Algorithmic Recourses for Machine Learning: A Review, November 2022. URL http://arxiv.org/abs/2010.10596.

Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR, October 2017. URL https://papers.ssrn.com/abstract=3063289. Place: Rochester, NY Type: SSRN Scholarly Paper.