Master Thesis

Path regularization for continuous counterfactual explanations

Author:
Auguste Baum

Supervisors:

From EPFL:
Prof. Pascal Frossard

From Swisscom AG:
Dr Daniel Dobos
Vincent Coriou
Natalie Bolón Brun

Signal Processing Laboratory (LTS4)
School of Computer and Communication Sciences (IC)
Swisscom Digital Lab, Swisscom AG

Lausanne, March 17, 2023

"Nulle montagne sans vallée."

French proverb

Abstract

Auguste Baum

Path regularization for continuous counterfactual explanations

The problem of explainability hinders the use of machine learning: in some applications justifying a model's decisions is critical to ensure trustworthiness, while in others, it is set to become a legal necessity. For a given input-output pair, counterfactual explanations address this need by providing an answer to the question "What would it take for the model to change its mind?". Such an explanation clearly shows which features are determinant to the model, but depending on the circumstance, it also shows an end-user how to obtain a more desirable model outcome. However, if the counterfactual looks too different from its original point, this can affect the trustworthiness of the explanation itself: unfortunately, the current methods usually do not display the *progression* that leads from the explained input to its counterfactual.

Latent Shift was introduced to address this issue: in this technique, the input is perturbed *continuously* towards a target class. However, so far this technique has been applied mostly to binary classification problems on images, and for good reason: image explanations are easily understandable by humans, so objective quality metrics matter less than for other domains, such as tabular data.

Thus, our contributions are as follows: firstly, we apply Latent Shift to (numerical) tabular data. This leads us to search for objective metrics for the quality of counterfactuals, because rows of data cannot be judged intuitively like images can. Secondly, we extend Latent Shift to multi-class problems, where targeting a particular class is not the same as simply moving a point away from its current class; we experiment with different loss functions to maximize the probability of reaching the target class. Thirdly, we propose a variation to Latent Shift to ensure paths satisfy certain interpretability constraints, e.g. the path should only go through *realistic* points. Finally, we study the robustness of our proposed method by using it with nonsensical constraints.

Acknowledgements

This project was very much facilitated by a number of people, and here are their well-deserved thanks. Thank you to Prof. Frossard from EPFL and to Dr Daniel Dobos from Swisscom AG for taking on the supervision of this project and for helpful comments. Thank you to everyone at the Swisscom Digital Lab for welcoming me as an intern, and making me feel at home; many thanks to all the other Master's students, especially Karim, Marie and of course Arnaud for well-deserved breaks and insightful discussions, often at the same time; a special thank you to Vincent and Nati for their precious guidance on what it means to do research, and a very large heart-shaped thank you to Constance, my parents and loved ones for engaging with me on my project, being there for me when I needed them, and always pushing me to take a step back and understand the ramifications of this research.

I wish you all the best.

Contents

List of Abbreviations

CF	**C**ounter**F**actual
CFX	**C**ounter**F**actual e**X**planation
NF	**N**ormalizing **F**low
NLL	**N**egative **L**og-**L**ikelihood
VAE	**V**ariational **A**uto**E**ncoder
XAI	e**X**planainable **A**rtifical **I**ntelligence

Chapter 1

Introduction

1.1 Context

1.1.1 Explainable AI

The term "Explainable AI" refers to the endeavor of extracting human-friendly *reasoning* from an AI system, either by a post hoc procedure or by guiding its learning process to give it the ability to produce said reasoning [70].

Some of the reasons driving research in explainable AI (referred to as XAI hereafter) are summarized in [70]:

- Scientists that use AI hope to recover human-understandable information about their problem of interest (e.g. in drug design or physics);
- Regulators hope to make AI-based decisions accountable, especially those that affect data subjects directly (e.g. in insurance, banking, or criminal justice);
- AI practitioners hope to limit unexpected failure scenarios in their work, especially in cases where failure can be costly (e.g. in medicine or finance).

Because of how ubiquitous AI is becoming, XAI research is also blossoming, and the research landscape is vast. Accordingly, some taxonomies have emerged recently that help differentiate methods [70, 3]. In particular, as mentioned, there exist methods that alter the training process of the model of interest to make it more interpretable; these are called *embedded* methods. On the other hand, many methods are designed to be applied to pre-trained models in a frozen state – the models can only be probed, not modified. These methods are referred to as *post hoc* (literally "after the fact"). In this work, we focus on the more general-purpose *post hoc* methods.

One other key characteristic of an XAI method is its *scope*: a *global* XAI method applies to a *whole* fully trained ML model, while a *local* method gives an explanation of *one prediction* by the model. Global methods produce compact results, but they report general information that does not necessarily reflect how the model acts in all situations, while local methods produce results with a more limited range, which

are more precise but can lead to misleading interpretations if applied to far-removed situations.

In the following, we focus on local explainability methods, relying on the assumption that aggregating many local explanations can help us grasp the model's general behavior.

1.1.2 Local explainability

Setting aside global methods, we can still subdivide local methods into several types of explanation. Some of them are outlined in [3]:

Local feature attribution approaches

A popular format for explanations is *feature attributions*: ratings of the importance of each input feature, which give an answer to the question "Which features matter to the model for its prediction?". Broadly speaking, there are two types of methods outputting feature attributions: perturbation- and gradient-based.

Perturbation-based methods One strategy is to find an approximation of the model: a simpler, more interpretable proxy model which has the same local behavior. In `LIME`, a commonly used method, the proxy is a linear model [54]. With the proxy model, it is often straightforward to extract information such as feature attribution: in the case of a linear model, the weight of each feature in the prediction is related to the magnitude of the regression coefficients.

However, finding a proxy model can take a long time, and the proxy would only hold for one point of interest. Similarly, finding the approximation requires access to at least some training data points, which means that part of the dataset needs to be available on the same machine where the proxy model is generated – which increases memory consumption and constitutes a potential privacy issue. Worse, it has been shown that `LIME` and other methods can be fooled into proposing explanations that do not accurately reflect the model behavior [58].

Gradient-based methods Another strategy involves computing the gradient of the classification function with respect to the input features. Indeed, by definition of the gradient, this precisely measures how sensitive the prediction is to a change in one of the input features. One of the most frequently used methods of this kind is `IntegratedGradients` [60], which answers the question: "How relevant to the prediction is feature i of the point of interest x compared with some other reference point x'?".

Notice that the explanation is only given for the reference point x', so choosing a different x' can yield a different explanation; furthermore, the choice of x' is not always obvious.

Interestingly, it has recently been shown that many pre-existing local explanation methods, including `LIME` and `IntegratedGradients`, can be formulated as instances of the same framework called Local Function Approximation [24].

Rule-based approaches

Rule-based techniques aim to approximate the model's behavior by simple logical rules on the input features that hold locally (i.e. for a high proportion of other input points in the neighborhood of the input of interest). As such, they can be understood as a special case of local approximation approaches. One rule-based approach, called `Anchors`, does this by reformulating the problem as a multi-armed bandit problem [55]. The advantage of rules as explanations is their intelligibility: it has been shown that rules depicting why a model behaved a certain way tend to be well understood compared to other types of explanation [43].

Counterfactual approaches

A counterfactual explanation (CFX hereafter) is one or more *virtual points* that are close to the explained input, but for which *the model prediction is different*. Such an explanation aims to answer the question: "What would have to change in the input for the model to change its prediction?" This is reminiscent of adversarial attacks, which consist in finding small perturbations that "trick" the model into making a nonsensical prediction [61, 46]. The differences between an adversarial perturbation and a CFX are subject to debate, and recently efforts have been made to reconcile the two research areas [22]. Some authors consider that while adversarial perturbations can be imperceptible, CFX should be sparse yet easily visible perturbations [39]. The argument could also be made that CFX also have a realism constraint [63], although that can be the case for adversarial perturbations too: if the perturbation is imperceptible, then the perturbed input is realistic since it is close to a real input point.

CFX provide value in two keys ways [56]:

- they indirectly show which features are being used by the model in its decisions, and
- they constitute a way to provide *recourse* when the model is used in a setting where recourse is required or desirable (for example in the context of a loan application).

The subject of this work will be this last kind of local explanation, and we will describe it further in the next section.

1.2 Counterfactual explanations

1.2.1 A first example: Wachter, Mittelstadt, and Russell

One of the first and most frequently cited sources on CFX is Wachter, Mittelstadt, and Russell [65]. In this work, the problem is formulated as an optimization problem constrained by the distance between the explained input x and a candidate counterfactual x':

$$\arg\min_{x'} \min_{\lambda} \{\lambda(f(x') - y')^2 + d(x, x')\} \tag{1.1}$$

where f is the model of interest, y' is the target model output (different from the factual output $y = f(x)$) and d is the aforementioned distance metric, which depends on the use-case.

The authors view CFX as an adequate solution to the "right to explanation" requirements set by the GDPR [30]. Furthermore, in certain contexts such as loan applications, a CFX could constitute a useful piece of advice that can be taken advantage of in a future trial: the candidate that sees their application denied can tell what they should do to improve their chances.

Yet, while CFX seem helpful in improving trust in black-box machine learning models, further thought brings to light difficult problems that threaten the trustworthiness of the explanations *themselves*.

1.2.2 Formalizing counterfactual explanations

As we saw when describing local explainability techniques, there are several competing forms of local explanation. The field of XAI in general has suffered from a crisis of formalism [44, 41]. In particular CFX have a unique set of issues relating to rating explanations, that set it apart from its siblings.

The concurrent goals of counterfactual explanations

As noted by the authors of [65], one appeal of CFX is their ability to provide a large quantity of information in a seemingly compact form: a CFX can serve as a local feature attribution rating, highlighting which features the model used to differentiate between classes in a specific prediction, and it can also enable stakeholders to offer recourse when this is appropriate. However, these two coupled objectives can sometimes conflict and make CFX appear less trustworthy as a result.

For example, consider the problem of allowing protected attributes to be modified. In a hypothetical loan application setting, consider a model that takes a set of personal information about a candidate and predicts if they are likely to default on their loan. In particular, suppose some protected attribute is being used in the prediction, such as ethnicity or gender. If a CFX system were used for recourse, it might be sensible to set it up to completely ignore such protected attributes: could we expect a person

to change their gender in order to increase their chances of receiving a loan? On the other hand, say a CFX system were used for testing the model's learning. If the protected attribute is frozen in place, then by definition it will never being identified as useful for the model's decision, *even if the model actually makes use of it*. In other words, if a CFX displays that the protected attribute is unchanged, in the absence of other information, it is not possible to tell if the model is unbiased, or if the CFX system is unbiased.

The out-of-distribution problem

One major drawback with CFX is that, since they generate virtual points, there is a risk that these new points look realistic, yet quite different from the data used to train the model. In this case the model could generalize poorly and the explanation could therefore be misleading. Worse, the new points could look completely unrealistic (e.g. containing logical inconsistencies, such as a person having a negative age), in which case the explanation might be difficult or impossible to interpret.

Determining whether a given point in the input space is "realistic" is a problem in itself, and is studied through various lenses from different disciplines of data analysis [67]. In the following we will refer to this as the "out-of-distribution" problem.

Semantic explanations

An aim of CFX would be that, if the model has recovered some meaningful information about the data, then the explanation should reveal it. That is to say, we would like to trust that, if the provided explanations are gibberish, then this must mean that what the model learned is not meaningful to a human, at a glance. Of course, this raises a highly philosophical issue: what is meaning? If a concept means nothing to you, then by definition you could easily miss it and move on. Therefore, if the explanation presents us with what seems to be gibberish, it is not possible to distinguish gibberish that actually constitutes interesting concepts from random noise that truly contains no information.

1.2.3 On the differences between image and tabular data

In the field of CFX, as everywhere in deep learning, the problem domain is an important consideration. In particular, the question of realism of the generated points has been approached differently depending on whether the data is in tabular form or in image form. Indeed, some advances have been made on metrics for realism of generated images: for example, the Fréchet inception distance (or FID) is the current state-of-the-art for metrics of image quality [26].

Additionally, when discussing the effectiveness of deep learning for image problems, one argument that is often invoked is the manifold hypothesis [4]. This states that for many tasks involving real-life high-dimensional data, the data generally do not span

across the whole input space $\mathcal{X}$; rather, they lie close to a low-dimensional manifold contained within it. In particular, images have intrinsic structure to them, for example it is often the case that rotating an image does not change the meaning it carries. This structure can be encoded within a neural network, e.g. with convolutional layers [69, 42], and doing this tends to achieve high success rates. Recently this idea of encoding the problem structure within a neural network architecture has been discussed in detail in [7]. Note that the manifold hypothesis was recently put back into question, with the authors of [8] instead proposing that the data lie on a union of manifolds.

By contrast, no such hypothesis has been made for tabular data, where feature engineering is a much more important factor in model performance. Accordingly, the state-of-the-art on tabular data is methods based on decision trees [57].

Now that we have presented the overall problem and its stakes, let us detail the technical background needed for our analyses.

Chapter 2

Technical background

In this chapter we present some technical notions that will be relevant in our analysis, and introduce some notation.

2.1 Preliminaries

For a vector x in $\mathbb{R}^N$ we write x_i for the i^{th} component of x; if the vector is written with a subscript already, e.g. y_{true}, we can also denote the i^{th} component by $y^{(i)}_{\text{true}}$.

The Kronecker delta function is defined as

$$\delta_{x,x'} = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$$

$y_i \in \mathbb{R}^N$ can also refer to an N-component vector that is 1 in position i and 0 everywhere else, so $y_i = (\delta_{ij})_{j=1}^N$.

A norm is a measure of magnitude; there are many existing norms, so when this is undetermined the norm of a vector x is often written $\|x\|$. The L_p norm is an example of a norm and is usually written $\|\cdot\|_p$. It is computed as $\|x\|_p = \sqrt[p]{\sum_i |x_i|^p}$ where x_i is the i^{th} component of x.

Let $F(x_1, \ldots, x_n)$ be a function of n variables outputting a scalar, differentiable with respect to each variable. In multivariable calculus, the *partial derivative* of F with respect to x_i generalizes the usual single-variable derivative and is written $\frac{\partial F}{\partial x_i}$.

The *gradient* of F is the vector

$$\nabla_x F = \begin{pmatrix} \frac{\partial F}{\partial x_1} & \cdots & \frac{\partial F}{\partial x_n} \end{pmatrix}$$

When F is vector-valued with output components $F_1, \ldots, F_p$, the equivalent of the gradient is the *Jacobian matrix*:

$$J_F(x) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_p}{\partial x_1} & \cdots & \frac{\partial F_p}{\partial x_n} \end{pmatrix}$$

2.2 Classifier

Consider the problem of detecting an object in a grayscale image. This problem can be formalized as follows.

Let there be an *input space* $\mathcal{X}$ with dimensionality D, typically $\mathbb{R}^D$ in our analysis. For the object detection task, $\mathcal{X} = \{\text{image with } D \text{ pixels}\}$.

Let there be a set of classes C containing C classes. For object detection $C = \{\text{object}, \text{no object}\}$ or $\{0, 1\}$.

A *classifier* is a function $f : \mathbb{R}^D \to \mathbb{R}^C$. Usually the outputs of the classifier are referred to as *logits*, written $u \in \mathbb{R}^C$, and they are further processed to obtain a probability mass using the $\mathrm{softmax}$ function:

$$\mathrm{softmax}(u) = (\mathrm{softmax}_i(u))_{i=1}^C \text{ where } \mathrm{softmax}_i(u) = \frac{\exp u_i}{\sum_{j=1}^C \exp u_j} \tag{2.1}$$

We use the notations $\mathrm{softmax}_i(x)$ and $\mathrm{softmax}(x)_i$ interchangeably. When it is unambiguous we also write $p_i = \mathrm{softmax}_i(x)$ for brevity.

A classification task consists in finding a classifier that fits real-life data as best as possible: given many examples of images $x \in \mathcal{X}$ labelled as $y_{\text{true}} \in \{0, 1\}$ depending on if the object of interest is present in x, find f such that $\mathscr{L}(f(x), y_{\text{true}})$ is small for all x, where $\mathscr{L}$ is some distance metric, often called a *loss*.

For example, when the training example is $(x, \text{no object})$ we encode it as $(x, 0)$ and the ideal probability mass p would be $(1, 0)$ (a 1 in position 0 and zeros elsewhere); when the training example is (x', object) we encode it as $(x', 1)$ and then the ideal p would be $(0, 1)$ (a 1 in position 1 and zeros elsewhere).

In our analysis, as in deep learning in general, we typically let $f = f_w$ be a deep learning model with trainable parameters w. Therefore, the classification task is reduced to finding the set of parameters w that minimizes the loss $\mathscr{L}$.

Finding the optimal w is usually done with an iterative optimization algorithm that minimizes $\mathscr{L}$, typically *gradient descent*. Gradient descent consists in computing the gradient of $\mathscr{L}$ with respect to w, $\nabla_w \mathscr{L}$, which gives the "direction of steepest ascent" of $\mathscr{L}$. Then, the model parameters are adjusted slightly in the *opposite direction* of the gradient (to make $\mathscr{L}$ decrease in subsequent training steps): $w_{t+1} = w_t - \nabla_w \mathscr{L}$. The

loss we use to train our classifiers in this work is the *cross-entropy* CE. Given logits output by f and the true class label c, the cross-entropy is computed as:

$$\mathrm{CE}(f(x), c) = -\log \mathrm{softmax}_c(f(x)) \tag{2.2}$$

A *multi-layer perceptron*, or MLP for short, is a classic example of a deep learning model architecture. It consists in several *layers*; in its most basic form, a layer consists in a linear transformation, followed by a nonlinear *activation function*. One of these activation functions is the *rectified linear unit*, usually abbreviated to ReLU, and defined as $\mathrm{ReLU}(x) = \max\{0, x\}$.

When optimizing the w (training the model), we do not iterate on the full dataset $\mathscr{D}$. We first separate it into a training set, a validation set and a test set, denoted by $\mathscr{D}_{\text{train}}$, $\mathscr{D}_{\text{validation}}$ and $\mathscr{D}_{\text{test}}$ respectively. The optimization algorithm is run on the train set repeatedly; an *epoch* corresponds to one full on the whole train set, and after every epoch we compute the validation error by running the model on $\mathscr{D}_{\text{validation}}$. When this validation error stops decreasing, this is a sign that the training process can be stopped. Finally, we test how well our model generalizes to unseen examples by running it on $\mathscr{D}_{\text{test}}$.

2.3 Autoencoder

A *generative model* is a function g that can generate points in $\mathcal{X}$ that look similar to points in a dataset; finding such a model is equivalent to finding the distribution of points in the dataset, denoted by $P(x)$.

In particular, a *latent variable generative model* is a mapping $\mathrm{Dec} : \mathcal{H} \to \mathcal{X}$ that allows modeling the data distribution $P(x)$, which is an approximation of the true distribution from which the data originate. To achieve this, we assume that a given $x \in \mathcal{X}$ has been generated by drawing a latent variable z from a known distribution with support in $\mathcal{H}$ and then setting $x = \mathrm{Dec}(z)$. $\mathcal{H}$ is called the *latent space* and its dimensionality is written H.

In simple terms, consider the problem of generating realistic handwritten digits. In order to generate such an image, one could start by sampling a digit z from 0 to 9 uniformly and then run $\mathrm{Dec}(z)$ to produce an image that looks like that digit, in handwritten form. This example is adapted from a tutorial by Doersch [16],

A subclass of generative models is that of *autoencoders*. An autoencoder is a pair of functions, commonly referred to as the "encoder" and "decoder" and denoted by $(\mathrm{Enc} : \mathcal{X} \to \mathcal{H}, \mathrm{Dec} : \mathcal{H} \to \mathcal{X})$ implicitly defining a latent space $\mathcal{H}$, with the constraint that the reconstruction error $\|x - \mathrm{Dec}(\mathrm{Enc}(x))\|$ be as small as possible. It is common for us to refer to points in latent space as "latents". In our work we make

use of autoencoder models that are based on deep neural networks and, like for classifiers are learned by optimizing their parameters: the encoder is written Enc_ϕ and the decoder is written Dec_ψ; when it is not ambiguous we omit the subscripts. In addition, for our purposes it is not sufficient that the reconstruction error be minimized: we require continuity of the decoder, so that mapping a straight line in $\mathcal{H}$ through Dec results in a continuous path in $\mathcal{X}$.

2.3.1 Variational autoencoder

One way to meet this continuity requirement is with a *variational autoencoder* (or VAE for short) [34]. This is an autoencoder model in which we learn a distribution $Q(z|x)$ that corresponds to the likelihood of a z giving rise to a particular x. This distribution is assumed to be a standard Gaussian parametrized by x: $Q(z|x) = \mathcal{N}(\mu(x), \Sigma(x))$ where $\mu(x)$ and $\Sigma(x)$ are learnable networks that output respectively the mean and the covariance matrix of the Gaussian. We enforce this assumption by minimizing the KL-divergence $\mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\mu(x), \Sigma(x)) \| \mathcal{N}(0, I))$. The encoder Enc is then given by $\mathrm{Enc}(x) = \mu(x)$.

For a more detailed introduction to VAEs we highly recommend the tutorial by Doersch [16].

2.3.2 Flow-based models

Another type of generative model is the *normalizing flow* (or NF for short), in which we assume again a latent variable z that is typically standard Gaussian-distributed and that leads to an x via a transformation g (our Dec function). Usually, this g is decomposed into several small perturbations g_i so that

$$x = g(z) = (g_K \circ g_{K-1} \circ \ldots \circ g_1)(z).$$

where $\circ$ denotes composition of functions. Importantly, each g_i is *invertible*, hence g itself is invertible and one can express and maximize the log-likelihood $\log P(x)$ using the change-of-variable formula:

$$\begin{aligned}
P(x) &= P(z) \left| \det \frac{\mathrm{d}g}{\mathrm{d}z} \right| \\
&= P(z) \left| \det \left(\frac{\mathrm{d}g_K(z_{K-1})}{\mathrm{d}z_{K-1}} \frac{\mathrm{d}g_{K-1}(z_{K-2})}{\mathrm{d}z_{K-2}} \cdots \frac{\mathrm{d}g_1(z)}{\mathrm{d}z} \right) \right| \\
&= P(z) \left| \det \left(\frac{\mathrm{d}g_K(z_{K-1})}{\mathrm{d}z_{K-1}} \right) \det \left(\frac{\mathrm{d}g_{K-1}(z_{K-2})}{\mathrm{d}z_{K-2}} \right) \ldots \det \left(\frac{\mathrm{d}g_1(z)}{\mathrm{d}z} \right) \right| \\
&= P(z) \left| \det \left(\frac{\mathrm{d}g_K(z_{K-1})}{\mathrm{d}z_{K-1}} \right) \right| \left| \det \left(\frac{\mathrm{d}g_{K-1}(z_{K-2})}{\mathrm{d}z_{K-2}} \right) \right| \ldots \left| \det \left(\frac{\mathrm{d}g_1(z)}{\mathrm{d}z} \right) \right| \\
\implies \log P(x) &= \log P(z) + \sum_{i=1}^{K} \log \left| \det J_{g_i(z_{i-1})}(z_{i-1}) \right| \quad \text{(defining } z_0 = z\text{)}
\end{aligned}$$

where $\det$ denotes the determinant of a matrix. We can then train the model by minimizing the *negative log-likelihood* (or NLL for short).

Unlike in VAEs, the transformation g is guaranteed to be invertible, but this requires that the dimensionality of $\mathcal{H}$ be at least D.

Nonlinear independent components estimation

This implementation of a NF first presented in [14], hereafter referred to as NICE, allows crafting flows g_i that are invertible, even if they contain highly nonlinear components. The g_i are referred to as *coupling layers*.

This is achieved by separating z_i into two parts: one left unmodified and the other put through some nonlinear layer with learnable parameters. Then, the two parts are passed to a coupling layer that should be invertible with respect to its first argument given the second. For example, for an *additive coupling layer* this is an addition: $g_i(a; b) = a + b$, which is indeed invertible with respect to a given b: namely, its inverse is $g_i^{-1}(a; b) = a - b$. Thus, by construction, the transformation is invertible regardless of the nonlinearity: in Equation 2.3 we show the relation between layer i and layer $i + 1$, with the nonlinearity in this layer denoted by h_{i+1}. z_i is divided into parts A and B, and it is possible to get z_i back entirely from z_{i+1}:

$$\begin{cases} z_{i+1}^A = z_i^A \\ z_{i+1}^B = h_{i+1}(z_i^A) + z_i^B \end{cases} \Leftrightarrow \begin{cases} z_i^A = z_{i+1}^A \\ z_i^B = -h_{i+1}(z_{i+1}^A) + z_{i+1}^B \end{cases} \tag{2.3}$$

Dinh, Krueger, and Bengio note that this transformation preserves volume (its Jacobian is 1), which can restrict the model's ability to fit the distribution during training, e.g. if some dimensions are found to be completely unimportant. To give the model more flexibility, they introduce scaling parameters S_{ii}. For a standard Gaussian prior,

with rescaling, the negative log-likelihood (NLL) can be derived as follows:

$$\begin{aligned}
\mathrm{NLL}(x) &= -\log p_X(x) \\
&= -\log\left(p_Z(z)\prod_{i=1}^{H}|S_{ii}|\right) \\
&= -\log p_Z(z) - \log\prod_{i=1}^{H}|S_{ii}| \\
&= -\log p_Z(z) - \sum_{i=1}^{H}\log|S_{ii}| \\
&= -\log p_Z(z) - \sum_{i=1}^{H}\log|S_{ii}| \\
&= -\log\left(\frac{1}{\sqrt{(2\pi)^H}}\exp\left(-\frac{z^T z}{2}\right)\right) - \sum_{i=1}^{H}\log|S_{ii}| \\
&= -\log\frac{1}{\sqrt{(2\pi)^H}} - \log\exp\left(-\frac{z^T z}{2}\right) - \sum_{i=1}^{H}\log|S_{ii}| \\
&= \frac{H}{2}\log 2\pi + \frac{z^T z}{2} - \sum_{i=1}^{H}\log|S_{ii}|\,.
\end{aligned}$$

where $z = \mathrm{Enc}(x)$.

2.4 Counterfactual explanations

Consider a classifier f trained to take an input $x \in \mathcal{X}$ and predict a class $c \in C$. We call this predicted class the *source* class and we write it source. A counterfactual explanation for x is a point x^{CF} that is comparable with x but for which the predicted class is different. In some counterfactual methods, it is possible to specify a *target* class that x^{CF} should be predicted as; this class is written target. It can happen that the counterfactual is in fact not predicted as target; in this case we say the counterfactual is not *valid*.

Equipped with these concepts and notation, we can now give an overview of the literature on counterfactual explanation generation methods making use of generative models.

Chapter 3

Previous work

In this chapter we describe some methods for counterfactual explanation (CFX) generation and related topics that were of interest during this project. First we present some areas of research that are closely linked to CFX generation, then we provide an overview on methods that rely on generative models to promote realistic CFX. A comprehensive review of many other CFX generation methods can be found in [63].

3.1 Related topics

3.1.1 Contrastive explanations

Contrastive explanations are close enough to CFX that they are often dealt with together: Stepin et al. even bring them together under the name "contfactual" [59].

Where counterfactual explanations are virtual points close to the explained input, potentially highlighting underlying concepts which are deemed important by the classifier, contrastive explanations are ratings of the importance given by the classifier of a given *user-defined* concept to a given class. Hence, contrastive explanations do not suffer from the problem of semanticity of explanations that was raised in section 1.2.2, but in return it is up to the user to propose meaningful concepts to evaluate.

Accordingly, Jacovi et al. develop a method for text data that evaluates how much a given concept contributes to a certain class compared to another [28]. For instance, they train a model for the "natural language inference" task: given two sequential sentences as input, the model should output whether the first sentence entails, contradicts or is neutral with respect to the second sentence. Then, given a concept of their choosing, e.g. the amount of overlapping words between the two sentences, they can compute a metric of contrastive importance of this concept between two classes. In the case of word overlap, they validate that in general this concept allows the model to differentiate between entailment and neutrality.

Bai et al. propose "concept gradients" (CGs) [2], a generalization of a pre-existing technique called "concept activation vectors" (CAVs) [32], to evaluate how much a concept influences the model behavior, which gives an indication of whether the

model makes use of this concept in its prediction. More precisely, concepts are evaluated using the correlation of gradients between a corresponding concept vector and the model outputs; however, CAVs assume that the concept can be expressed as a linear function, whereas CGs have no such requirement and are therefore capable of evaluating more flexible concepts.

3.1.2 Causal learning and counterfactuals

Counterfactual statements are closely related to causal inference [47]. This link is formally made in Pearl's theory of *structural causal models* (or SCMs for short): given a complete formal description of the causal relationships between variables, the theory describes a natural procedure to compute how a change in one variable would influence the others [51].

Karimi et al. develop this relationship in the context of CFX in the form of a lower bound statement: given some cost function, if the true SCM is known fully, then the Pearl procedure yields a CF that has minimum cost [31]. In other words, any other proposed CF will in some way neglect some variable relationships, and hence include redundant perturbations which increase the cost.

This brings us to the topic of causal discovery: the problem of extracting a causal graph from (usually observational) data. An introduction to classical methods can be found in [13], and a review of the literature on ML-powered techniques up to 2022 is presented in [64]. As mentioned, the full causal graph for a given task is a powerful tool which enables all kinds of inference to be carried out. However, the data used for training machine learning models seldom includes causal information about how the data were obtained, because real-world data collection processes often do not allow for causal experiments to take place, be it for physical (e.g. in economy) or ethical reasons (e.g. in medicine). Hence, neural network models (and statistical models more generally) can only uncover *correlations* between variables: the true form of the causal graph can only be known up to undirected edges between variables and possibly some unobserved latent variables.

3.1.3 Semantic latent features

When building a latent variable generative model, the idea of being able to tune human-understandable parameters is appealing: for example, when generating pictures of human faces, one might want to generate a smiling face in particular, or a face of a person wearing glasses. One way to achieve this is to encode into the latent space the concepts of "smiling" and "glasses", so that a particular direction in latent space represents a given concept. Then, given an image latent z, we can increase its "smiling factor" by perturbing z in the direction representing the concept of smiling.

Thus, some research has been done to promote this encoding of semantic information, and also to analyze whether some popular generative models have this semanticity property even when not explicitly trained for it.

For example, recent work demonstrates this exact kind of manipulation on a diffusion model [38]. In this work, the authors detail what they mean by semanticity:

Concepts are universal If you perturb two inputs in the direction of the same concept c, then in both cases the result should exhibit c.

Concepts scale If you perturb in the direction of c by 2 units, the result will exhibit the concept more than if you perturb by 1 unit.

Concepts add If you perturb in a direction that is a combination of those representing concepts c_1 and c_2, then the result will be modified in these two ways at the same time.

Non-destructiveness Perturbing in a semantic direction does not result in an unrealistic ("broken") point.

This description is reminiscent of adversarial perturbations, and in particular the universality of semanticity is challenged in [46]. In this work, the authors demonstrate that, given a conventional deep learning model, it is possible to find *one* small constant perturbation which, when applied to *any* input in the dataset, leads to a change in model prediction.

Yet, for many applications it is often sufficient to require weaker properties of latent features, such as disentanglement, efficiency or non-spuriousness. Wang and Jordan bridge the gap between latent representations and causal learning by formulating these properties in terms of causal quantities; because these quantities are difficult to compute in general, they derive a metric approximating the degree of efficiency and non-spuriousness, which can be used to regularize representation learning models [66]. This can only be used in the supervised learning setting, where a label is known and can be used for training. In fact, Locatello et al. demonstrate that achieving a disentangled representation is not possible without labels [45].

3.2 CFX generation methods

We now present some previous works on CF generation methods, covering in particular those that rely on generative models (all except `DiCE`).

3.2.1 Methods without generative models

CF-Rule duality Geng, Schleich, and Suciu prove a duality theorem linking rule-based and CF explanations, assuming that certain properties hold on the CFX [23]. They exploit this duality in one direction by devising two methods for extracting rule-based explanations from CFX. In doing so, they demonstrate that a great many

methods in the literature do not fit the requirements for producing proper rules; namely, most of them do not have the ability to specify logical feasibility constraints on the CFX.

`DiCE` Mothilal, Sharma, and Tan generate a diverse set of CFX by gradient descent, solving for k candidate CFX at the same time [48]. Diversity is modeled by a metric taken from the study of determinantal point processes, which is based on the pairwise distances between the k CFX. Using this constraint, they succeed at keeping a high rate of valid CFX while increasing diversity.

Wachter *et al.* We described the gradient-descent procedure presented by Wachter, Mittelstadt, and Russell [65] in subsection 1.2.1; the other gradient-descent-based methods we will now review are inspired by this work.

3.2.2 Methods with generative models

`Revise` Joshi et al. follow the strategy outlined by Wachter, Mittelstadt, and Russell in that they frame the CFX search as a constrained optimization problem and approach it with gradient descent: If x^* is an example with true label $y^* = 0$ then they find a CFX x' by solving:

$$x' = \arg\min_x c(x, x^*)$$
$$\text{such that } f(x') = y' = 1$$

where c is some cost function that is situation-dependent. However, in order to ensure the realism of the CFX, they perform the optimization in a latent space learned through a VAE.

For example, they use as a cost function the L_1 distance between the original input x^* and the candidate counterfactual x'. The VAE encoder is written $\mathcal{F}$ and the decoder is called $\mathcal{G}$. Thus, letting $z' = \mathcal{F}(x')$, and ℓ a loss function like the one used for training f (such as the cross-entropy), the loss that is minimized is the following:

$$\ell(f(\mathcal{G}(z')), 1) + \lambda_{\text{distance}} \|\mathcal{G}(z') - x\|_1 \tag{3.1}$$

The resulting CFX generation algorithm is called `Revise` [29].

`CFProto` Van Looveren and Klaise also use a VAE to ensure realism, but they follow a different strategy [62]. For a given x, they build a list of "prototypes", one per class, which are interpreted as typical examples of those classes. The prototype of class c for x, proto_c, is the centroid of the k-neighborhood of $\text{Enc}(x)$: the average latent over the k points predicted as c closest to $\text{Enc}(x)$. Then, the optimization algorithm for finding x^{CF} pushes the path towards the closest prototype. The reasoning is that proto_c is likely realistic, because it is an average of real points, and it is likely of class

c, for the same reason; therefore, the closer x^{CF} is to a prototype, the more realistic and valid it is likely to be.

Diffeomorphic Counterfactuals `Revise`, which we discussed in section 3.2.2, relies on a VAE to promote realistic CFX. However, this model is in general not invertible, so that there is a loss of information when sending inputs through it. This negatively affects the ,fits this assumption very welnegatively affects the trustworthiness of the explanations: if the autoencoder is not trained well enough, the latent space will not accurately model the data distribution and the resulting CFX will be of little use.

For this reason, in [17] Dombrowski, Gerken, and Kessel use normalizing flows instead, which are invertible by design. Furthermore, they draw on differential geometry to prove that if the NF is well-trained, then the CFX will stay within the data manifold.

They show that for multi-class problems on images, their method achieves sensible counterfactuals compared to simply perturbing the inputs in input space.

`CLUE` Antorán et al. introduce `CLUE` [1], a method for producing counterfactuals that are as realistic as possible by way of a conditional VAE. The method is very similar to `Revise`: the loss function that is minimized to perturb x via gradient descent is

$$\mathscr{L}(z') = h(y|\,\mathrm{Dec}(z')) + \|\mathrm{Dec}(z') - x\|_1 \tag{3.2}$$

where h is some measure of "uncertainty" (what we call realism) of $\mathrm{Dec}(z')$ with respect to a target label y. Their approach to evaluation of CFX is intriguing: to get a known ground-truth data distribution without resorting to purely fabricated datasets, they train a so-called "ground-truth" VAE on a real dataset (the MNIST hand-written digits dataset); this VAE is then used to generate a new artificial dataset, and the method is tested on these *artificial* data where metrics such as the "ground-truth" likelihood can be measured.

Importantly, the authors validate their findings with a user study, which is not frequent in works on CFX.

3.2.3 Methods with class-aware generative models

`CRUDS` Downs et al. make use of a conditional subspace VAE (or CSVAE for short) to adapt the latent representation to the classification task: the latent space is decomposed (or "disentangled") into a product of small spaces: $\mathcal{H} = \mathcal{Z} \times \prod_i \mathcal{W}_i$, where each space $\mathcal{W}_i$ contains information on the features most correlated with label i [36]. For an input x, they let $\mathrm{Enc}(x) = (z, w)$ where z is not sensitive to the label and w is sensitive to the label; thus, by altering only w based on the target class, they can then form a counterfactual latent as $z^{\mathrm{CF}} = (z, w^{\mathrm{CF}})$ and let $x^{\mathrm{CF}} = \mathrm{Dec}(z^{\mathrm{CF}})$. In fact, by

sampling several w^{CF} they can obtain a whole set of CFX that are uniquely different from x.

As a side note, recall the recent proposal for the Union of manifolds hypothesis [8] we mentioned in subsection 1.2.3: this method fits this assumption very well.

VAE-NF blend Zhang, Barr, and Paisley train the autoencoder conjointly with the classifier, so that the latent representation used for CFX generation is also used for classification [68]. In particular, the latent representation architecture consists of normalizing flows within a VAE, an idea first developed in [53].

3.2.4 Summary of methods

We summarize the presented works in terms of which generative model is used (if any) and whether or not the generative model is trained using the class labels in some way. We also mention when a work does not specifically target or test their method on tabular data problems.

	Tabular data	Generative model	Class-aware
`CFProto` [62]	Yes	VAE	No
`CLUE` [1]	Yes	VAE	No
`CRUDS` [19]	Yes	CSVAE	Yes
Diffeo. CF [17]	No	RealNVP	No
`Revise` [29]	Yes	VAE	No
VAE-NF [68]	Yes	VAE+NF [53]	Yes
`DiCE` [48]	Yes	None	NA
Rule-CF duality [23]	Yes	None	NA
Wachter *et al.* [65]	Yes	None	NA

TABLE 3.1: Summary of CF generation methods we presented, in alphabetical order. "Tabular data" refers to whether or not the work specifically addresses tabular data; "Class-aware" refers to whether or not the generative model is trained with access to class labels or similar information.

3.3 Metrics for CFX

In general, measuring the quality of an explanation is not a straightforward endeavor. However, in our case we have at least one precise goal: to change the class to a target class. When this is achieved, we say that the explanation is *valid*. Hence, our first and foremost metric is the *validity rate*: the fraction of CFX that reached the target.

When a method produces several CFX, it is often desirable to guarantee *diversity*, so that the proposed CFX are not all the same.

3.3.1 Distance

For a CFX to be relevant to the explained input, it should be close enough to it; thus some notion of *distance* is often used, e.g. the Euclidean (L_2) norm. Furthermore, it is often preferred to have *sparse* perturbations, in the interest of simplicity; for this the L_1 loss is a common choice [29, 1]. Van Looveren and Klaise even propose a linear combination of the two [62].

3.3.2 Feasibility

When a CFX is intended to provide recourse, authors tend to mention *feasibility* as a requirement. This refers to the possibility for, e.g. a user, to actually follow the recourse path proposed in the form of a CFX.

For example, Pawelczyk, Broelemann, and Kasneci introduce metrics based on *percentile shift*, that is, for feature i, how far x_i^{CF} has shifted in terms of percentile compared to x_i [50]. For example, letting $\mathrm{percentile}_i^{\mathscr{D}}(x) \in [0, 1]$ denote the percentile of x for feature $i \in \{1, \dots, D\}$, they define

$$\textit{total percentile shift as } \sum_{i=1}^{D} \left|\mathrm{percentile}_i^{\mathscr{D}}(x^{\mathrm{CF}}) - \mathrm{percentile}_i^{\mathscr{D}}(x)\right|$$
$$\textit{maximum percentile shift as } \max_{i \in \{1,\dots,D\}} \left|\mathrm{percentile}_i^{\mathscr{D}}(x^{\mathrm{CF}}) - \mathrm{percentile}_i^{\mathscr{D}}(x)\right|$$

Before them, Wachter, Mittelstadt, and Russell already considered a metric of distributional shift, which is based on the median absolute deviation for feature k [65]:

$$d(x, x') = \sum_{k \in F} \frac{\left|x_k - x'_k\right|}{\mathrm{MAD}_k} \quad \text{where} \quad \mathrm{MAD}_k = \mathrm{median}_{j \in P}\left\{\left|x_{j,k} - \mathrm{median}_{l \in P}\{x_{l,k}\}\right|\right\}$$

where P is the set of data points, F the set of features and $x_{j,k}$ is feature k of point x_j.

3.3.3 Realism

Measuring realism of a data point is a topic of research in itself and a number of metrics have been used in the CFX literature. Some make use of mathematically justified estimators, such as the *maximum mean discrepancy* (or MMD for short) [68], which can be cumbersome to compute, while others take advantage of generative models, e.g. using the reconstruction error of an autoencoder [62]. Other metrics such as *proximity* or *connectedness* [40] are derived from the distance between points and draw from older metrics such as the *local outlier factor* [6].

Now that we have covered some of the recent advances on CFX methods powered by generative models, let us present our original work.

Chapter 4

Methods

We now present our work in more detail.

We introduce our method for CFX generation, which is an alteration of a simple yet effective recent method from the literature. In particular we study how to adapt this method to the multi-class setting, and then we relax its requirements in an effort to achieve more interpretable CFX without compromising validity.

4.1 CFX generation

4.1.1 Problem statement

One possible formulation of the CFX generation problem is as follows:

> Let $f : \mathbb{R}^D \to [0, 1]$ be a binary classifier such that $f(x)$ outputs the predicted probability of x being in class 1. Let x be a point that is predicted to be of class 0, i.e. such that $f(x) < \frac{1}{2}$. Find a point x^{CF} for which $f(x^{\mathrm{CF}}) > \frac{1}{2}$ such that x^{CF} is comparable with x by a human.

One solution to this is Latent Shift.

4.1.2 Proposed solutions

Latent Shift was presented by Cohen et al. in 2022 as a CFX generation method for binary classification models on image data, specifically X-ray images of human lungs [10]. Given an input image x predicted as class 0 (i.e. for which $f(x)$ is low), Latent Shift produces a new image close to x that is predicted as class 1 (i.e. for which $f(x)$ is high.).

Let us describe the method. Let there be an autoencoder with E the encoder and D the decoder, and $z = E(x)$. Then, given some coefficient λ_{LS}, a candidate counterfactual x' is computed as:

$$x' = D(\, z + \lambda_{\mathrm{LS}} \nabla_z f(D(z))\,)$$

where $\nabla_z f(D(z))$ denotes the gradient of $f(D(z))$ at $z \in \mathcal{H}$.

In other words, x is perturbed in the direction of the gradient of the classifier f (the direction of steepest ascent), in order for f to increase. However, instead of computing the gradient in input space, they compute the gradient of the classifier *in latent space* to promote realistic images. Indeed, as discussed in subsection 1.2.3, for many image problems the data can be assumed to lie on a low-dimensional subspace of $\mathcal{X}$; therefore, if a generative model can recover this subspace, it can encode the input data in fewer dimensions with little information loss.

A summary diagram of the Latent Shift procedure is shown in Figure 4.1. On the left, the data manifold in input space is non-convex, so by perturbing along the gradient in the input space we risk exiting the data manifold. Hence, we first run the encoder, which we denote by Enc, then perturb the latent following the gradient of the classifier by some factor λ_{LS}, and finally run the decoder written Dec to get the candidate counterfactual x^{CF}.

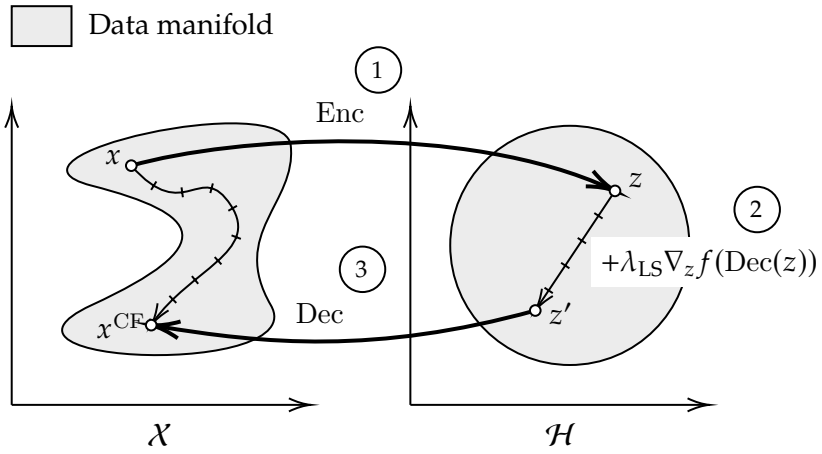


FIGURE 4.1: A sketch of the Latent Shift algorithm. Step 1 is applying Enc, step 2 is perturbing along the gradient of the classifier, and step 3 is applying Dec. Adapted from [10].

We outline some of the benefits of Latent Shift here:

1. First, this method is generally simple and easy to implement compared to some of those discussed in chapter 3. It is also fast to perform once the generative model is trained, as it requires only one gradient computation (compared to an undetermined number in classical gradient-descent based solutions).

2. Yet, one very interesting aspect of this method is that we can easily visualize the equivalent continuous path taken in input space (outlined in the left part of Figure 4.1) by running the procedure with different values of λ_{LS}; thus we can see the final result x^{CF}, but also the full progression leading to it. This is relevant for the problem of recourse mentioned in section 1.1.2: if a CFX is meant as advice for a data subject to alter their behavior to achieve a better outcome, then this advice would be more meaningful if the CFX would detail what *individual steps* are needed to achieve that change. This is outlined in research

challenge no. 2 of [63].

3. Finally, like in usual gradient descent, by setting λ_{LS} to a negative value we can also produce new points that are *more strongly* predicted as class 0.

However, in the original Latent Shift work, the method is applied to images, and the generative model used is an autoencoder architecture based on residual networks [25]. This is an acceptable choice of model for image data, but it has two important consequences:

1. The reconstruction error of the autoencoder is non-zero: encoding an image to get the starting point of the path is not an invertible operation. This implies the CF path in input space generally does not go through the original point, even for $\lambda_{\text{LS}} = 0$; in fact, if the reconstruction error is too large this problem will impact the trustworthiness of the explanation.

2. Basic autoencoders have no requirement to map inputs *continuously*, so the generated path can look very complex which also makes the explanation less trustworthy.

To overcome these issues, we turn to normalizing flows: generative models that are invertible by design. Our analyses are generally not specific to normalizing flows though, so in the following we refer to our generative model as an autoencoder or a normalizing flow interchangeably.

We choose the NICE architecture from [14] presented in subsection 2.3.2. In particular, we assume a standard Gaussian prior distribution, i.e. we enforce that the distribution in the latent space is standard Gaussian. As shown in subsection 2.3.2, if H is the dimensionality of $\mathcal{H}$ and $z = \text{Enc}(x)$ then for a Gaussian prior the NLL is written

$$\text{NLL}(x) = \frac{H}{2} \log 2\pi + \frac{z^T z}{2} - \sum_{i=1}^{H} \log |S_{ii}|$$

which is the loss that is minimized.

Importantly, there is nothing in our approach that cannot be done with the usual gradient descent procedure—which has enjoyed many improvements to promote fast convergence and avoiding local minima [33]. The Latent Shift method is nothing but a simplification of `Revise`, the procedure outlined in [29], where the gradient is computed only once and the constraint on the L_1 distance is dropped. However, they differ in goals in that the authors of [29] do not keep a record of the individual steps taken during the gradient descent run. Therefore, in the following we test our methods both on Latent Shift and `Revise`, with and without the distance constraint to stay close to the original `Revise` procedure. Our objective is to study to what extent Latent Shift can be used as a perhaps less accurate, yet much faster CFX method.

4.2 Multi-class extension

4.2.1 Problem statement

Recall that the authors of Latent Shift propose to recover a potential counterfactual by perturbing the input using the gradient of the classifier in latent space:

$$x^{\mathrm{CF}} = \mathrm{Dec}(z + \lambda_{\mathrm{LS}} \nabla_z f(\mathrm{Dec}(z))).$$

When $f : \mathcal{X} \to [0, 1]$ is a binary classifier outputting the **predicted probability** that x is of class 1, this is clearly sufficient. However, in multi-class problems where f outputs a vector of **logits** in $\mathbb{R}^C$ there are several other functions we can use. To see this, we reformulate the latent shift strategy with a generic loss that accounts for validity, denoted by $\mathscr{L}_{\mathrm{validity}}$:

Let f be the classifier of interest, which outputs logits in $\mathbb{R}^C$ rather than a probability mass. Given $x \in \mathcal{X}$ and an autoencoder $\mathrm{AE} = (\mathrm{Enc}, \mathrm{Dec})$, a counterfactual path can be generated by computing the gradient of a so-called "validity loss" $\mathscr{L}_{\mathrm{validity}}$ and perturbing $z = \mathrm{Enc}(x)$ in the direction of steepest *descent* of $\mathscr{L}_{\mathrm{validity}}$:

$$x^{\mathrm{CF}} = \mathrm{Dec}(z - \lambda \nabla_z \mathscr{L}_{\mathrm{validity}}(z)).$$

In basic Latent Shift, the validity loss would then be $\mathscr{L}_{\mathrm{validity}}(z) = -\sigma(f(\mathrm{Dec}(z)))$ where σ denotes the sigmoid function, for example (recall that in the original paper the classifier outputs a probability in $[0, 1]$).

Hence, our problem is as follows:

> Find a validity loss $\mathscr{L}_{\mathrm{validity}}$ that, for a given $x \in \mathcal{X}$, gives rise to a class-changing path in input space when used in a gradient-based path generation method such as the one described above.

4.2.2 Proposed solutions

Note that this extension to the multi-class setting means that we have a choice to specify a given target class: we could set a requirement to simply change the class, without preference of target class. In our analysis we focus on the case where a target class is specified.

`PrbTarget` In order to mimic the solution discussed in [10], we have

$$\mathscr{L}_{\mathrm{validity}}(z) = -\mathrm{softmax}(\, f(\mathrm{Dec}(z))\,)_{\mathrm{target}}, \tag{4.1}$$

i.e. the component of the output probability mass corresponding to class target. We call this loss `PrbTarget`. As before, the codomain is $[0, 1]$, and the loss corresponds

directly to what we wish to achieve. However, the multi-class setting assumption gives rise to other candidate losses.

Indeed, in the binary classification case, reaching towards the target class is exactly the same as moving away from the source class, since $p_{\text{source}} = 1 - p_{\text{target}}$. By contrast, in the multi-class setting, different priorities need to be distinguished on a case-by-case basis: is it more important to reach the target (at the risk of potentially staying in the source class) or to get away from the source (at the risk of not reaching the target)? The natural extension we presented is favoring the first option, but in fact there is no reason to neglect the second.

`PrbSource` To address this, we introduce the following alternative loss, with regularization $\lambda \in [0, 1]$:

$$\mathscr{L}_{\text{validity}}(z) = -\text{softmax}(\, f(\text{Dec}(z))\,)_{\text{target}} + \lambda \text{softmax}(\, f(\text{Dec}(z))\,)_{\text{source}} \tag{4.2}$$

which can be interpreted as adding the constraint of going towards the target as well as *going away from the source*. We call it $\texttt{PrbSource}_\lambda$.

In fact, we can go further: in Latent Shift, the objective of increasing p_{target} is also equivalent to decreasing $\sum_{\substack{c=1 \\ c \neq \text{target}}}^{C} p_c$, since in binary classification this is just p_{source}; however, in multi-class problems they are distinct quantities. Therefore we also define:

$$\texttt{PrbOthers}_\lambda(z) = -\text{softmax}(\, f(\text{Dec}(z))\,)_{\text{target}} + \lambda \sum_{\substack{c=1 \\ c \neq \text{target}}}^{C} \text{softmax}(\, f(\text{Dec}(z))\,)_c$$

But notice

$$\begin{aligned}
\texttt{PrbOthers}_\lambda(z) &= -\text{softmax}(\, f(\text{Dec}(z))\,)_{\text{target}} + \lambda(1 - \text{softmax}(\, f(\text{Dec}(z))\,)_{\text{target}}) \\
&= -(\lambda + 1)\text{softmax}(\, f(\text{Dec}(z))\,)_{\text{target}} + \lambda \\
&= (\lambda + 1)\texttt{PrbTarget}(z) + \lambda
\end{aligned}$$

which is a rescaling of `PrbTarget`, so in practice `PrbOthers` is not used.

To examine our intuition for why these proposals are sensible, we can compute the gradient of the loss, for example

$$\nabla_z \texttt{PrbSource}_\lambda = \nabla_z \left(-\text{softmax}(\, f(\text{Dec}(z))\,)_{\text{target}} + \lambda \text{softmax}(\, f(\text{Dec}(z))\,)_{\text{source}}\right)$$

Theoretical analysis of gradients

In practice, differentiating with respect to z depends on the architectures of the classifier and of the autoencoder, so we cannot draw information from it in the general case. The next best thing we can do is to notice that in our loss expressions, z only

ever appears within $f(\mathrm{Dec}(z))$, so we let $u(z) = f(\mathrm{Dec}(z))$ denote the output logits. By the chain rule we then have:

$$\nabla_z \mathscr{L}_{\text{validity}}(z) = \nabla_{u(z)} \mathscr{L}_{\text{validity}}(u(z)) \times J_{u(z)}(z)\,.$$

Furthermore, to lighten the calculations we let $p_i(u) = \mathrm{softmax}(u)_i$. In the following we usually just write u to mean $u(z)$ and p_i to mean $p_i(u)$ for brevity.

The gradient with respect to z is then written:

$$\nabla_z \mathscr{L}_{\text{validity}} = \nabla_u \mathscr{L}_{\text{validity}}(u) \times J_u(z)\,.$$

As mentioned, we cannot say much about $J_u(z)$, but importantly, it does not depend on the loss, so we can at least focus on $\nabla_u \mathscr{L}_{\text{validity}}$.

For `PrbSource` the gradient with respect to u is written

$$\nabla_u \texttt{PrbSource}_\lambda = \nabla_u \left(-p_{\text{target}} + \lambda p_{\text{source}}\right)$$

according to our short-hand notation.

We show in section A that for $i \in \{1, \ldots, C\}$ and $j \in \{1, \ldots, C\}$:

$$\frac{\partial p_i}{\partial u_j} = p_i(\delta_{ij} - p_j)$$

where δ_{ij} is the Kronecker delta.

Thus, for $j \in \{1, \ldots, C\}$ the gradient of `PrbSource` is written:

$$\begin{aligned}
\frac{\partial \texttt{PrbSource}_\lambda}{\partial u_j} &= -\frac{\partial p_{\text{target}} + \lambda p_{\text{source}}}{\partial u_j} \\
&= -\frac{\partial p_{\text{target}}}{\partial u_j} + \lambda \frac{\partial p_{\text{source}}}{\partial u_j} \\
&= -p_{\text{target}}(\delta_{j,\text{target}} - p_j) + \lambda p_{\text{source}}(\delta_{j,\text{source}} - p_j).
\end{aligned}$$

What should we conclude from this? Intuitively, a good validity loss should fulfill some basic requirements:

- When p_{target} is 1, we expect the gradient $\nabla_z \mathscr{L}_{\text{validity}}$ to be close to 0; indeed, if the target is reached, there is no reason to keep perturbing z. Therefore, we first check whether $\nabla_u \mathscr{L}_{\text{validity}} = 0$; if so, by the chain rule, we then have $\nabla_z \mathscr{L}_{\text{validity}} = 0$. If $\nabla_u \mathscr{L}_{\text{validity}} \neq 0$ then we cannot conclude in general, but we can at least

check whether

$$\frac{\partial \mathscr{L}_{\text{validity}}}{\partial u_{\text{target}}} < 0$$
$$\text{and } \forall j \neq \text{target} \quad \frac{\partial \mathscr{L}_{\text{validity}}}{\partial u_j} > 0.$$

If so, this indicates the path is pushed towards the target class and it is pulled away from the other classes.

- When p_{target} is 0, we can also check the previously stated condition on $\nabla_u \mathscr{L}_{\text{validity}}$.

Let us now examine whether these requirements are satisfied. Recall

$$\frac{\partial \texttt{PrbSource}_\lambda}{\partial u_j} = -p_{\text{target}}(\delta_{j,\text{target}} - p_j) + \lambda p_{\text{source}}(\delta_{j,\text{source}} - p_j).$$

- When p_{target} is 1, this means $p_j = \delta_{j,\text{target}}$, so we have for all j:

$$\begin{aligned}\frac{\partial \texttt{PrbSource}_\lambda}{\partial u_j} &= -1 \times (\delta_{j,\text{target}} - \delta_{j,\text{target}}) + \lambda \times 0 \times (\delta_{j,\text{source}} - p_j) \\ &= 0, \text{ so } \nabla_z \texttt{PrbSource}_\lambda = 0 \text{ as required.}\end{aligned}$$

- When p_{target} is 0, we get:

$$\begin{aligned}\frac{\partial \texttt{PrbSource}_\lambda}{\partial u_j} &= -0 + \lambda p_{\text{source}}(\delta_{j,\text{source}} - p_j) \\ &= \begin{cases} 0 & \text{if } j = \text{target} \\ \lambda p_{\text{source}}(1 - p_{\text{source}}) & \text{if } j = \text{source} \\ -\lambda p_{\text{source}} p_j & \text{otherwise.} \end{cases}\end{aligned}$$

The components for the classes that are not source or target are negative, which means these classes are favored compared to target; this goes against our intuition.

Compare this with `PrbTarget`: since $\texttt{PrbTarget} = \texttt{PrbSource}_{\lambda=0}$ we see that in both cases $\nabla_u \texttt{PrbTarget} = 0$. This means that if the path gets stuck at $p_{\text{target}} = 0$ then it cannot recover.

Log-probability-based losses

When training classifiers, using the probabilities directly in the loss is unusual; a more common choice for comparing predicted probabilities with true labels is the cross-entropy CE [49]. The cross-entropy involves the logarithm of the predicted

probabilities, so we can easily modify our previous attempts to get:

$$\begin{aligned}
\texttt{LogPrbTarget}: &\quad \mathscr{L}_{\text{validity}}(u) = -\log p_{\text{target}} \\
\texttt{LogPrbSource}_\lambda: &\quad \mathscr{L}_{\text{validity}}(u) = -\log p_{\text{target}} + \lambda \log p_{\text{source}} \\
\texttt{LogPrbOthers}_\lambda: &\quad \mathscr{L}_{\text{validity}}(u) = -\log p_{\text{target}} + \lambda \sum_{\substack{c=1 \\ c \neq \text{target}}}^{C} \log p_c
\end{aligned}$$

where $\lambda \in [0, 1]$ is a regularization parameter.

Let us compute the gradients of our log-probability losses to get a better understanding of their behavior.

`LogPrbSource` and `LogPrbTarget` Recall the expression for `LogPrbSource` is

$$\texttt{LogPrbSource}_\lambda(u) = -\log p_{\text{target}} + \lambda \log p_{\text{source}}.$$

Notice that $\texttt{LogPrbSource}_{\lambda=0}(u) = \texttt{LogPrbTarget}(u)$, so we only have to compute the gradient of $\texttt{LogPrbSource}_\lambda$.

We have

$$\frac{\partial \log p_i}{\partial u_j} = \frac{\partial p_i}{\partial u_j} \times \frac{1}{p_i} = \frac{p_i(\delta_{ij} - p_j)}{p_i} = \delta_{ij} - p_j$$

Hence

$$\begin{aligned}
\frac{\partial \texttt{LogPrbSource}_\lambda}{\partial u_j} &= -\frac{\partial \log p_{\text{target}}}{\partial u_j} + \lambda \frac{\partial \log p_{\text{source}}}{\partial u_j} \\
&= -(\delta_{j,\text{target}} - p_j) + \lambda(\delta_{j,\text{source}} - p_j) \\
&= -\delta_{j,\text{target}} + p_j + \lambda \delta_{j,\text{source}} - \lambda p_j \\
&= (1 - \lambda) p_j + \lambda \delta_{j,\text{source}} - \delta_{j,\text{target}}.
\end{aligned}$$

We check our intuition:

- For $p_{\text{target}} = 1$ this gives us

$$\begin{aligned}
\frac{\partial \texttt{LogPrbSource}_\lambda}{\partial u_j} &= (1 - \lambda)\delta_{j,\text{target}} + \lambda \delta_{j,\text{source}} - \delta_{j,\text{target}} \\
&= \lambda(\delta_{j,\text{source}} - \delta_{j,\text{target}}) \\
&= \begin{cases} -\lambda & \text{if } j = \text{target} \\ \lambda & \text{if } j = \text{source} \\ 0 & \text{otherwise,} \end{cases}
\end{aligned}$$

so the gradient keeps pushing u_{target} higher and u_{source} lower, as required. When $\lambda = 0$, which corresponds to `LogPrbTarget`, the gradient is zero which is also acceptable intuitively.

- For $p_{\text{target}} = 0$ we get

$$\begin{aligned}\frac{\partial \texttt{LogPrbSource}_\lambda}{\partial u_j} &= (1-\lambda)p_j + \lambda\delta_{j,\text{source}} - \delta_{j,\text{target}} \\ &= \begin{cases} -1 & \text{if } j = \text{target} \\ (1-\lambda)p_{\text{source}} + \lambda & \text{if } j = \text{source} \\ (1-\lambda)p_j & \text{otherwise.} \end{cases}\end{aligned}$$

  When $j \neq \text{target}$ the gradient is larger than 0 so again, class target dominates.

`LogPrbOthers` Recall the expression for `LogPrbOthers`:

$$\texttt{LogPrbOthers}_\lambda(u) = -\log p_{\text{target}} + \lambda \sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} \log p_c.$$

Thus its gradient is

$$\frac{\partial \texttt{LogPrbOthers}_\lambda}{\partial u_j} = p_j(1-\lambda(C-1)) + \lambda(1-\delta_{j,\text{target}}) - \delta_{j,\text{target}}.$$

The full derivation can be found in section A.

We check our intuition:

- For $p_{\text{target}} = 1$ this gives us

$$\begin{aligned}\frac{\partial \texttt{LogPrbOthers}_\lambda}{\partial u_j} &= \delta_{j,\text{target}}(1-\lambda(C-1)) + \lambda(1-\delta_{j,\text{target}}) - \delta_{j,\text{target}} \\ &= \delta_{j,\text{target}} - \delta_{j,\text{target}}\lambda(C-1) + \lambda - \lambda\delta_{j,\text{target}} - \delta_{j,\text{target}} \\ &= -\delta_{j,\text{target}}\lambda(C-1) + \lambda - \lambda\delta_{j,\text{target}} \\ &= -\lambda(\delta_{j,\text{target}}(C-1) - 1 + \delta_{j,\text{target}}) \\ &= -\lambda(C \times \delta_{j,\text{target}} - 1) \\ &= \begin{cases} -(C-1)\lambda & \text{if } j = \text{target} \\ \lambda & \text{otherwise} \end{cases}\end{aligned}$$

  so the gradient pushes u_{target} higher than that for other classes, as required.

- For $p_{\text{target}} = 0$ we get

$$\frac{\partial \texttt{LogPrbOthers}_\lambda}{\partial u_j} = p_j(1 - \lambda(C-1)) + \lambda(1 - \delta_{j,\text{target}}) - \delta_{j,\text{target}}$$
$$= \begin{cases} -1 & \text{if } j = \text{target} \\ p_j(1 - \lambda(C-1)) + \lambda & \text{otherwise.} \end{cases}$$

Here it is unclear if the gradient components for $j \neq \text{target}$ are more negative than -1. It turns out it can happen: for example, if $\lambda = 1$ and $p_j = 1$ then we get $1 - (C-1) + 1 = 3 - C$, so for $C > 4$, u_j will be pushed higher than u_{target}. We show this the behavior for $\lambda = 1$ in Figure 4.2.

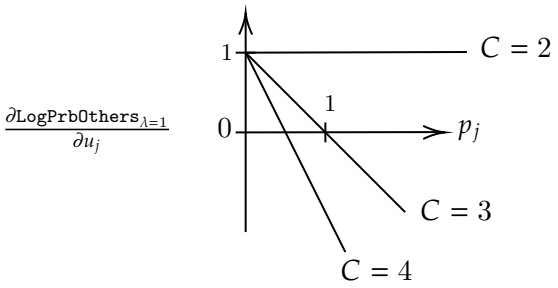


FIGURE 4.2: Behavior of the gradient of `LogPrbOthers` for $\lambda = 1$, for different values of C. As C increases, the threshold at which the gradient component changes signs decreases.

Logit-based losses

Our previous loss functions are somewhat difficult to control because of the interactions between the different classes: when we state our requirement to increase p_{target}, this can be understood either as increasing u_{target}, or as decreasing the other u_c. For this reason, we formulate losses optimizing for the logits directly (with $\lambda \in [0, 1]$):

$$\texttt{LogitTarget}: \quad \mathscr{L}_{\text{validity}}(u) = -u_{\text{target}}$$
$$\texttt{LogitSource}_\lambda: \quad \mathscr{L}_{\text{validity}}(u) = -u_{\text{target}} + \lambda u_{\text{source}}$$
$$\texttt{LogitOthers}_\lambda: \quad \mathscr{L}_{\text{validity}}(u) = -u_{\text{target}} + \lambda \sum_{\substack{c=1 \\ c \neq \text{target}}}^{C} u_c$$

Here the gradients are easy to compute since the loss is linear in the logits:

$$\begin{aligned}
\frac{\partial \texttt{LogitTarget}}{\partial u_j} &= -\delta_{j,\text{target}} \\
\frac{\partial \texttt{LogitSource}_\lambda}{\partial u_j} &= -\delta_{j,\text{target}} + \lambda \delta_{j,\text{source}} \\
\frac{\partial \texttt{LogitOthers}_\lambda}{\partial u_j} &= -\delta_{j,\text{target}} + \lambda \sum_{\substack{c=1 \\ c \neq \text{target}}}^{C} \delta_{jc} \\
&= -\delta_{j,\text{target}} + \lambda \times \begin{cases} 0 & \text{if } j = \text{target} \\ 1 & \text{otherwise} \end{cases} \\
&= -\delta_{j,\text{target}} + \lambda(1 - \delta_{j,\text{target}}) \\
&= -(\lambda + 1)\delta_{j,\text{target}} + \lambda.
\end{aligned}$$

Hence the behavior is also clear: in all cases u_{target} always gets pushed up higher than the other logits, no matter the value of u.

To conclude on this section, let us summarize our findings.

- The behavior of the probability-based losses does not fit our intuitive requirements: when $p_{\text{target}} = 0$, our initial guess `PrbTarget` completely stops the path, and our first refinement `PrbSource` pushes the other classes up.
- The log-probability-based losses overcome these issues, except `LogPrbOthers` which can also promote the other classes under certain conditions.
- The logit-based losses generally behave according to our intuition for the cases we considered.

4.3 Path regularization

4.3.1 Problem statement

In this section and in the following, we denote a CFX path by $(x_t)_{t=1}^T \in \mathcal{X}^T$, where x_T is the end point of the path and x_1 is its starting point, ideally as close to x as possible (so T is the number of points in the path). Similarly, we also define $(z_t)_{t=1}^T \in \mathcal{H}^T$ such that $(x_t)_{t=1}^T = (\text{Dec}(z_t))_{t=1}^T$.

A typical requirement on a counterfactual x^{CF} for an input x is for $\left\|x^{\text{CF}} - x\right\|$ to be small, where $\|\cdot\|$ is the L_1 norm for example. While this makes sense for methods that produce a discrete set of counterfactuals, we focus on methods that produce *continuous explanation paths*, hence we wish to ensure these requirements are satisfied *along the path*. In the example of distance, this would give us the constraint that the endpoint of the path $x^{\text{CF}} = \text{Dec}(z_T)$ be close to the starting point, but also that *on the whole* the path $(z_t)_{t=1}^T$ stay as close as possible to the starting point.

In the case of gradient-descent optimization paths, this can be addressed by placing these constraints in the loss function during the computation of the next step: this is done in `Revise`, where the loss function includes the L_1 distance [29]:

$$\mathscr{L}_{\text{CF}}(z_t) = \mathscr{L}_{\text{validity}}(z_t) + \lambda_{\text{distance}} \left\| \text{Dec}(z_t) - x \right\|_1 .$$

However, for Latent Shift this solution is not appropriate because the gradient is only computed *once*, for $z = \text{Enc}(x)$. Indeed, even though the first step goes in the right direction, overall the whole path can still go in the wrong direction because the information carried by the gradient is less relevant the further away we go from the input.

Hence, our second problem statement is the following:

> Given a path CF method involving a generative model such as a normalizing flow, how can we constrain a whole path $(x_t)_{t=1}^T$ according to some requirements?

4.3.2 Proposed solution

Since we can extend many of the point-wise measures to whole paths in a natural way, e.g. with the mean or the maximum, we propose applying this as a loss on the whole path and using it to *train the autoencoder* in addition to its usual loss function (e.g. the NLL). This added component to the loss can take various forms, and in general we call it *path regularization*. The full process is described with pseudocode in algorithm 1. Note that in NF-based autoencoders, the decoder is the inverse of the encoder and thus does not need to be trained like it would in a VAE; this is why we show explicitly only the trainable parameters of Enc (denoted by ϕ).

Algorithm 1: Learning a normalizing flow latent space by SGD with path regularization

Input: Classifier f, AE $(\text{Enc}_\phi, \text{Dec})$, (training) data, CF loss $\mathscr{L}_{\text{CF}}$, path loss $\mathscr{L}_{\text{path}}$, regularization coefficients λ_{path} and λ_{CF}

Output: ϕ^* that minimizes the NLL and the path loss and the CF loss

while *not converged* **do**

- $x \leftarrow$ `get_input`(data)
- $z \leftarrow \text{Enc}_\phi(x)$
- source $\leftarrow \arg\max_{c \in \mathcal{C}} f(x)$
- target $\leftarrow$ `random_target`(x) `// such that` target $\neq$ source
- path $\leftarrow$ `generate_path`$_{\mathscr{L}_{\text{CF}}}(z, \text{source}, \text{target})$ `//` path $= (x_t^{\text{CF}})_{t=1}^T$
- $\phi \leftarrow \phi - \nabla_\phi(\text{NLL}(z) + \lambda_{\text{path}}\mathscr{L}_{\text{path}}(\text{path}) + \lambda_{\text{CF}}\mathscr{L}_{\text{CF}}(\text{path}))$

end

Since the one obvious constraint on paths is that they should be valid, we include a term $\mathscr{L}_{\text{CF}}$. In fact we ended up not making use of this to keep the problem simple and prevent potential stability issues during training; in future work it could be used for example to impose frozen features on the paths.

$\mathscr{L}_{\text{path}}$ represents our criteria for an interpretable path. So what criteria should we enforce? We choose to apply the following constraint: the path should go from source to target passing through as few other classes as possible, and ideally, without passing through any other class. Thus we formulate `BoundaryCrossLoss` as follows:

$$\texttt{BoundaryCrossLoss}(\text{path}) = -\frac{1}{S}\sum_{t=1}^{S} \max\{p^{(t)}_{\text{source}}, p^{(t)}_{\text{target}}\}$$

where $p_i^{(t)}$ is the predicted probability $\text{softmax}(\, f(x_t)\,)_i$.

Note that class information is indirectly given during the training of the autoencoder, so the autoencoder is dependent on the classification task. By contrast, in basic Latent Shift the authors make a conscious decision to keep the autoencoder entirely separate from the classifier. Their justification is that this allows for reusing the autoencoder on classifiers that have learned different tasks but that act on the same input data; for example, different classifiers each trained to detect a particular lung disease. While a valid argument, Locatello et al. have demonstrated that some properties of latent representations can only be achieved in the presence of class label information [45].

We are now in position to describe our experiments and findings.

Chapter 5

Experiments and results

We now present our experiments to address the problems stated in chapter 4.

First we give details on our datasets and settings common to all experiments, next we present our metrics of choice and then we delve into the experiments themselves, presenting the experimental setup, the results and discussion for each of them in turn.

5.1 Datasets

In our experiments we use one custom synthetic dataset generated in a parametrizable way, as well as some real datasets.

We always take out the categorical features and we standardize each feature.

5.1.1 `CakeOnSea`

Our analyses are complicated by the impossibility of visualizing counterfactuals for tabular data in general. Thus, in order to start approaching the problem intuitively, one of our datasets is a synthetic dataset with no more than two features having an influence on the response. The two features, denoted by x_0 and x_1, range from 0 to 50, except for the so-called "dead zone" which is $[25, 35]^2$, in which there are no points at all. There are three classes, and the class decision rule is as follows:

$$\begin{aligned} (x_0, x_1) \in [35, 45]^2 &\implies y = 2 \\ x_1 < 25 &\implies y = 0 \\ (x_0, x_1) \in \text{dead zone} \vee (x_0, x_1) \notin [0, 50]^2 &\implies y \text{ unknown} \\ \text{otherwise} \qquad y &= 1 \end{aligned}$$

Note that we explicitly refrain from assigning a class to points in the dead zone or outside $[0, 50]^2$. The decision rule can be visualized in Figure 5.1.

To generate the dataset, we randomly sample points in the $[0, 50]^2$ square and then take out points from the dead zone. By default, the sampling distribution is a uniform $\mathcal{U}([0, 50]^2)$.

Our implementation also allows for producing more features that are linear combinations of the first two, to simulate data with an arbitrary number of columns while keeping the dataset structure intact. However, in the interest of keeping things simple, we use only the 2-feature version in our analyses.

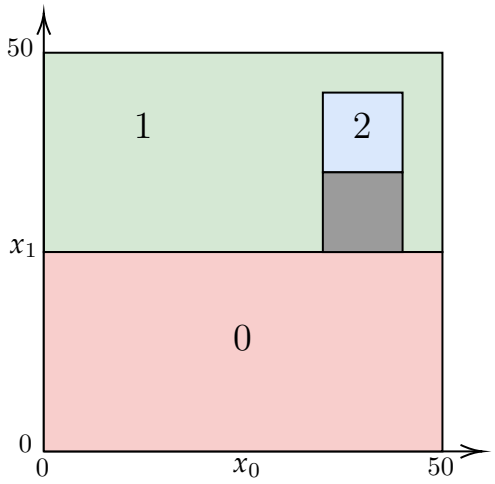


FIGURE 5.1: Decision rule for the `CakeOnSea`dataset. The so-called "dead zone" is shown in dark gray between class 2 and class 0.

5.1.2 `ForestCover`

This dataset is taken from the UCI Machine Learning repository[1] [20] based on work in [5]. It contains data on 581012 trees such as their elevation, their horizontal distance to the nearest surface water features and the type of soil they are in, as well as their species, among seven tree species. There is a total of 13 recorded variables, including the tree species. Of the 12 variables used for prediction, two are categorical, and encoded as one-hot columns. Because our method does not trivially extend to categorical features yet, we remove the one-hot columns; we thus have $D = 10$.

5.1.3 `WineQuality`

The `WineQuality` dataset originates from [12] and can be found in the UCI repository[2]. It consists of data on 6498 Portuguese "Vinho Verde" wines, such as their pH, their density and their alcohol percentage, as well as a quality rating from 0 to 10. There are 13 variables including the quality rating column; except for the quality and whether they are red or white, all of them are numerical. As before, we remove the categorical features (which is just the red-white feature).

[1] https://archive.ics.uci.edu/ml/datasets/Covertype
[2] https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Because the wine quality is an ordinal variable (categorical but ordered), it is not necessarily appropriate to run a classifier on the dataset directly; thus, we group quality ratings together into 3 classes. In fact, there are no wines with a quality rating of 0, 1, 2 or 10, and 43% of the rows have a rating of 6 out of 10. Hence, for our classification task we create the following class mapping:

- class 0 corresponds to a quality rating of 5 or lower (33% of the data),
- class 1 to a rating of 6 (43%) and
- class 2 to a rating of 7 or higher (18%).

5.1.4 `OnlineNewsPopularity`

This dataset [21] also from the UCI repository[3] contains data on 39644 news posts recovered from the Mashable website, such as the number of HTML tags of a certain type, the time of publishing, and also machine-learning-derived features such as the subjectivity and polarity of the language used, and how well the post fits within a given topic as determined by latent Dirichlet allocation. As before, we remove the categorical columns, leaving us with 46 columns including the target variable.

The target variable is the number of shares of a given article, which is an integer; in order to make it usable for classification tasks, we create the following class mapping:

- Articles with a number of shares below the median (i.e. in the worst 50%) are considered to be in class 0,
- Articles between the 50% and the 75% percentiles are considered class 1,
- Articles between the 75% and 95% percentiles are considered class 2,
- Articles above the 95% percentile (i.e. in the best 5%) are considered class 3.

Thus, class 0 contains 50% of the data, class 1 contains 25%, class 2 contains 20% and class 3 contains 5%.

5.1.5 Summary of datasets

All datasets are divided into training, validation, and test sets containing 60%, 20%, and 20% of the data respectively.

In Table 5.1 we summarize the relevant information about our datasets and settings when using them for training.

[3] https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity

Name	D	C	$\lvert\mathcal{D}_{\text{train}}\rvert$	$\lvert\mathcal{D}_{\text{validation}}\rvert$	$\lvert\mathcal{D}_{\text{test}}\rvert$	$\lvert\mathcal{D}_{\text{train}}\rvert / D$
`CakeOnSea`	2	3	57607	19202	19202	28803
`ForestCover`	10	7	348608	116202	116202	34861
`WineQuality`	11	3	3900	1299	1299	355
`OnlineNewsPopularity`	45	4	23788	7928	7928	529

TABLE 5.1: Summary of our dataset settings. D refers to the number of features, C to the number of classes, and $\lvert\mathcal{D}_{\text{train}}\rvert$, $\lvert\mathcal{D}_{\text{validation}}\rvert$, $\lvert\mathcal{D}_{\text{test}}\rvert$ to the size of the train, validation, and test set respectively.

5.2 Models

All models are trained using the Adam algorithm [33] with a learning rate of 1×10^{-3}. We let the models train for an undetermined number of epochs, performing early stopping based on the validation loss with a patience parameter of 3.

The models are implemented in `pytorch`[4] and trained on a machine with the following characteristics:

- CPU: Intel Core i7-7700K CPU @ 4.20GHz
- RAM: 32 GB
- GPU: Nvidia GeForce GTX 1080 with 8 GB VRAM via CUDA 11.7

5.2.1 Classifier

The classifier architecture in our analyses is an MLP consisting of two 50-node linear layers each followed by a ReLU, then one last linear layer outputting C logits. A diagram of the architecture is shown in Figure 5.2.

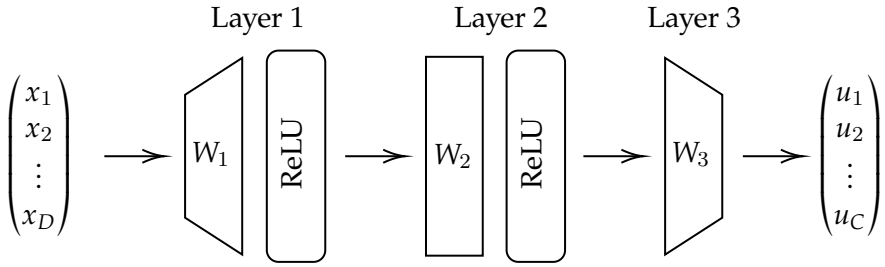
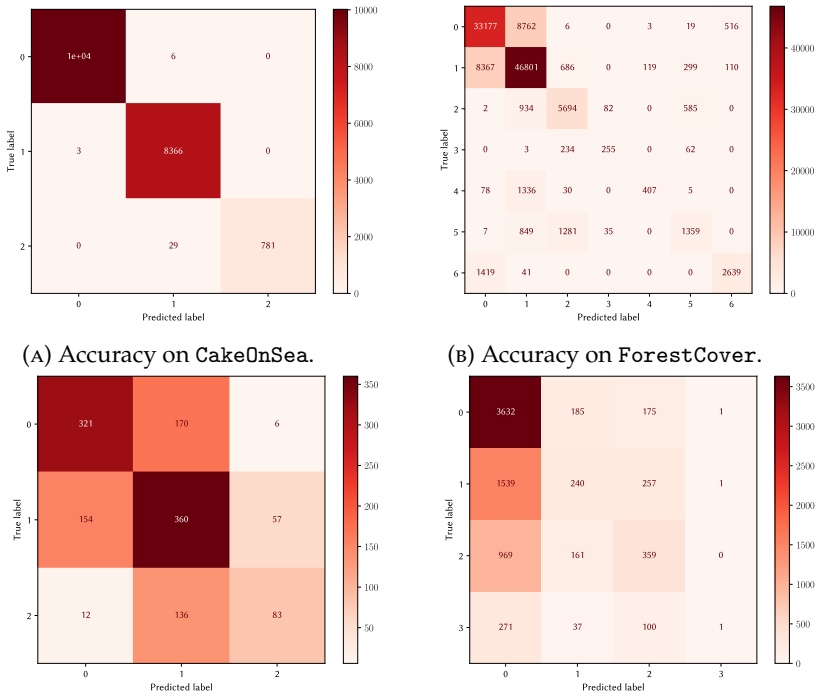


FIGURE 5.2: A diagram of our classifier architecture. The W_i are matrices containing trainable parameters; the input x is put through the model and the result is the logits u.

The batch size for each dataset is as follows:

- `CakeOnSea`: 12000
- `ForestCover`: 25000

[4] https://pytorch.org



(A) Accuracy on `CakeOnSea`.

(B) Accuracy on `ForestCover`.

(C) Accuracy on `WineQuality`.

(D) Accuracy on `OnlineNewsPopularity`.

FIGURE 5.3: Accuracy of our MLP classifier on each dataset.

- `WineQuality`: 500
- `OnlineNewsPopularity`: 3000

The batch sizes are chosen so that one epoch takes less than 10 steps, except for `ForestCover` where the size of the train set would have resulted in very large batches. In each experiment we ended up using different batch sizes so we re-state them when appropriate.

In Figure 5.3 we show the test predictions of our classifier architecture for each dataset.

It seems the classifier achieves low accuracy on `WineQuality` and `OnlineNewsPopularity` in particular. These datasets are originally designed for regression, so they are perhaps not well suited for classification tasks; however, their size is also likely a factor: they have on the order of 10^2 training examples per dimension where the others have on the order of 10^4.

All datasets also have significant class imbalance: in `ForestCover` the test set contains 56382 points with true class 1 while only 554 points in class 3.

5.2.2 Autoencoder

We build a normalizing flow model according to the NICE architecture described in subsection 2.3.2 [14]. The model is trained according to a Gaussian prior distribution, and its layers are 4 additive coupling layers where we divide the input into the odd columns and the even columns, and the nonlinearity in each coupling layer is an MLP

with 2 layers of 20 nodes each, taking a $\frac{D}{2}$-dimensional vector as input and outputting $\frac{D}{2}$-dimensional vector.

5.3 Metrics

In this section we describe the various metrics that will be used in some or all of our experiments.

5.3.1 Validity

Validity is the one requirement where the metric of interest is unambiguous: either a path is valid or it is not. For measuring validity, for each $x \in \mathcal{D}_{\text{test}}$ we select a random target class (that is not the same as the originally predicted class), produce a path, then check its validity against the chosen target class; we average this over the whole test set.

5.3.2 Computational performance

Ensuring that CFX generation is fast is an important concern for stakeholders, especially if one needs to generate paths for many points of interest. Hence, we measure the time taken *for one step of one path*. This is because depending on the number of steps chosen by the user, a path can take longer to generate, and methods such as Latent Shift can take advantage of optimizations to compute many paths at once.

5.3.3 Realism

Following the reasoning presented in [62], we measure realism using a generative model. However, while Van Looveren and Klaise use a VAE and thus estimate realism using the reconstruction error, we can compute the likelihood directly. Indeed, normalizing flows explicitely model the data distribution $P(x)$ (which is an approximation of the true distribution from which the training data originates) via the NLL, so we can compute it by design. The NLL is in $\mathbb{R}_+$ and the closer it is to 0, the more realistic the point is deemed to be.

We also used the local outlier factor [6], for which an implementation is provided by the `scikit-learn` Python package, but it did not scale to all of our datasets.

5.3.4 Extending metrics to a path

All the CFX paths are 100 points long (including the starting point, so the number of steps is 99) and the step size λ_{LS} is 0.1. We found this to be satisfactory so that the path is long enough to reach the target class, yet with step size short enough that the path does not go too far too quickly.

Many metrics take a point $x' \in \mathcal{X}$ as input and return a score, but in our case we wish to score a whole path $(x_t)_{t=1}^{T}$. Hence, given a metric, we aggregate over the path (that is, from its starting point to the first step at which target is reached).

For this, we first produce paths in the form of a 3-dimensional tensor in $\mathbb{R}^{S \times B \times D}$, where S is the number of steps in the path and B the batch size. We compute the model predictions resulting in a 2-dimensional tensor in $C^{S \times B}$. Based on this and the target classes we generated in C^B, we construct a mask $\texttt{path\_mask} \in \{0,1\}^{S \times B}$ that highlights the valid paths: if path j is not valid, then $\forall i \in \{1, \dots, S\}$ $\texttt{path\_mask}_{ij} = 0$, and if path j is valid then $\texttt{path\_mask}_{ij} = 1$ up to and *not including* the first index where target is reached, and 0 after that.

Then we can use this mask to exclude certain elements from the computation, e.g. all points from invalid paths.

An example of the procedure is shown in Figure 5.4.

$$\begin{array}{rc} \text{CFX} \\ \text{predicted} \\ \text{classes} \end{array} \begin{pmatrix} 0 & 2 & 1 & 1 \\ 0 & 2 & 0 & 1 \\ 1 & 2 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathrm{target} \quad \begin{pmatrix} 1 & 0 & 0 & 2 \end{pmatrix} \qquad \texttt{path\_mask}$$

FIGURE 5.4: An example of the path masking procedure. The predicted classes for each path step are arranged in a $S \times B$ matrix; the target classes are in a B-sized vector, with $S = 3$ and $B = 4$.

Note that if no paths are valid, the resulting product will be equal to zero, which can be misleading: $\mathrm{NLL} = 0$ is a perfect score. For this reason we take care to always report the validity rate alongside any metric computed this way.

Let us now describe our experiments in detail.

5.4 Experiment 1: Validity losses

In this experiment, we seek to determine the validity loss that maximizes validity rate among a number of choices. For this we measure the validity rate over CFs generated from the test set across several CF methods and losses.

Description For a given dataset, we train one classifier. Then for this dataset, we train several autoencoders all with the same architecture and with different seeds to average out random effects due to model initialization and dataset train-test split. The seeds are generated uniformly based on one primary seed.

For a given dataset, we measure the validity rate across the whole test set. That is, for every test point, we run the classifier to get the source class ($\mathrm{source} = \arg\max_{c \in C} f(x)$), and we sample the target class uniformly from $C \backslash \{\mathrm{source}\}$. Then we generate a path

using the given loss $\mathscr{L}_{\text{validity}}$, and count it as valid or not. It is important to sample the target uniformly to get a full understanding because, even when the label distribution is not uniform, in general there is no reason to prefer explanations for a given target rather than another.

We average the rates over the different seeds, and report the standard error.

Baselines The methods we use to produce the paths are variations on Latent Shift and `Revise`. To compare methods on a par, we run both methods twice: once with regularization on the L_1 distance, with $\lambda_{\text{distance}} = 0.3$ and once without regularization (with $\lambda_{\text{distance}}$ set to 0). In the future using some automatic tuning procedure would be best: in our case $\lambda_{\text{distance}} = 1$ led to immobile paths whereas $\lambda_{\text{distance}} = 0.3$ could be too close to performing no regularization at all.

In fact, our implementation of both Latent Shift and `Revise` is a joint abstraction that takes a parameter for the frequency of the gradient computation: for `Revise` this is set to 1 (i.e. we recompute the gradient after every step) while for Latent Shift it is null (i.e. we never recompute it after the first step). This way the number of steps and step size can be set to the same values with confidence. However, this means that we do not run `Revise` using a more sophisticated optimization algorithm like `Adam` or `RMSprop`.

The losses compared in this experiment are as follows:

- The probability-based losses: `PrbTarget`, `PrbSource`
- The log-probability-based losses: `LogPrbTarget`, `LogPrbSource`, `LogPrbOthers`
- The logit-based losses: `LogitTarget`, `LogitSource`, `LogitOthers`

where we test all losses that can be parametrized with the default parameter of $\lambda = 1$ and also with $\lambda = 0.1$, for a total of 13 losses (`PrbTarget` is not parametrized, and recall that `PrbOthers` is equivalent to `PrbTarget`.). Here again, a larger scale parameter search should have been done, but instead we first select the best-performing loss and then show the progression for $\lambda \in [0, 1]$.

Datasets We run our experiment on the datasets described in section 5.1, namely `CakeOnSea`, `ForestCover`, `WineQuality` and `OnlineNewsPopularity`.

The batch sizes are the same as those for the classifier, which are reported in subsection 5.2.1.

Metrics The one metric in this experiment is the validity rate, i.e. the proportion of paths that reach their target class across the whole test set.

Results The results for each dataset and loss are given in Table 5.2. We highlight in gray the five losses that yield the best average validity rate across all path methods;

the results are not uniform across datasets so we resort to this rough estimate to decide between the losses.

Path method	LS			$LS_{\lambda_{distance}=0}$			Revise			$Revise_{\lambda_{distance}=0}$		
							CakeOnSea					
PrbTarget	4.7	±	0.0	4.8	±	0.1	6.4	±	0.1	6.3	±	0.1
PrbSource	6.1	±	0.2	6.2	±	0.2	7.1	±	0.1	7.1	±	0.2
$PrbSource_{\lambda=0.1}$	5.0	±	0.1	5.0	±	0.2	6.3	±	0.1	6.4	±	0.1
LogPrbTarget	31.6	±	2.5	31.4	±	2.6	39.7	±	2.2	39.9	±	2.3
LogPrbSource	31.3	±	2.5	31.4	±	2.6	39.1	±	2.7	39.1	±	2.7
$LogPrbSource_{\lambda=0.1}$	31.6	±	2.5	31.6	±	2.5	40.0	±	2.2	40.1	±	2.2
LogPrbOthers	30.8	±	0.9	30.8	±	0.8	36.0	±	1.3	36.1	±	1.2
$LogPrbOthers_{\lambda=0.1}$	32.5	±	2.3	32.6	±	2.5	40.9	±	1.9	40.9	±	1.9
LogitTarget	76.5	±	1.5	76.4	±	1.5	99.8	±	0.1	99.8	±	0.1
LogitSource	62.9	±	3.2	62.6	±	3.2	100.0	±	0.0	100.0	±	0.0
$LogitSource_{\lambda=0.1}$	75.2	±	2.4	75.5	±	2.4	99.4	±	0.4	99.4	±	0.4
LogitOthers	73.1	±	3.1	73.1	±	3.0	96.6	±	3.4	96.6	±	3.4
$LogitOthers_{\lambda=0.1}$	76.3	±	1.6	76.1	±	1.6	99.8	±	0.2	99.8	±	0.2
							ForestCover					
PrbTarget	12.6	±	0.2	12.5	±	0.1	18.0	±	0.1	18.1	±	0.1
PrbSource	16.3	±	0.2	16.2	±	0.2	19.2	±	0.1	19.4	±	0.1
$PrbSource_{\lambda=0.1}$	12.7	±	0.1	12.8	±	0.1	18.2	±	0.1	18.1	±	0.1
LogPrbTarget	54.7	±	0.5	54.7	±	0.5	83.2	±	0.1	83.2	±	0.1
LogPrbSource	54.0	±	0.5	54.3	±	0.5	86.2	±	0.2	86.2	±	0.2
$LogPrbSource_{\lambda=0.1}$	54.8	±	0.4	54.8	±	0.5	83.9	±	0.1	83.8	±	0.1
LogPrbOthers	29.2	±	0.6	29.1	±	0.7	33.4	±	0.6	33.4	±	0.6
$LogPrbOthers_{\lambda=0.1}$	53.8	±	0.6	53.8	±	0.6	78.5	±	0.4	78.6	±	0.4
LogitTarget	54.0	±	0.7	54.0	±	0.7	77.5	±	1.0	77.5	±	1.0
LogitSource	54.9	±	0.7	54.8	±	0.6	88.0	±	0.2	88.1	±	0.1
$LogitSource_{\lambda=0.1}$	54.7	±	0.7	54.6	±	0.7	79.1	±	0.8	79.2	±	0.8
LogitOthers	38.0	±	0.6	38.1	±	0.5	46.7	±	0.4	46.7	±	0.4
$LogitOthers_{\lambda=0.1}$	51.8	±	0.8	51.7	±	0.7	75.0	±	1.0	75.0	±	1.0
							OnlineNewsPopularity					
PrbTarget	11.3	±	1.4	11.4	±	1.3	10.9	±	1.1	11.1	±	1.2
PrbSource	24.2	±	2.1	24.4	±	2.3	23.8	±	2.2	24.1	±	2.2
$PrbSource_{\lambda=0.1}$	12.8	±	1.3	12.9	±	1.6	12.6	±	1.3	12.5	±	1.3
LogPrbTarget	40.2	±	1.8	40.3	±	1.6	32.8	±	2.0	32.5	±	2.0
LogPrbSource	44.8	±	1.1	44.9	±	1.1	48.1	±	1.5	48.7	±	1.4
$LogPrbSource_{\lambda=0.1}$	41.5	±	1.6	41.3	±	1.5	35.8	±	2.0	35.4	±	2.0
LogPrbOthers	13.7	±	0.5	13.6	±	0.7	15.4	±	0.9	14.9	±	0.9
$LogPrbOthers_{\lambda=0.1}$	36.9	±	2.1	37.4	±	2.0	29.9	±	2.1	29.9	±	2.0
LogitTarget	24.8	±	1.8	24.7	±	1.9	23.4	±	1.7	23.5	±	1.7
LogitSource	45.1	±	0.9	45.2	±	0.8	48.7	±	1.3	48.2	±	1.2
$LogitSource_{\lambda=0.1}$	28.6	±	2.1	28.6	±	1.9	27.8	±	1.9	27.9	±	1.8
LogitOthers	26.6	±	1.2	26.8	±	1.1	24.3	±	1.2	24.8	±	1.4
$LogitOthers_{\lambda=0.1}$	24.7	±	1.8	24.4	±	1.7	23.5	±	1.6	23.4	±	1.4
							WineQuality					
PrbTarget	26.4	±	0.8	26.5	±	1.1	29.7	±	0.8	29.8	±	0.5
PrbSource	34.3	±	0.4	34.8	±	0.6	40.7	±	0.4	39.7	±	1.4
$PrbSource_{\lambda=0.1}$	28.2	±	0.7	27.3	±	0.8	31.5	±	0.7	30.7	±	0.8
LogPrbTarget	67.3	±	0.9	68.1	±	1.4	67.7	±	0.5	67.7	±	0.8
LogPrbSource	70.3	±	1.3	69.8	±	2.1	73.5	±	1.2	73.9	±	0.6
$LogPrbSource_{\lambda=0.1}$	67.1	±	1.9	68.2	±	1.2	69.6	±	1.1	69.4	±	0.7
LogPrbOthers	54.2	±	0.6	53.9	±	1.0	65.1	±	0.7	64.7	±	0.5
$LogPrbOthers_{\lambda=0.1}$	68.2	±	1.4	66.9	±	1.4	69.3	±	1.0	68.8	±	0.9
LogitTarget	52.1	±	1.3	52.6	±	1.0	57.8	±	0.9	57.7	±	1.4
LogitSource	70.2	±	1.4	69.8	±	1.1	74.5	±	1.1	73.6	±	0.7
$LogitSource_{\lambda=0.1}$	57.0	±	0.9	55.4	±	1.5	61.6	±	0.5	62.4	±	0.9
LogitOthers	63.7	±	1.2	63.7	±	1.0	71.0	±	1.3	71.5	±	1.0
$LogitOthers_{\lambda=0.1}$	54.5	±	0.9	54.7	±	1.0	60.3	±	1.0	60.3	±	0.9

TABLE 5.2: Validity rate means with their standard error. Highlighted for each dataset are the five losses with the best average results.

The validity rate achieved by `Revise` is generally much more acceptable than that of Latent Shift, across all losses. This is expected given that `Revise` can correct its trajectory as the path is generated, while Latent Shift cannot.

`LogitSource` is always among the best validity losses. However, it is not always clear which value of λ should be used. Hence, we show the validity rate for $\lambda \in \{0.01, 0.05, 0.1, 0.2, 0.5, 1\}$ in Figure 5.5.

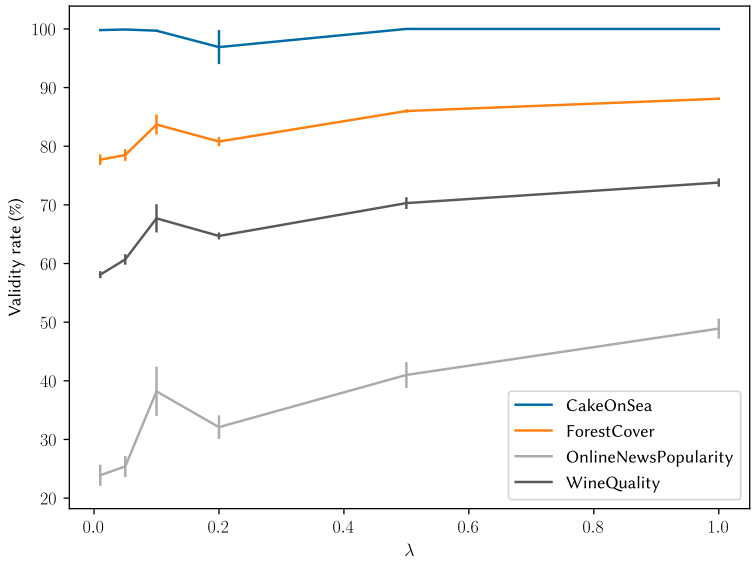


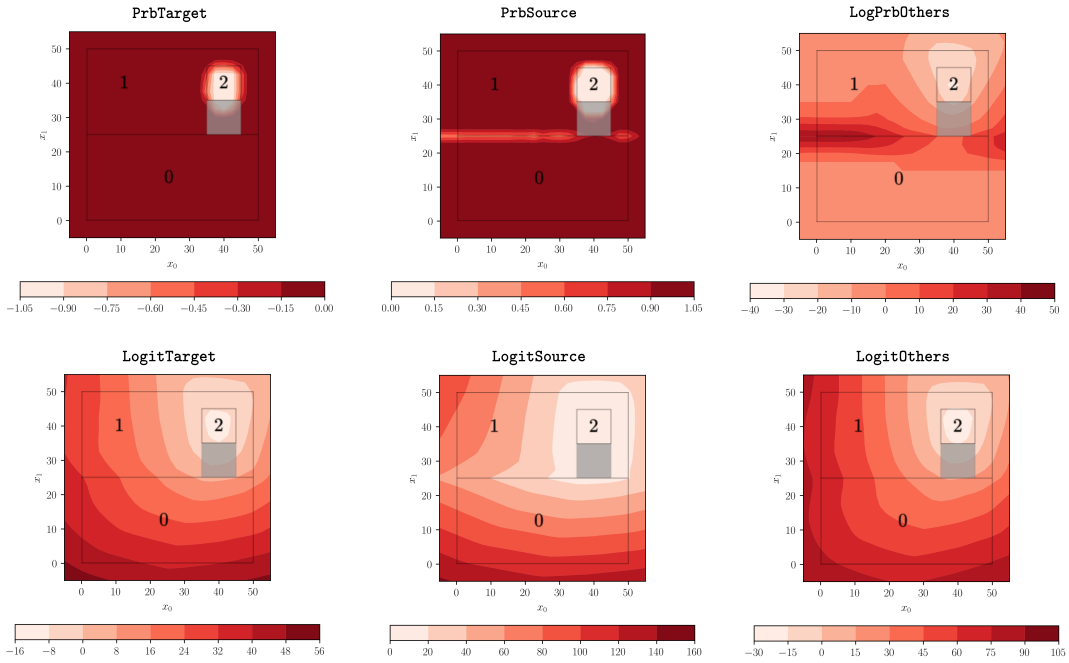
FIGURE 5.5: Validity rate for $\texttt{LogitSource}_{\lambda}$ for different values of λ.

From this plot it is clear that in general letting $\lambda = 1$ leads to higher validity rate; however the validity is unusually high for $\lambda = 0.1$ compared to the others.

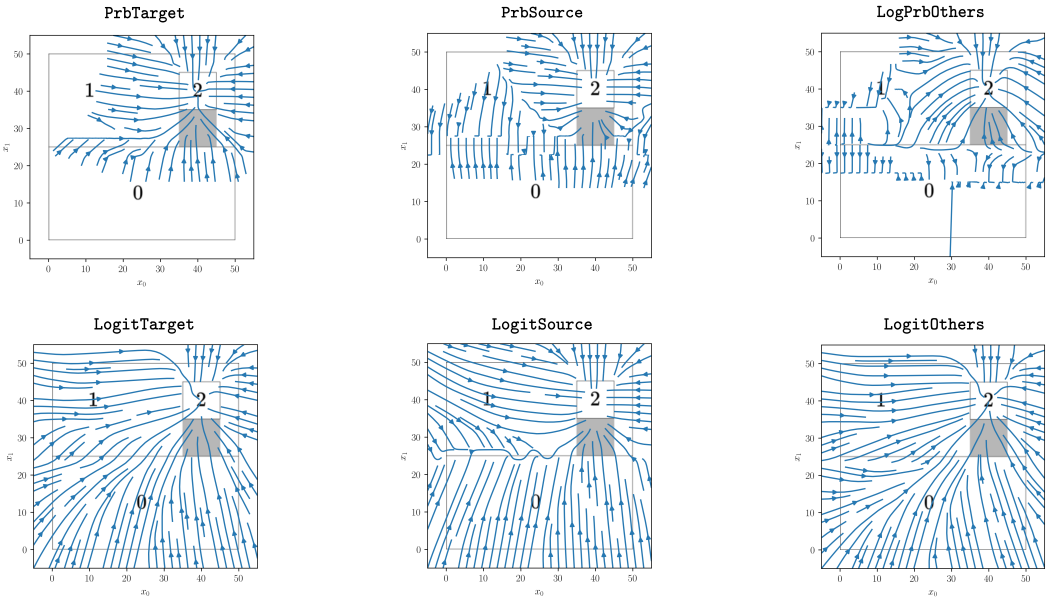
To make better sense of our findings given our considerations on gradients in subsection 4.2.2, we also plot our various losses and their gradients on `CakeOnSea`. Here, we select plots for some losses and where the target class is always class 2. For reference, the full plots are shown in Appendix B.

Interpretation From the plot values in the probability-based losses in Figure 5.6a, we can tell that for `CakeOnSea` the prediction probabilities are all close to 1, except perhaps near the decision boundaries. Hence, we can verify our intuitive thinking from subsection 4.2.2: for the probability-based losses, the loss is very flat, i.e. the gradient is close to zero. For `LogPrbOthers` it seems the logits from the other classes counteract the target logit, so that paradoxically the paths get pushed away from class 2 in places. For the logit-based losses however, the behavior is fairly uniform across the whole data distribution, and so the gradients clearly point the path towards class 2.

This leads us to choose $\texttt{LogitSource}_{\lambda=1}$ as validity loss in the rest of our analyses.



(A) Loss profile for various losses on `CakeOnSea`, with target = 2.



(B) Opposite of the loss gradient for various losses on `CakeOnSea`, with target = 2.

FIGURE 5.6: Losses and opposite of their gradient for various losses on `CakeOnSea`. We show `PrbTarget`, `PrbSource`, `LogPrbSource`, `LogitTarget`, `LogitSource`, `LogitOthers` with $\lambda = 1$ and target = 2.

5.5 Experiment 2: Path regularized training

Having determined an appropriate validity loss to use for generating valid paths, we turn now to our objective of interpretability. In this experiment we seek to determine if Latent Shift equipped with a path regularized autoencoder can perform on the same level as `Revise`, increasing interpretability without compromising validity.

Our proxy for interpretability of a path is *the number of class boundary crossings*: the path should ideally transition from source to target without going through other classes. We thus add our `BoundaryCrossLoss` to the NLL when computing the loss of the autoencoder. Our final loss for training the autoencoder is

$$\mathscr{L}_{\text{NF}}(\phi) = \text{NLL}(\phi) + \texttt{BoundaryCrossLoss}(\text{path})$$

where path is a path generated with some CF method.

Description First, we generate some seeds. For a given dataset, we train one classifier, as in section 5.4. Then, for each path method and seed, we train two autoencoders: with and without path regularization.

Then, each autoencoder is used to produce explanation paths on the whole test set, like in section 5.4, and metrics are computed with the paths as input. Finally, for each explanation method, we aggregate the metrics obtained for each seed and report the standard error.

Baselines Since the distance parameter seems inconsequential from our previous results, in this experiment we use Latent Shift and `Revise` in their basic form, i.e. with no distance regularization for Latent Shift and with $\lambda_{\text{distance}} = 0.3$ for `Revise`.

Datasets For memory performance reasons (the loss involves paths in the form of three-dimensional tensors) we reduce the batch size for each dataset compared to the previous experiment. We compute the batch size so that one training epoch (one pass through $\mathscr{D}_{\text{train}}$) takes 20 steps, except for `ForestCover` where the batches are still too big. The final numbers are:

- `CakeOnSea`: 2881
- `ForestCover`: 5000
- `WineQuality`: 195
- `OnlineNewsPopularity`: 1190

Metrics We measure the validity rate, as before, as well as the number of boundary crossings *before reaching* target (so if the path goes directly from source to target the value is 0) which is the proxy for interpretability we chose. For this we use the masking procedure described in subsection 5.3.4.

For measuring realism, we use a specially-trained autoencoder with the same architecture as the others, trained using only the NLL as a loss. The realism score for x^{CF} is given by its NLL as computed by our special autoencoder; the metric is extended to the paths using path masking again.

We also measure the computational complexity (the duration for one computing one step of one path).

Results The results for each dataset and method are given in Table 5.3.

Path method	Latent Shift						$\texttt{Revise}_{\lambda_{\text{distance}=0.3}}$					
Path regularization	No			Yes			No			Yes		
	`CakeOnSea`											
Validity rate (%) (↑)	68.7	±	4.4	72.2	±	1.9	100.0	±	0.0	100.0	±	0.0
Mean NLL (↓)	3.7	±	1.0	3.0	±	8.8E-02	2.6	±	4.5E-02	2.7	±	2.2E-02
Δt_{step} (ns) (↓)	21.7	±	1.6	22.5	±	1.7	1085.9	±	102.3	1166.7	±	112.4
Mean #BC (↓)	0.25	±	0.03	0.24	±	0.02	2.05	±	0.70	0.92	±	0.06
	`ForestCover`											
Validity rate (%) (↑)	55.3	±	2.3	55.5	±	2.4	87.5	±	0.9	87.4	±	0.6
Mean NLL (↓)	78	±	11	45	±	1.9	6.3	±	0.51	6.1	±	0.83
Δt_{step} (ns) (↓)	22.8	±	1.9	23.9	±	1.2	1254.2	±	105.0	1321.0	±	41.2
Mean #BC (↓)	0.86	±	0.05	0.86	±	0.07	2.77	±	0.06	2.86	±	0.18
	`OnlineNewsPopularity`											
Validity rate (%) (↑)	42.7	±	1.3	41.8	±	0.9	44.2	±	2.0	42.8	±	2.4
Mean NLL (↓)	37	±	3.0	33	±	1.8	33	±	2.7	30	±	0.96
Δt_{step} (ns) (↓)	202.0	±	7.0	205.6	±	7.8	14104.7	±	132.5	14014.7	±	20.9
Mean #BC (↓)	0.23	±	0.01	0.24	±	0.01	1.54	±	0.12	1.52	±	0.13
	`WineQuality`											
Validity rate (%) (↑)	73.3	±	1.5	73.0	±	1.4	78.7	±	1.8	77.9	±	1.5
Mean NLL (↓)	10	±	4.7E-02	10	±	7.0E-02	9.8	±	5.3E-02	9.7	±	4.8E-02
Δt_{step} (ns) (↓)	562.9	±	2.8	652.3	±	52.5	36485.2	±	143.6	36688.4	±	319.3
Mean #BC (↓)	0.17	±	0.00	0.17	±	0.00	0.16	±	0.01	0.17	±	0.01

TABLE 5.3: Path metrics with their standard error. (↑) indicates higher is better, (↓) indicates lower is better.

It seems the regularization did not achieve its objective, as the mean number of boundary crossings stays approximately the same regardless of whether or not regularization was performed. The only case where an effect is observed is for `Revise` on `CakeOnSea`. In no instance do we see a decrease in validity rate when applying path regularization. Similarly, path regularization has no effect on realism, except for Latent Shift on `CakeOnSea` and `ForestCover` where the NLL is decreased when applying regularization. The time per step does not depend on the autoencoder training, only on the architecture. Yet, we can observe the difference in time performance between Latent Shift and `Revise`: Latent Shift is up to 70 times faster than `Revise` on `OnlineNewsPopularity`.

Interpretation The regularization was not strong enough to alter the training of the autoencoder. Recall our loss function is given by

$$\texttt{BoundaryCrossLoss}(\text{path}) = -\frac{1}{S}\sum_{t=1}^{S} \max\{p^{(t)}_{\text{source}}, p^{(t)}_{\text{target}}\}$$

which is in $[-1, 0]$. By contrast, the NLL is computed according to a Gaussian prior distribution which can typically be as large as H (see subsection 4.1.2), which is larger than 10 for all datasets except for `CakeOnSea` where $H = 2$. Therefore, one easy improvement is to multiply the `BoundaryCrossLoss` by H.

5.6 Experiment 3: Robustness of explanations with path regularization

Given that we were able to achieve more interpretable paths using path regularization, we ask whether it is possible to meet any interpretability criterion, even a nonsensical one. For example, can we use path regularization so that all paths pass through one designated point, regardless of x and target, while staying valid? To do so, we select a point $x^\bullet \in \mathcal{X}$ that we can reasonably expect to be unrealistic, but is not completely out of reach compared to points in the train set. Then, we devise a path loss that encodes the distance from a path to $x^\bullet$, and let our path regularization training enforce this constraint. The loss is computed as:

$$\mathscr{L}_{\text{path}}(x^\bullet, (z_t)_{t=1}^T) = \min_{t\in\{1,\dots,T\}} \{\|\text{Dec}(z_t) - x^\bullet\|_2\}$$

Description We perform path regularized training using an adversarial path loss. The loss encodes how close a path passes by a given point $x^\bullet$, which is potentially unrealistic. In our experiment, we pick a $x^\bullet$ that is easy to compute, likely unrealistic yet not completely out of bounds for the dataset:

$$x^\bullet = (\max\{x_i : x \in \mathscr{D}_{\text{train}}\})_{i=1}^D \tag{5.1}$$

i.e. $x^\bullet_i$ is the largest value encountered in the train set for feature i.

Apart from this the setup is the same as that described in section 5.5, where we compare paths with the distance path regularization and without.

Baselines We use the same baselines as in the previous experiment (section 5.5).

Datasets We use our datasets with the following batch sizes:

- `CakeOnSea`: 2881
- `ForestCover`: 3000

- `WineQuality`: 195
- `OnlineNewsPopularity`: 1190

We reduced the batch size for `ForestCover` again for memory performance reasons.

Metrics As before, we measure the validity rate, the time performance as well as the NLL (with masking).

In addition, we measure the mean distance from a path to $x^\bullet$ *before the target is reached* using the masking procedure.

Results The results for each dataset and method are given in Table 5.4.

Path method	Latent Shift						$\texttt{Revise}_{\lambda_{\text{distance}=0.3}}$					
Path regularization	No			Yes			No			Yes		
	`CakeOnSea`											
Validity rate (%) (↑)	68.5	±	4.3	67.9	±	1.6	100.0	±	0.0	100.0	±	0.0
Mean NLL (↓)	3.5	±	0.8	2.7	±	< 0.1	2.6	±	< 0.1	2.9	±	< 0.1
Δt_{step} (ns) (↓)	14.9	±	0.5	17.4	±	0.6	733.4	±	57.2	780.7	±	65.2
Mean distance to $x^\bullet$ (↓)	2.09	±	0.06	2.05	±	0.04	1.81	±	0.05	1.80	±	0.02
	`ForestCover`											
Validity rate (%) (↑)	55.3	±	0.5	29.5	±	2.5	85.8	±	0.2	50.0	±	3.0
Mean NLL (↓)	142	±	18	4.2E+04	±	1.7E+04	5.0	±	< 0.4	1.7E+04	±	5.8E+02
Δt_{step} (ns) (↓)	28.9	±	0.3	29.7	±	0.4	1625.9	±	1.3	1617.0	±	8.3
Mean distance to $x^\bullet$ (↓)	14.72	±	0.07	9.80	±	0.77	14.86	±	0.02	6.50	±	0.26
	`OnlineNewsPopularity`											
Validity rate (%) (↑)	43.1	±	1.4	41.4	±	10.0	43.6	±	2.1	46.5	±	4.0
Mean NLL (↓)	37.2	±	3.3	4.5E+03	±	4.2E+03	34	±	< 3	336	±	31
Δt_{step} (ns) (↓)	59.6	±	0.8	62.7	±	2.2	3882.7	±	45.3	3884.2	±	60.3
Mean distance to $x^\bullet$ (↓)	371.78	±	0.06	368.56	±	2.68	371.83	±	0.04	371.13	±	0.38
	`WineQuality`											
Validity rate (%) (↑)	72.8	±	2.5	77.5	±	3.4	78.8	±	0.9	84.1	±	2.8
Mean NLL (↓)	10	±	< 0.1	14	±	< 2	10	±	<0.1	11	±	< 1
Δt_{step} (ns) (↓)	99.5	±	9.1	91.2	±	1.3	4907.3	±	14.2	5038.4	±	41.7
Mean distance to $x^\bullet$ (↓)	29.21	±	0.00	28.85	±	0.15	29.25	±	0.01	29.06	±	0.02

TABLE 5.4: Path metrics with their standard error. (↑) indicates higher is better, (↓) indicates lower is better.

We see that for `CakeOnSea` and `WineQuality` the regularization had little effect.

However, it did induce changes in behavior for the two remaining datasets:

- for `ForestCover` the mean distance decreased and the NLL heavily increased;
- for `OnlineNewsPopularity` the mean distance did not change but the NLL heavily increased.

The time measurements are much more reasonable than in the previous experiment, although the trends are the same.

Interpretation The regularization did not apply uniformly for all datasets; the distance-based loss is not bounded to a small interval like the `BoundaryCrossLoss`, which could have had the opposite effect of causing the NLL not to be prioritized during training. More precisely, decreasing the distance should have the effect of increasing the NLL; but if the NLL increases too much then the total loss will increase, at which point the early stopping mechanism will fire.

Relatedly, the L_2 distance naturally grows as D increases, as a consequence of the curse of dimensionality; while the NLL is also affected as discussed in the previous experiment, this could make the training even more unstable.

This concludes our experiments in this project.

Chapter 6

Discussion and future work

6.1 Discussion of findings

In experiment 1, we confirmed that the probability-based losses do not naturally extend to the multi-class setting. We showed that log-probability- and logit-based losses held their promises, achieving higher validity rates. However, it is still unclear why the validity rate spikes around $\lambda = 0.1$, and whether this generalizes to the all losses apart from `LogitSource`. It is also difficult to see what drives the competition between logits theoretically, which would help explain why the log-probability losses are on par with the logit losses for all datasets except `CakeOnSea`. The fact that the classifier achieves worse accuracy for those other datasets is likely a factor, as in this case our theoretical findings do not necessarily hold: the cases we studied are when the predicted probability of the target class p_{target} is 0 or 1. In the other cases, it is possible that the losses prioritize moving away from the source class rather than towards the target class.

In experiment 2, we did not choose the correct settings to completely demonstrate the effectiveness of path regularization in increasing interpretability. Similarly in experiment 3, the adversarial loss was not adjusted well enough to affect autoencoder behavior.

One important consideration is how the path loss is combined with the NLL during training. For example, our `BoundaryCrossLoss` was computed based on predicted probabilities, but we could have resolved the scale issue by replacing them with logits or log-probabilities.

Additionally, our initial idea was to apply the masking scheme *also during training*, not just at test time: indeed, what we value is the interpretability of the *valid* paths, *before the target is reached*, so it does not make sense to penalize the model for things outside of this scope. Of course, using the masking scheme on its own could have caused the model to completely ignore validity; the extra $\mathscr{L}_{\text{CF}}$ term in the NF loss was intended to prevent this. Unfortunately, when applying all of these changes at the same time, stability problems arose that prompted us to approach simpler solutions first.

Our decision to use a normalized version of `Revise`, that is, without using more specialized gradient-descent algorithms, also potentially penalized this method compared to the original procedure presented by Joshi et al. [29].

6.2 Additional comments

In light of our results, we now discuss additional considerations on how we chose to conduct our project.

6.2.1 Categorical data

One of the specificities of tabular data compared to image data is that the former can contain categorical or ordinal features, which is challenging to deal with in machine learning and specifically with neural networks: the state-of-the-art to this day are still decision-tree based methods such as `XGBoost` [9] or `CatBoost` [52]. Also note the difficulty to measure distances between points with categorical data.

In our work, we adapted methods from image problems, in which the question of discrete features is not raised; many methods developed for tabular data treat these features as numerical and then apply rounding as needed. Since normalizing flows are invertible we would expect this not to be necessary; however, normalizing flows is especially ill-adapted to categorical data because they have to map their discrete distribution to a continuous one (e.g. a standard Gaussian), which is an unstable process. This was still considered an open problem as of 2021 [37], but some solutions have been explored: for example, dequantization consists in adding noise to the discrete features so that the distribution is not discrete but heavily multi-modal instead, so that learning is smoother and the points can be turned back to their discrete form with confidence (as long as the learning process succeeded) [27]. In the interest of simplicity we decided to leave this as future work.

6.2.2 `CakeOnSea`

The `CakeOnSea` dataset was produced as a way to easily visualize CF paths, as those could be drawn in two dimensions. However, this only served to highlight the difficulty of rating explanations visually: indeed, they did not seem to bring more information about the model's behavior than simply visualizing its decision boundaries. Several choices could have been made to highlight explanation performance more directly. For example, using a toy dataset where the best features are obvious to a human (such as a circle centered on the origin and radius 1 as class 0, and the rest as class 1), or a dataset generated using known causal rules, like those used in [31].

6.2.3 On "Diffeomorphic Counterfactuals"

The method presented in the "Diffeomorphic Counterfactuals" paper [17], which we discussed in section 3.2.2, is very similar to ours: using gradient descent in the latent space of a NF-based generative model to maximize the predicted probability of the target class. Hence, it seems important to us to highlight the differences between our work and theirs.

First, their work is only tested on image data, while we focus on applications for tabular data. Second, while in [17] the method is equivalent to minimizing our `PrbTarget`, we challenged this choice of loss and proposed alternatives that achieved more satisfactory results in terms of validity. Third, we introduced path regularization to make our method class-aware, which is necessary for enhancing the latent representation [45].

On another note, the authors of [17] recently published an update to their work on arXiv [18].

We now expound on some of the choices we made that further distinguish our work from the current literature.

Why no images?

Counterfactual explanations are most often applied to images, for the simple reason that it is much easier for humans to judge the quality of images, and in particular their realism. Since our methods are inspired from works that have been mostly tested on image-related problems, it is sensible to ask why we refrained from applying our methods to images.

For the reasons just evoked, methods specifically tested on and developed for tabular data are much rarer in the XAI literature. Thus, our work intended to address this lack of material in the hopes of providing insights specific to tabular data problems.

Additionally, though some works try applying their methods both to image and tabular data, in this work we tried to keep away from the subjectivity and unfalsifiable character of some XAI research [41]; because images are so easily understood by humans, it can also happen that visual explanations are overanalyzed and interpreted wrongly.

Why NICE?

There are many normalizing flow architectures [14, 35, 15] (see [37] for a comprehensive review), and even combinations between normalizing flows and other generative models such as VAEs [53]. One reason for choosing NICE is its simplicity and therefore ease of implementation for our proof-of-concepts. However, NFs also have undesirable topological properties [11] that could be addressed in future work by stating more precise requirements on the latent representation.

Chapter 7

Conclusion

In this work, we studied counterfactual explanation methods for tabular data. We gave an overview of explainable AI, distinguished different methods and their scope, settling on CFX for their ease of comprehension and ability to provide recourse. Then, we reviewed the literature on CFX and found many methods based on performing gradient descent in a latent space learned through a generative model such as a variational autoencoder or a normalizing flow, which help keep the CFX search within the data manifold and thus guarantee that the CFX are realistic and useful. Each work gave different criteria for the quality of an explanation, which shows that progress can still be made in this regard.

We drew upon one of these methods, Latent Shift, as a simple and fast method that showed promise on image data, in particular by displaying the progression from a prediction to its counterfactual, something that is seldom shown in the literature despite the significant amount of iterative methods. We expanded on it in several key ways:

1. We applied it to tabular data while the original Latent Shift paper focused on image data, where explanation quality is easier to define.
2. We implemented it using NICE—an architecture for normalizing flows—as the generative model, to ensure trustworthiness in the produced explanation paths, whereas a basic autoencoder was used in the original Latent Shift paper.
3. We applied it to multi-class classification problems where the original work focused on binary classification. In doing so, we introduced and analyzed different candidates for the right loss function to minimize to achieve high validity.
4. In an effort to compete with the usual gradient descent methods, such as `Revise` [29], we formulated the path regularization procedure, to embed interpretability criteria within the latent space, whereas the authors of [10] explicitly refrained from tailoring the latent representation to the task. We achieved modest results with our experiment settings, but we are confident that this procedure is theoretically sound.

5. We probed the path regularization procedure to test its boundaries, in an effort to show that it can be misused to produce nonsensical explanations.

In future work, we would consider attempting our analyses with more sensible experimental parameters and exploring the landscape for more appropriate latent representations.

Appendix A

Theoretical details and proofs

Gradient of softmax Let us compute the gradient of softmax in general:

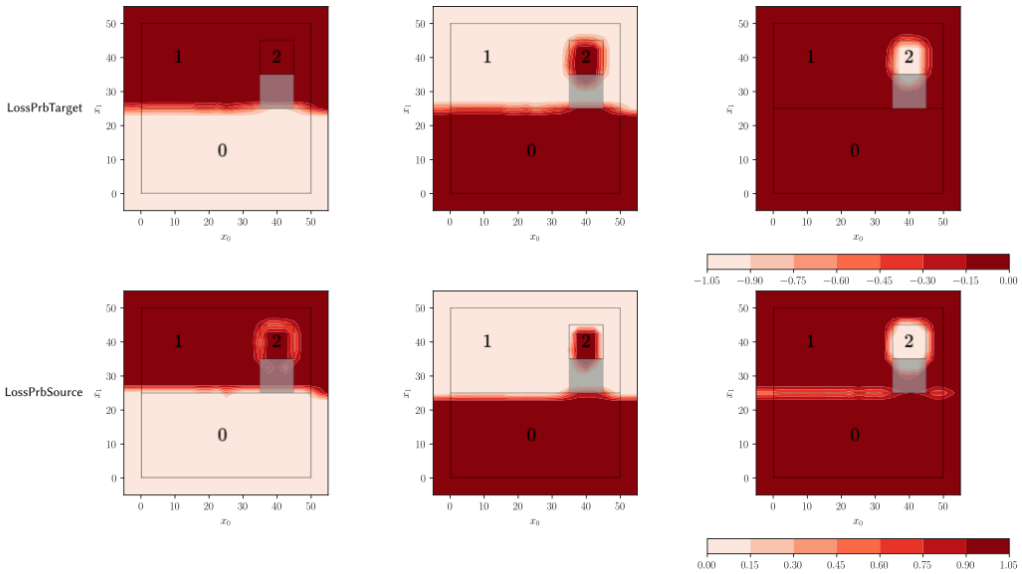
$$\begin{aligned}
\nabla_{u_j}\mathrm{softmax}(u)_i &= \nabla_{u_j}\frac{\mathrm{e}^{u_i}}{\sum_{c\in C}\mathrm{e}^{u_c}} \\
&= \frac{\nabla_{u_j}\mathrm{e}^{u_i}\sum_{c\in C}\mathrm{e}^{u_c} - \mathrm{e}^{u_i}\nabla_{u_j}\sum_{c\in C}\mathrm{e}^{u_c}}{\left(\sum_{c\in C}\mathrm{e}^{u_c}\right)^2} \\
&= \frac{\delta_{ij}\mathrm{e}^{u_i}\sum_{c\in C}\mathrm{e}^{u_c} - \mathrm{e}^{u_i}\mathrm{e}^{u_j}}{\left(\sum_{c\in C}\mathrm{e}^{u_c}\right)^2} \\
&= \delta_{ij}\frac{\mathrm{e}^{u_i}}{\sum_{c\in C}\mathrm{e}^{u_c}} - \frac{\mathrm{e}^{u_i}}{\sum_{c\in C}\mathrm{e}^{u_c}}\frac{\mathrm{e}^{u_j}}{\sum_{c\in C}\mathrm{e}^{u_c}} \\
&= \mathrm{softmax}(u)_i(\delta_{ij} - \mathrm{softmax}(u)_j).
\end{aligned}$$

Gradient of $\texttt{LogPrbOthers}_\lambda$ We have

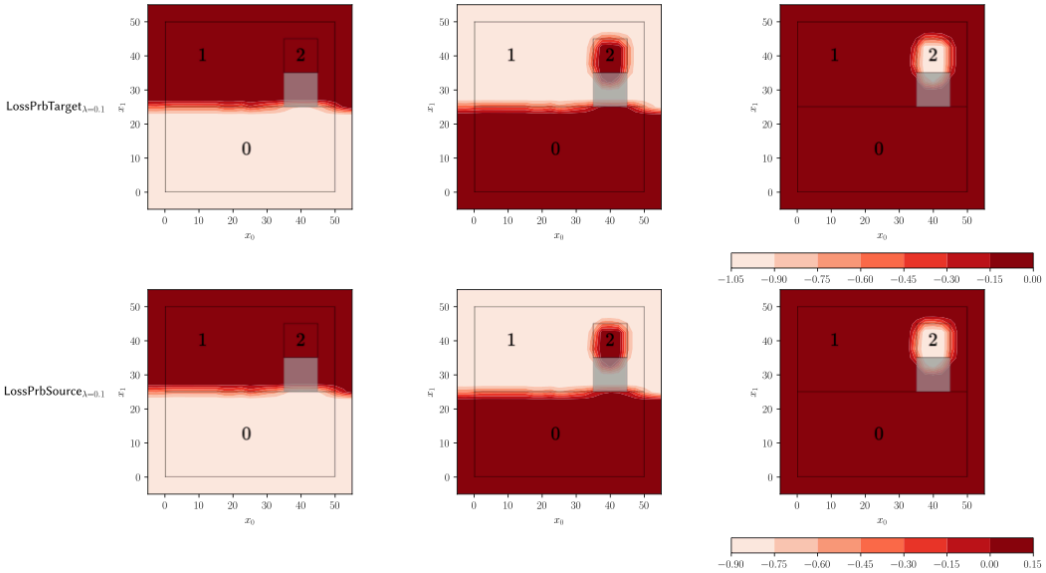
$$
\begin{aligned}
\nabla_{u_j}\texttt{LogPrbOthers}_\lambda &= -\nabla_{u_j}\log p_{\text{target}} + \lambda \sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} \nabla_{u_j}\log p_c \\
&= -(\delta_{j,\text{target}} - p_j) + \lambda \sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} (\delta_{cj} - p_j) \\
&= -\delta_{j,\text{target}} + p_j + \lambda\Big(\sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} \delta_{cj}\Big) - \lambda \sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} p_j \\
&= -\delta_{j,\text{target}} + p_j + \lambda\Big(\sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} \delta_{cj}\Big) - \lambda p_j \sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} 1 \\
&= -\delta_{j,\text{target}} + p_j + \lambda\Big(\sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} \delta_{cj}\Big) - \lambda p_j(C-1) \\
&= -\delta_{j,\text{target}} + p_j(1-\lambda(C-1)) + \lambda\Big(\sum_{\substack{c=1 \\ c\neq\text{target}}}^{C} \delta_{cj}\Big) \\
&= \begin{cases} -1 + p_{\text{target}}(1-\lambda(C-1)) & \text{if } j = \text{target} \\ 0 + p_j(1-\lambda(C-1)) + \lambda & \text{otherwise} \end{cases} \\
&= -\delta_{j,\text{target}} + p_j(1-\lambda(C-1)) + \lambda(1-\delta_{j,\text{target}}) \\
&= p_j(1-\lambda(C-1)) + \lambda(1-\delta_{j,\text{target}}) - \delta_{j,\text{target}}.
\end{aligned}
$$

Appendix B

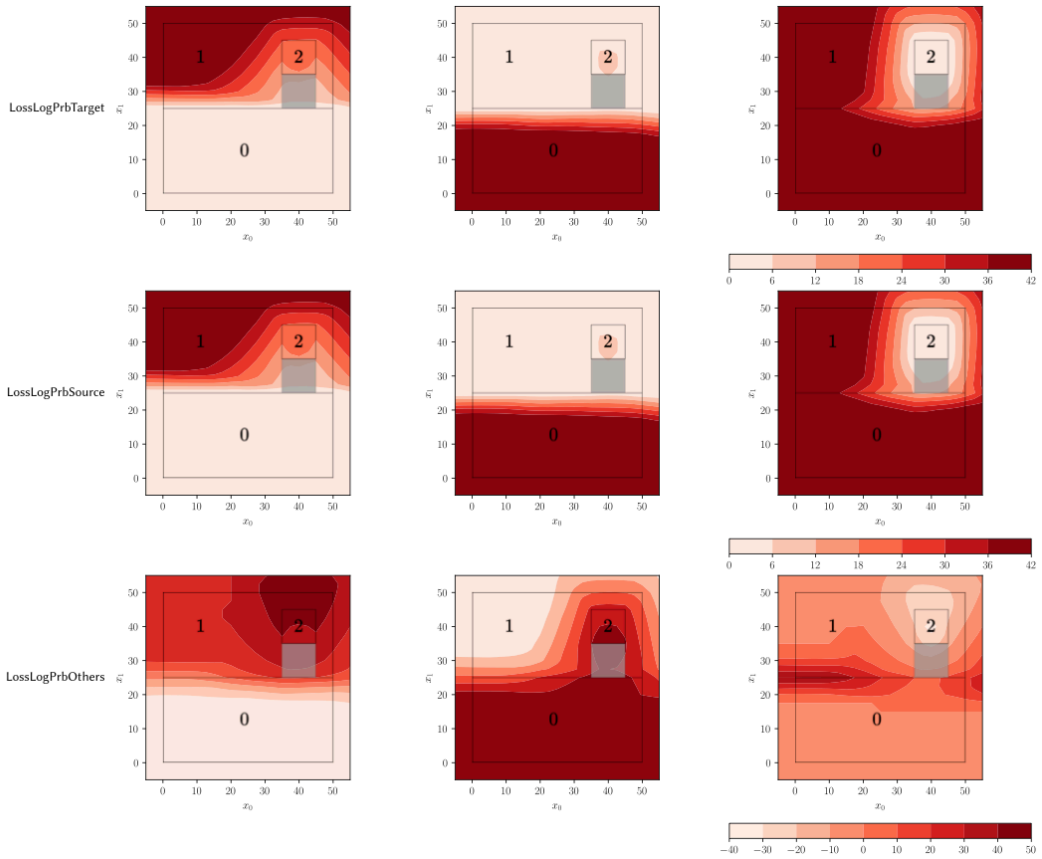
Supplementary visualizations



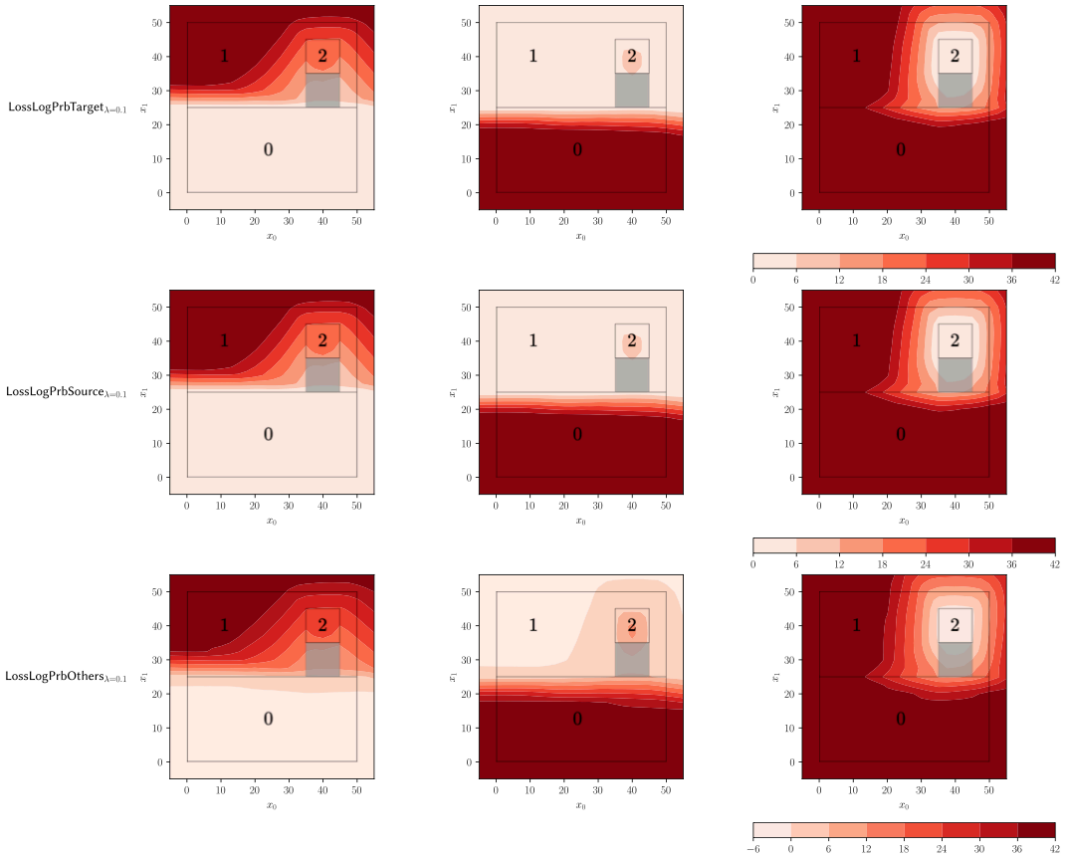
(A) Probability-based losses with $\lambda = 1$.



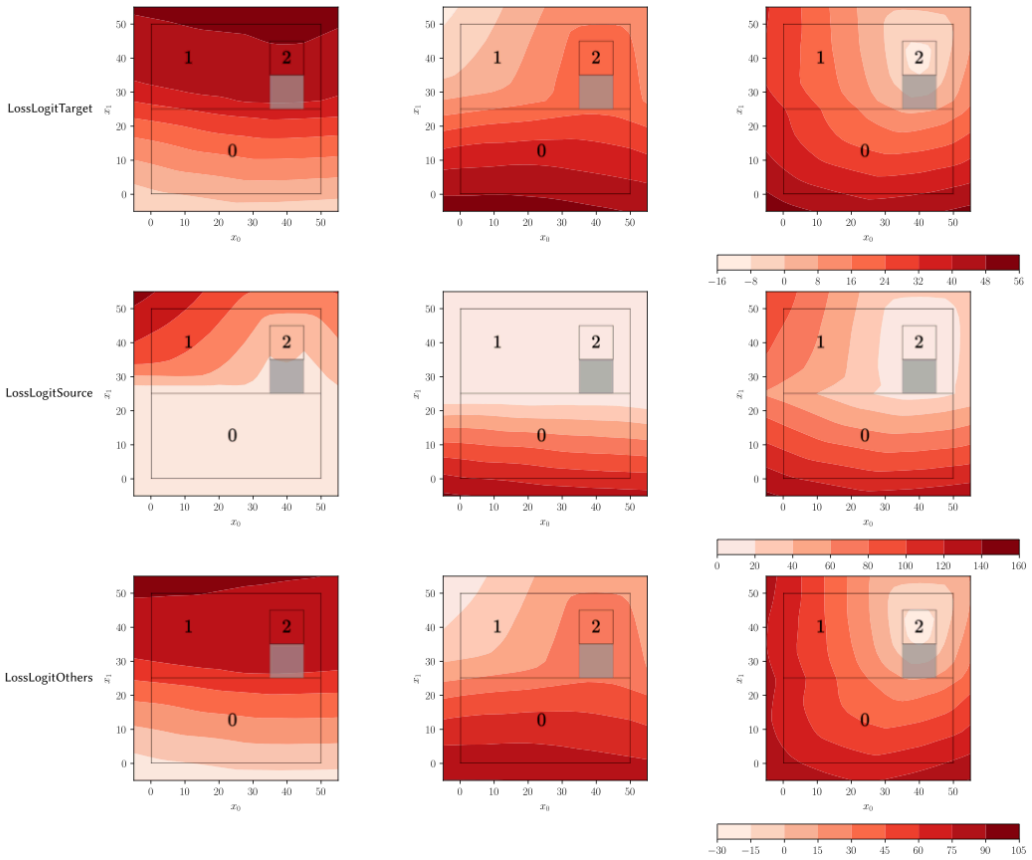
(B) Probability-based losses with $\lambda = 0.1$.



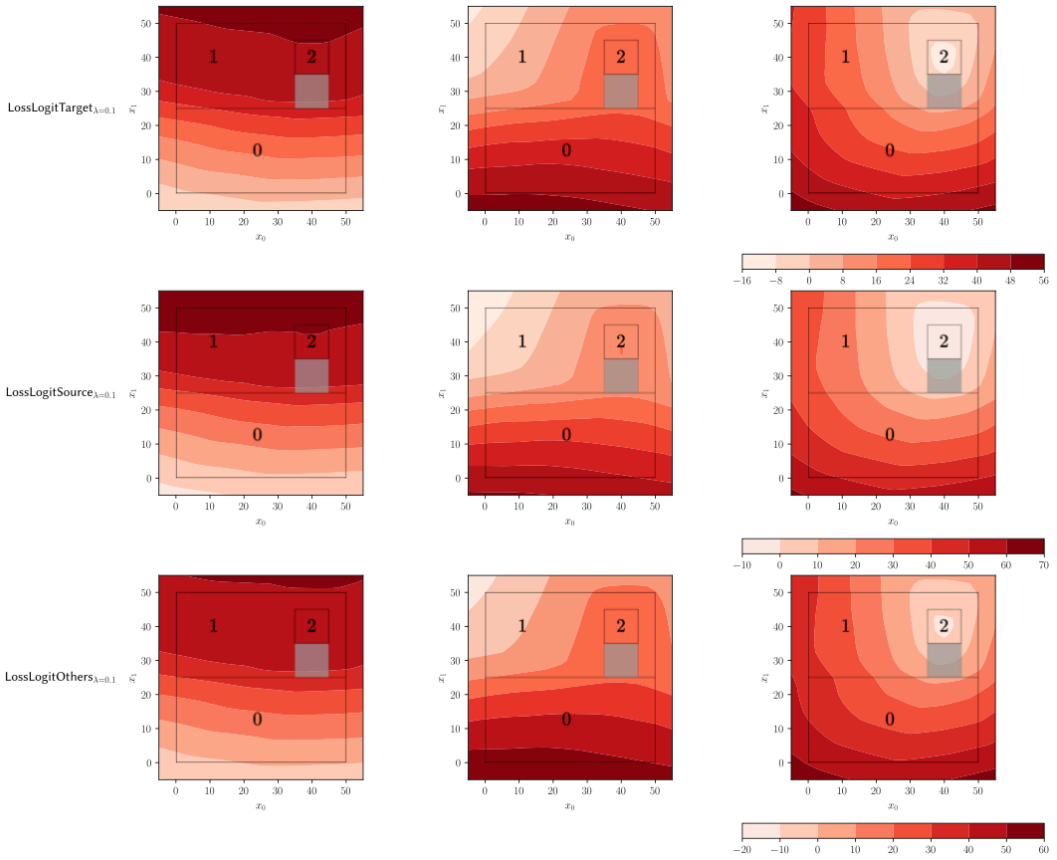
(c) Log-probability-based losses with $\lambda = 1$.



(d) Log-probability-based losses with $\lambda = 0.1$.

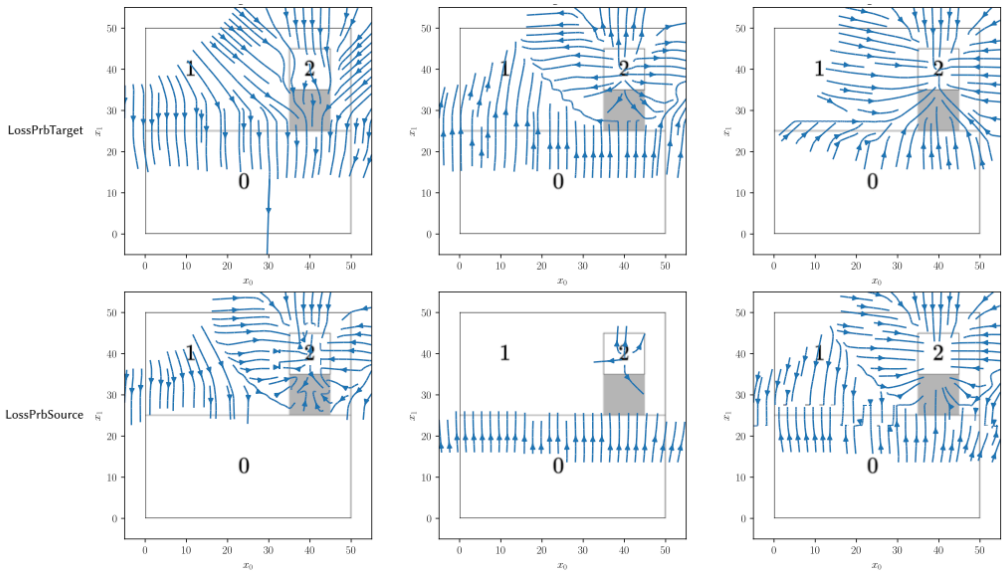


(E) Logit-based losses with $\lambda = 1$.

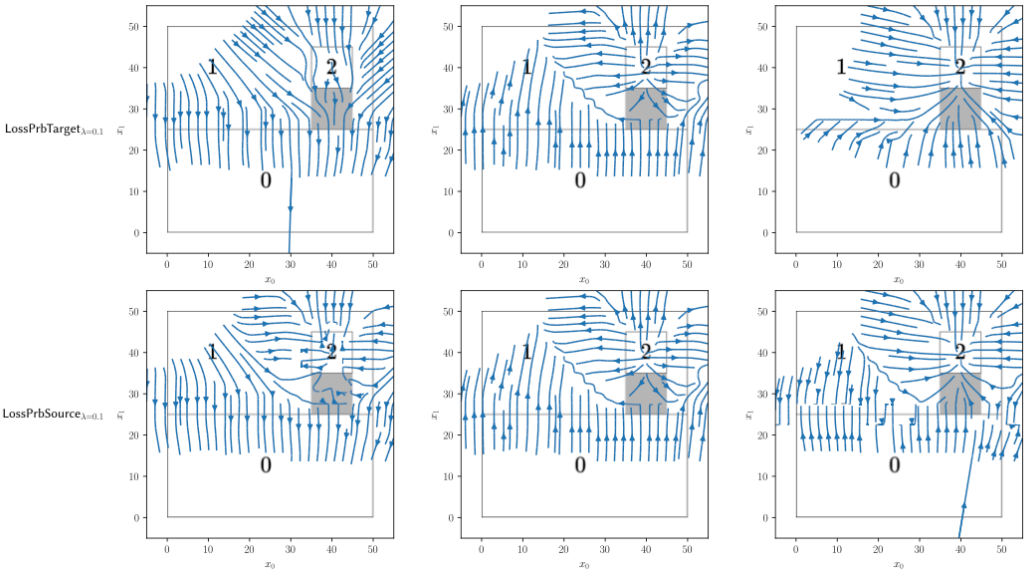


(F) Logit-based losses with $\lambda = 0.1$.

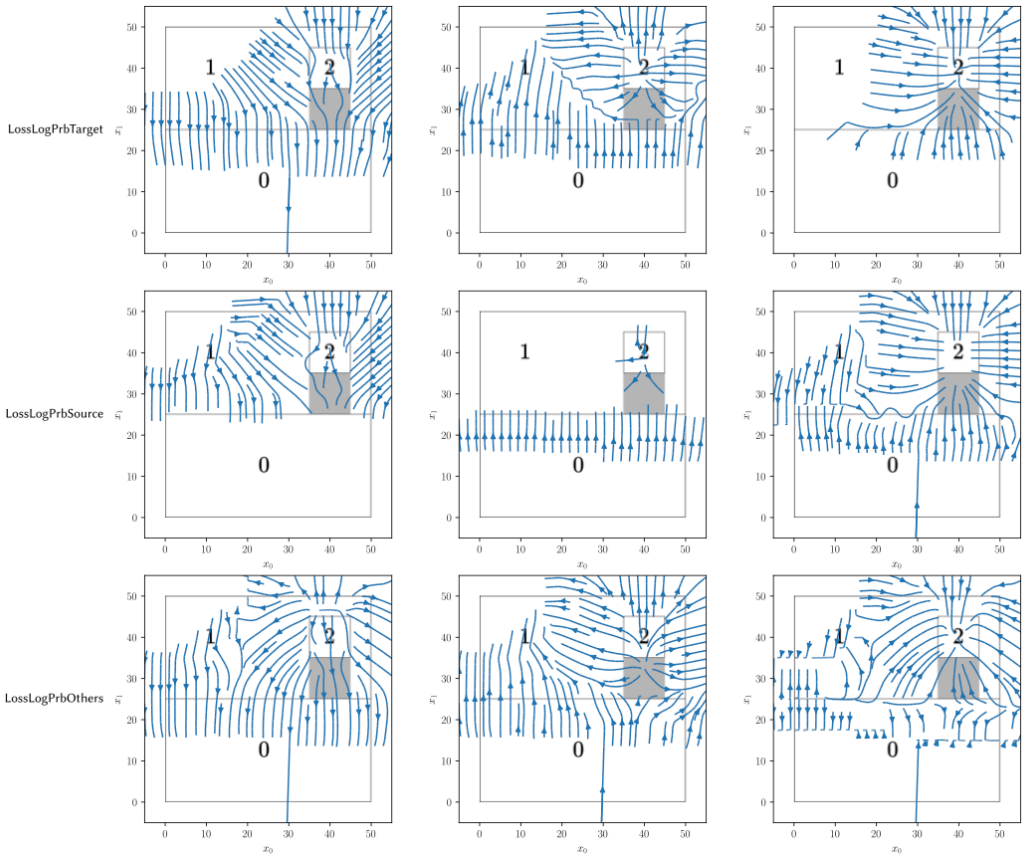
FIGURE B.1: Validity losses on `CakeOnSea`. The column indicates the target class (0 on the left, 1 in the middle, 2 on the right). Each colorbar applies to the row of plots right above it.



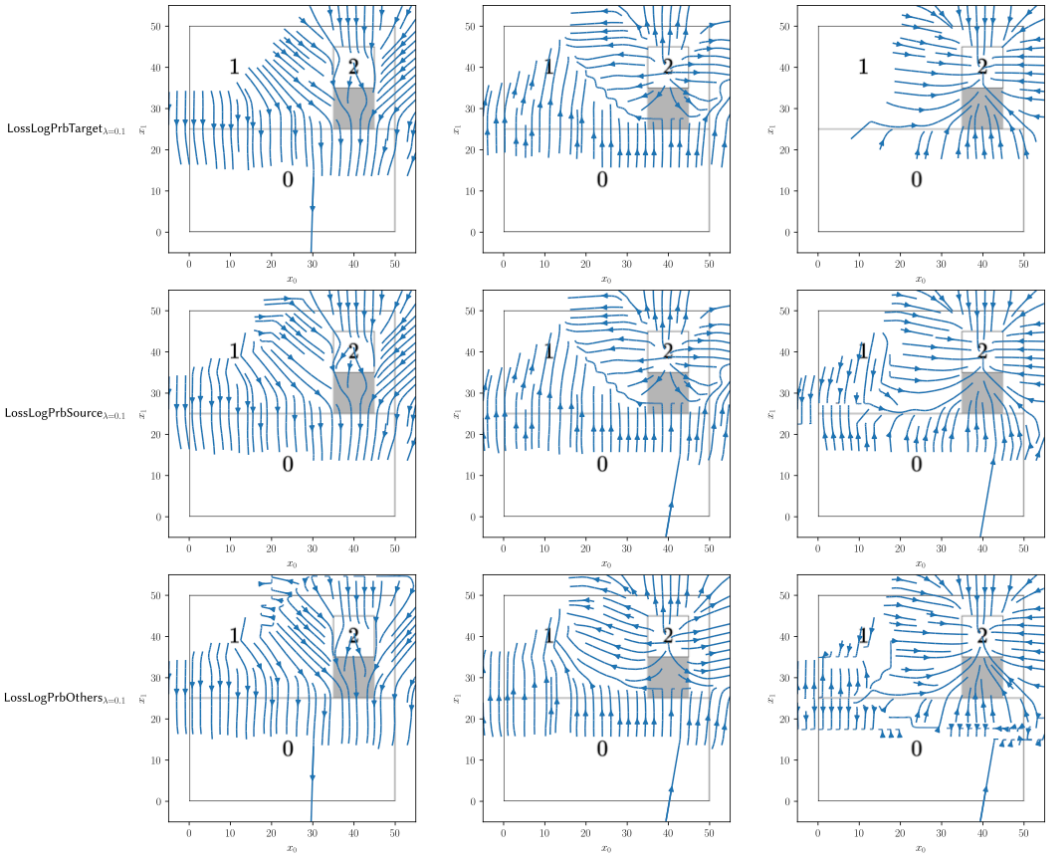
(A) Probability-based losses with $\lambda = 1$.



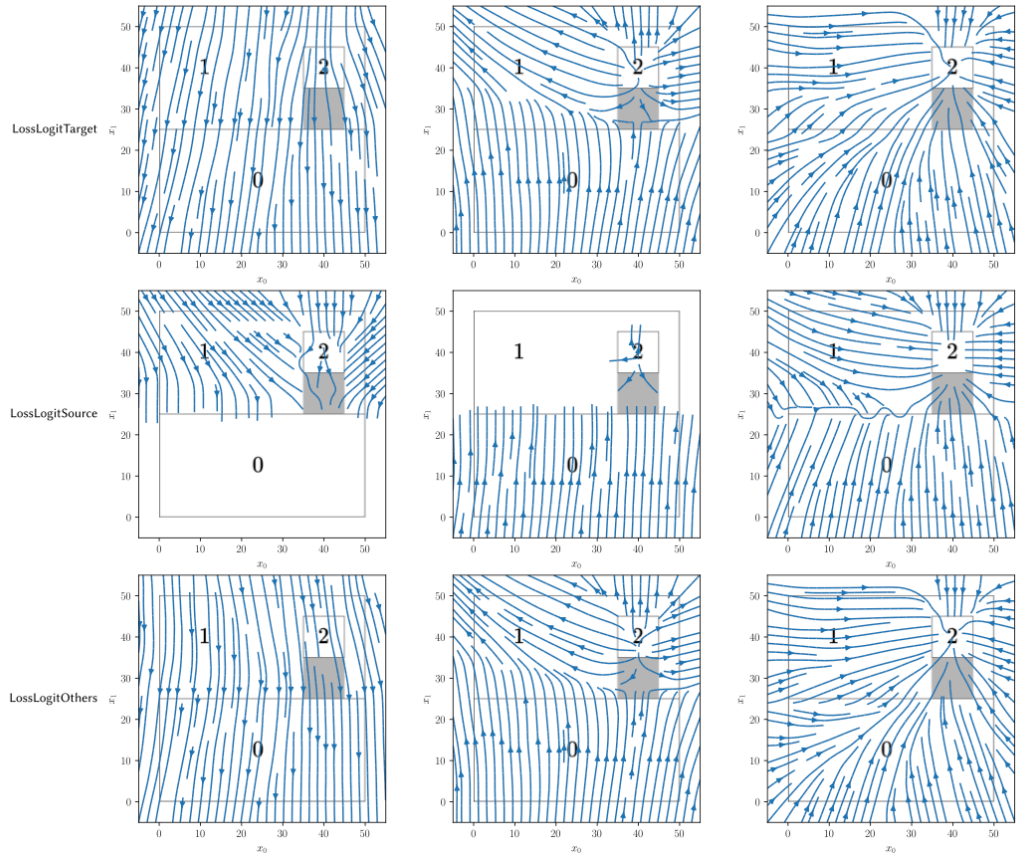
(B) Probability-based losses with $\lambda = 0.1$.



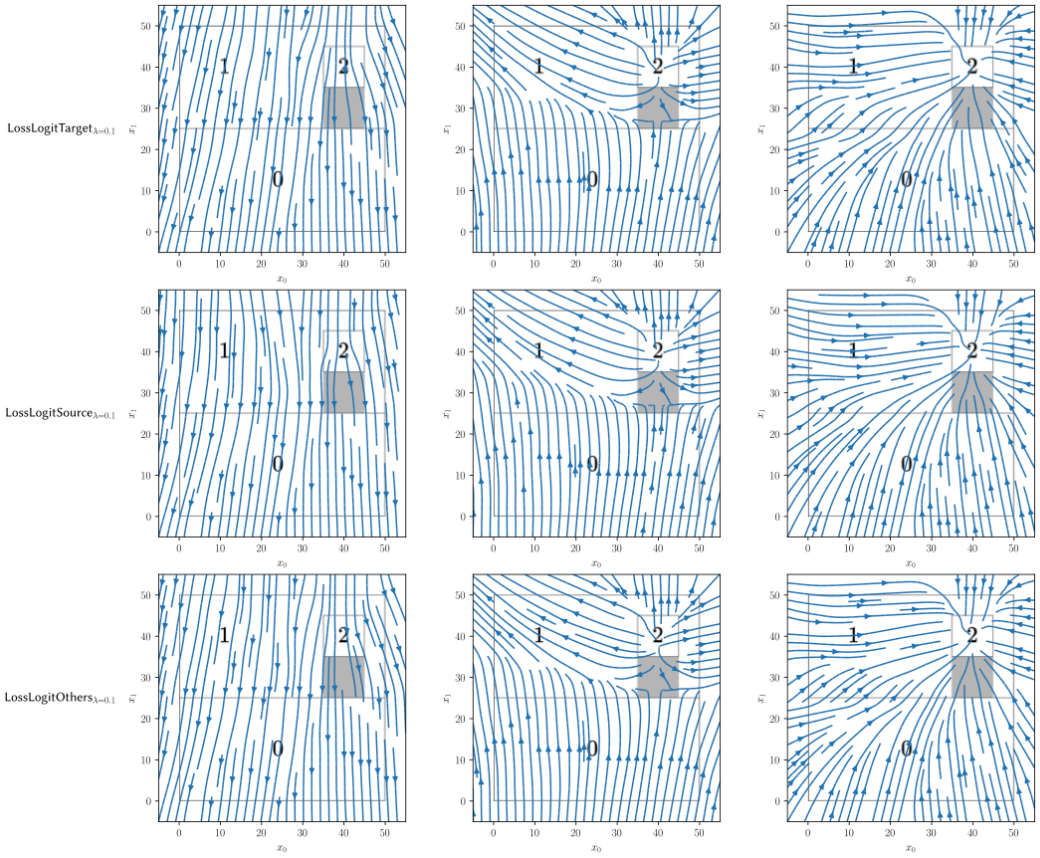
(c) Log-probability-based losses with $\lambda = 1$.



(d) Log-probability-based losses with $\lambda = 0.1$.



(E) Logit-based losses with $\lambda = 1$.



(F) Logit-based losses with $\lambda = 0.1$.

FIGURE B.2: Gradients of validity losses on `CakeOnSea`. The column indicates the target class.

Bibliography

[1] Javier Antorán et al. "Getting a CLUE: A Method for Explaining Uncertainty Estimates". In: International Conference on Learning Representations. Mar. 17, 2021. URL: https://openreview.net/forum?id=XSLF1XFq5h (visited on 10/06/2022) (pages 17–19).

[2] Andrew Bai et al. *Concept Gradient: Concept-based Interpretation Without Linear Assumption*. Version 1. 2022. DOI: 10.48550/ARXIV.2208.14966. URL: https://arxiv.org/abs/2208.14966 (visited on 12/19/2022). preprint (page 13).

[3] Vaishak Belle and Ioannis Papantonis. "Principles and Practice of Explainable Machine Learning". In: *Frontiers in Big Data* 4 (2021). ISSN: 2624-909X. URL: https://www.frontiersin.org/articles/10.3389/fdata.2021.688969 (visited on 09/05/2022) (pages 1, 2).

[4] Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (Aug. 2013), pp. 1798–1828. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2013.50. URL: http://ieeexplore.ieee.org/document/6472238/ (visited on 03/13/2023) (page 5).

[5] Jock A. Blackard and Denis J. Dean. "Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables". In: *Computers and Electronics in Agriculture* 24.3 (Dec. 1999), pp. 131–151. ISSN: 01681699. DOI: 10.1016/S0168-1699(99)00046-0. URL: https://linkinghub.elsevier.com/retrieve/pii/S0168169999000460 (visited on 03/06/2023) (page 34).

[6] Markus M. Breunig et al. "LOF: Identifying Density-Based Local Outliers". In: *ACM SIGMOD Record* 29.2 (May 16, 2000), pp. 93–104. ISSN: 0163-5808. DOI: 10.1145/335191.335388. URL: https://doi.org/10.1145/335191.335388 (visited on 10/10/2022) (pages 19, 38).

[7] Michael M. Bronstein et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". Version 2. In: (2021). DOI: 10.48550/ARXIV.2104.13478. URL: https://arxiv.org/abs/2104.13478 (visited on 04/26/2022) (page 6).

[8] Bradley CA Brown et al. "The Union of Manifolds Hypothesis". In: NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations. Nov. 9, 2022. URL: https://openreview.net/forum?id=aJp8UXRKvVm (visited on 03/13/2023) (pages 6, 18).

[9] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 13, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785 (visited on 03/13/2023) (page 51).

[10] Joseph Paul Cohen et al. "Gifsplanation via Latent Shift: A Simple Autoencoder Approach to Counterfactual Generation for Chest X-rays". In: Medical Imaging with Deep Learning. July 22, 2022. URL: https://openreview.net/forum?id=rnunjvgxAMt (visited on 09/28/2022) (pages 20, 21, 23, 53).

[11] Robert Cornish et al. "Relaxing Bijectivity Constraints with Continuously Indexed Normalising Flows". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 2133–2143. URL: http://proceedings.mlr.press/v119/cornish20a.html (visited on 03/16/2023) (page 52).

[12] Paulo Cortez et al. "Modeling Wine Preferences by Data Mining from Physicochemical Properties". In: *Decision Support Systems* 47.4 (Nov. 2009), pp. 547–553. ISSN: 01679236. DOI: 10.1016/j.dss.2009.05.016. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167923609001377 (visited on 03/06/2023) (page 34).

[13] Scott Cunningham. *Causal Inference: The Mixtape*. New Haven ; London: Yale University Press, 2021. 572 pp. ISBN: 978-0-300-25168-5 (page 14).

[14] Laurent Dinh, David Krueger, and Yoshua Bengio. "NICE: Non-linear Independent Components Estimation". In: *3rd International Conference on Learning Representations, {ICLR} 2015*. ICLR. arXiv, Apr. 10, 2015. DOI: 10.48550/arXiv.1410.8516. arXiv: 1410.8516 [cs]. URL: http://arxiv.org/abs/1410.8516 (visited on 11/21/2022) (pages 11, 22, 37, 52).

[15] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density Estimation Using Real NVP". In: International Conference on Learning Representations. July 21, 2022. URL: https://openreview.net/forum?id=HkpbnH9lx (visited on 03/13/2023) (page 52).

[16] Carl Doersch. *Tutorial on Variational Autoencoders*. Jan. 3, 2021. arXiv: arXiv:1606.05908. URL: http://arxiv.org/abs/1606.05908 (visited on 11/11/2022). preprint (pages 9, 10).

[17] Ann-Kathrin Dombrowski, Jan E. Gerken, and Pan Kessel. "Diffeomorphic Explanations with Normalizing Flows". In: ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models. June 24, 2021. URL: https://openreview.net/forum?id=ZBR9EpEl6G4 (visited on 03/10/2023) (pages 17, 18, 52).

[18] Ann-Kathrin Dombrowski et al. *Diffeomorphic Counterfactuals with Generative Models*. Version 2. June 16, 2022. DOI: 10.48550/arXiv.2206.05075. arXiv:

arXiv:2206.05075. URL: http://arxiv.org/abs/2206.05075 (visited on 03/13/2023). preprint (page 52).

[19] Michael Downs et al. "CRUDS: Counterfactual Recourse Using Disentangled Subspaces". In: *ICML Workshop on Human Interpretability in Machine Learning*. 2020, pp. 1–23. URL: https://finale.seas.harvard.edu/publications/cruds-counterfactual-recourse-using-disentangled-subspaces (pages 17, 18).

[20] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2019. URL: https://archive.ics.uci.edu/ml/datasets (page 34).

[21] Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez. "A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News". In: *Progress in Artificial Intelligence*. Ed. by Francisco Pereira et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 535–546. ISBN: 978-3-319-23485-4. DOI: 10.1007/978-3-319-23485-4_53 (page 35).

[22] Timo Freiesleben. "The Intriguing Relation Between Counterfactual Explanations and Adversarial Examples". In: *Minds and Machines* 32.1 (Mar. 1, 2022), pp. 77–109. ISSN: 1572-8641. DOI: 10.1007/s11023-021-09580-9. URL: https://doi.org/10.1007/s11023-021-09580-9 (visited on 03/14/2023) (page 3).

[23] Zixuan Geng, Maximilian Schleich, and Dan Suciu. "Computing Rule-Based Explanations by Leveraging Counterfactuals". In: *Proceedings of the VLDB Endowment* 16.3 (Nov. 1, 2022), pp. 420–432. ISSN: 2150-8097. DOI: 10.14778/3570690.3570693. URL: https://doi.org/10.14778/3570690.3570693 (visited on 03/15/2023) (pages 15, 18).

[24] Tessa Han, Suraj Srinivas, and Himabindu Lakkaraju. "Which Explanation Should I Choose? A Function Approximation Perspective to Characterizing Post Hoc Explanations". In: Advances in Neural Information Processing Systems. Oct. 31, 2022. URL: https://openreview.net/forum?id=rTvH1_SRyXs (visited on 03/06/2023) (page 3).

[25] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 03/15/2023) (page 22).

[26] Martin Heusel et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://papers.nips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html (visited on 03/14/2023) (page 5).

[27] Jonathan Ho et al. "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design". In: *Proceedings of the 36th*

International Conference on Machine Learning. International Conference on Machine Learning. PMLR, May 24, 2019, pp. 2722–2730. URL: https://proceedings.mlr.press/v97/ho19a.html (visited on 03/15/2023) (page 51).

[28] Alon Jacovi et al. "Contrastive Explanations for Model Interpretability". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2021. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 1597–1611. DOI: 10.18653/v1/2021.emnlp-main.120. URL: https://aclanthology.org/2021.emnlp-main.120 (visited on 10/04/2022) (page 13).

[29] Shalmali Joshi et al. "Towards Realistic Individual Recourse and Actionable Explanations in Black-Box Decision Making Systems". July 22, 2019. DOI: 10.48550/arXiv.1907.09615. arXiv: 1907.09615 [cs, stat]. URL: http://arxiv.org/abs/1907.09615 (visited on 12/09/2022) (pages 16, 18, 19, 22, 31, 51, 53).

[30] Margot E. Kaminski. *The Right to Explanation, Explained*. June 15, 2018. DOI: 10.2139/ssrn.3196985. URL: https://papers.ssrn.com/abstract=3196985 (visited on 02/28/2023). preprint (page 4).

[31] Amir-Hossein Karimi et al. "Algorithmic Recourse under Imperfect Causal Knowledge: A Probabilistic Approach". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 265–277. URL: https://proceedings.neurips.cc/paper/2020/hash/02a3c7fb3f489288ae6942498498db20-Abstract.html (visited on 10/07/2022) (pages 14, 51).

[32] Been Kim et al. "Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)". In: *Proceedings of the 35th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, July 3, 2018, pp. 2668–2677. URL: https://proceedings.mlr.press/v80/kim18d.html (visited on 03/02/2023) (page 13).

[33] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, {ICLR} 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. arXiv, 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980 (visited on 02/27/2023) (pages 22, 36).

[34] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, {ICLR} 2014*. Dec. 23, 2013. URL: https://openreview.net/forum?id=33X9fd2-9FyZd (visited on 03/15/2023) (page 10).

[35] Durk P Kingma and Prafulla Dhariwal. "Glow: Generative Flow with Invertible 1x1 Convolutions". In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/d139db6a236200b21cc7f752979132d0-Abstract.html (visited on 03/13/2023) (page 52).

[36] Jack Klys, Jake Snell, and Richard Zemel. "Learning Latent Subspaces in Variational Autoencoders". In: *Advances in Neural Information Processing Systems*. 32nd Conference on Neural Information Processing Systems (NeurIPS 2018). Vol. 31. Montréal, Québec, Canada: Curran Associates, Inc., 2018. ISBN: 978-1-5108-8447-2. arXiv: 1812.06190v1 [cs.LG]. URL: https://proceedings.neurips.cc/paper/2018/file/73e5080f0f3804cb9cf470a8ce895dac-Paper.pdf (page 17).

[37] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11 (Nov. 1, 2021), pp. 3964–3979. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2020.2992934. URL: https://ieeexplore.ieee.org/document/9089305/ (visited on 03/13/2023) (pages 51, 52).

[38] Mingi Kwon, Jaeseok Jeong, and Youngjung Uh. "Diffusion Models Already Have A Semantic Latent Space". In: International Conference on Learning Representations. Feb. 1, 2023. URL: https://openreview.net/forum?id=pd1P2eUBVfq (visited on 02/27/2023) (page 15).

[39] Thibault Laugel. "Local Post-hoc Interpretability for Black-box Classifiers". Theses. Sorbonne Université, CNRS, LIP6, F-75005 Paris, France, July 2020. URL: https://hal.archives-ouvertes.fr/tel-03002496 (visited on 10/10/2022) (page 3).

[40] Thibault Laugel et al. "Issues with Post-Hoc Counterfactual Explanations: A Discussion". June 11, 2019. DOI: 10.48550/arXiv.1906.04774. arXiv: 1906.04774 [cs, stat]. URL: http://arxiv.org/abs/1906.04774 (visited on 10/10/2022) (page 19).

[41] Matthew L. Leavitt and Ari Morcos. "Towards Falsifiable Interpretability Research". Oct. 22, 2020. arXiv: 2010.12016 [cs, stat]. URL: http://arxiv.org/abs/2010.12016 (visited on 11/18/2022) (pages 4, 52).

[42] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1989.1.4.541. URL: https://direct.mit.edu/neco/article/1/4/541-551/5515 (visited on 03/13/2023) (page 6).

[43] Brian Y. Lim, Anind K. Dey, and Daniel Avrahami. "Why and Why Not Explanations Improve the Intelligibility of Context-Aware Intelligent Systems". In: *Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009, Boston, MA, USA, April 4-9, 2009*. Ed. by Dan R. Olsen Jr et al. ACM, 2009, pp. 2119–2128. DOI: 10.1145/1518701.1519023 (page 3).

[44] Zachary C. Lipton. "The Mythos of Model Interpretability". In: *Communications of the ACM* 61.10 (Sept. 26, 2018), pp. 36–43. ISSN: 0001-0782. DOI: 10.1145/3233231. URL: https://doi.org/10.1145/3233231 (visited on 03/15/2023) (page 4).

[45] Francesco Locatello et al. "Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations". In: *Proceedings of the 36th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, May 24, 2019, pp. 4114–4124. URL: https://proceedings.mlr.press/v97/locatello19a.html (visited on 03/15/2023) (pages 15, 32, 52).

[46] Seyed-Mohsen Moosavi-Dezfooli et al. "Universal Adversarial Perturbations". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). July 2017, pp. 86–94. DOI: 10.1109/CVPR.2017.17 (pages 3, 15).

[47] Stephen L. Morgan and Christopher Winship. *Counterfactuals and Causal Inference Methods and Principles for Social Research*. Second edition. New York, NY: Cambridge University Press, 2015. ISBN: 978-1-107-58799-1 (page 14).

[48] Ramaravind Kommiya 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*. Jan. 27, 2020, pp. 607–617. DOI: 10.1145/3351095.3372850. arXiv: 1905.07697 [cs, stat]. URL: http://arxiv.org/abs/1905.07697 (visited on 12/19/2022) (pages 16, 18).

[49] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning Series. Cambridge, MA: MIT Press, 2012. 1067 pp. ISBN: 978-0-262-01802-9 (page 26).

[50] Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. "Learning Model-Agnostic Counterfactual Explanations for Tabular Data". In: *Proceedings of The Web Conference 2020*. WWW '20. New York, NY, USA: Association for Computing Machinery, Apr. 20, 2020, pp. 3126–3132. ISBN: 978-1-4503-7023-3. DOI: 10.1145/3366423.3380087. URL: https://doi.org/10.1145/3366423.3380087 (visited on 10/04/2022) (page 19).

[51] Judea Pearl, Madelyn Glymour, and Nicholas P. Jewell. *Causal Inference in Statistics: A Primer*. Chichester, West Sussex: Wiley, 2016. 136 pp. ISBN: 978-1-119-18684-7 (page 14).

[52] Liudmila Prokhorenkova et al. "CatBoost: Unbiased Boosting with Categorical Features". In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/14491b756b3a51daac41c24863285549-Abstract.html (visited on 03/13/2023) (page 51).

[53] Danilo Rezende and Shakir Mohamed. "Variational Inference with Normalizing Flows". In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, June 1, 2015, pp. 1530–1538. URL: https://proceedings.mlr.press/v37/rezende15.html (visited on 11/23/2022) (pages 18, 52).

[54] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. New York, NY, USA: Association for Computing Machinery, Aug. 13, 2016, pp. 1135–1144. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778 (visited on 09/22/2022) (page 2).

[55] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Anchors: High-Precision Model-Agnostic Explanations". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press, Feb. 2, 2018, pp. 1527–1535. ISBN: 978-1-57735-800-8 (page 3).

[56] Wojciech Samek et al., eds. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Vol. 11700. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-28953-9 978-3-030-28954-6. DOI: 10.1007/978-3-030-28954-6. URL: http://link.springer.com/10.1007/978-3-030-28954-6 (visited on 03/20/2020) (page 3).

[57] Ravid Shwartz-Ziv and Amitai Armon. "Tabular Data: Deep Learning Is Not All You Need". In: *Information Fusion* 81 (May 2022), pp. 84–90. ISSN: 15662535. DOI: 10.1016/j.inffus.2021.11.011. URL: https://linkinghub.elsevier.com/retrieve/pii/S1566253521002360 (visited on 03/14/2023) (page 6).

[58] Dylan Slack et al. "Fooling LIME and SHAP: Adversarial Attacks on Post Hoc Explanation Methods". In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*. AIES '20. New York, NY, USA: Association for Computing Machinery, Feb. 7, 2020, pp. 180–186. ISBN: 978-1-4503-7110-0. DOI: 10.1145/3375627.3375830. URL: https://doi.org/10.1145/3375627.3375830 (visited on 03/06/2023) (page 2).

[59] Ilia Stepin et al. "A Survey of Contrastive and Counterfactual Explanation Generation Methods for Explainable Artificial Intelligence". In: *IEEE Access* 9 (2021), pp. 11974–12001. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3051315. URL: https://www.researchgate.net/publication/348697583_A_Survey_of_Contrastive_and_Counterfactual_Explanation_Generation_Methods_for_Explainable_Artificial_Intelligence (page 13).

[60] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic Attribution for Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, Aug. 6, 2017, pp. 3319–3328 (page 2).

[61] Christian Szegedy et al. "Intriguing Properties of Neural Networks". In: *International Conference on Learning Representations*. 2014. URL: http://arxiv.org/abs/1312.6199 (visited on 01/09/2023) (page 3).

[62] Arnaud Van Looveren and Janis Klaise. "Interpretable Counterfactual Explanations Guided by Prototypes". In: *Machine Learning and Knowledge Discovery in Databases. Research Track*. Ed. by Nuria Oliver et al. Vol. 12976. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 650–665. ISBN: 978-3-030-86519-1 978-3-030-86520-7. DOI: 10.1007/978-3-030-86520-7_40. URL: https://link.springer.com/10.1007/978-3-030-86520-7_40 (visited on 11/30/2022) (pages 16, 18, 19, 38).

[63] Sahil Verma, John Dickerson, and Keegan Hines. "Counterfactual Explanations for Machine Learning: A Review". Oct. 20, 2020. arXiv: 2010.10596 [cs, stat]. URL: http://arxiv.org/abs/2010.10596 (visited on 10/05/2022) (pages 3, 13, 22).

[64] Matthew J. Vowels, Necati Cihan Camgoz, and Richard Bowden. "D'ya Like DAGs? A Survey on Structure Learning and Causal Discovery". In: *ACM Computing Surveys* 55.4 (May 31, 2023), pp. 1–36. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3527154. URL: https://dl.acm.org/doi/10.1145/3527154 (visited on 12/05/2022) (page 14).

[65] Sandra Wachter, Brent Mittelstadt, and Chris Russell. "Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR". In: *SSRN Electronic Journal* (2017). ISSN: 1556-5068. DOI: 10.2139/ssrn.3063289. URL: https://www.ssrn.com/abstract=3063289 (visited on 09/26/2022) (pages 4, 16, 18, 19).

[66] Yixin Wang and Michael Jordan. "Representation Learning as Finding Necessary and Sufficient Causes". In: ICML 2022: Workshop on Spurious Correlations, Invariance and Stability. July 30, 2022. URL: https://openreview.net/forum?id=_rqQZ_HL_fY (visited on 12/05/2022) (page 15).

[67] Jingkang Yang et al. "Generalized Out-of-Distribution Detection: A Survey". Aug. 3, 2022. DOI: 10.48550/arXiv.2110.11334. arXiv: 2110.11334 [cs]. URL: http://arxiv.org/abs/2110.11334 (visited on 10/26/2022) (page 5).

[68] Wei Zhang, Brian Barr, and John Paisley. "An Interpretable Deep Classifier for Counterfactual Generation". In: *Proceedings of the Third ACM International Conference on AI in Finance*. ICAIF '22. New York, NY, USA: Association for Computing Machinery, Oct. 26, 2022, pp. 36–43. ISBN: 978-1-4503-9376-8. DOI: 10.1145/3533271.3561722. URL: https://doi.org/10.1145/3533271.3561722 (visited on 11/21/2022) (pages 18, 19).

[69] Wei Zhang et al. "Shift-Invariant Pattern Recognition Neural Network and Its Optical Architecture". In: *Proceedings of Annual Conference of the Japan Society of Applied Physics*. Montreal, CA. 1988, pp. 2147–2151 (page 6).

[70] Yu Zhang et al. "A Survey on Neural Network Interpretability". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 5.5 (Oct. 2021), pp. 726–742. ISSN: 2471-285X. DOI: 10.1109/TETCI.2021.3100641. arXiv: 2012.14261 [cs]. URL: http://arxiv.org/abs/2012.14261 (visited on 07/25/2022) (page 1).