VCNet: A self-explaining model for realistic counterfactual generation

Victor Guyomard[1,2], Françoise Fessant[1], Thomas Guyet[3]
Tassadit Bouadi[2], Alexandre Termier[2]

1. Orange Labs, Lannion, France
victor.guyomard@orange.com
2. Univ Rennes, Inria, CNRS, IRISA, Rennes, France
3. Inria, Centre de Lyon, France

Counterfactual explanation is a common class of methods to make local explanations of machine learning decisions. For a given instance, these methods aim to find the smallest modification of feature values that changes the predicted decision made by a machine learning model. One of the challenges of counterfactual explanation is the efficient generation of realistic counterfactuals. To address this challenge, we propose VCNet – Variational Counter Net – a model architecture that combines a predictor and a counterfactual generator that are jointly trained, for regression or classification tasks. VCNet is able to both generate predictions, and to generate counterfactual explanations without having to solve another minimisation problem. Our contribution is the generation of counterfactuals that are close to the distribution of the predicted class. This is done by learning a variational autoencoder conditionally to the output of the predictor in a join-training fashion. We present an empirical evaluation on tabular datasets and across several interpretability metrics. The results are competitive with the state-of-the-art method.

1 Introduction

Improvements of machine learning techniques for decision systems has led to the rise of applications in various domains such as healthcare, credit or justice. The eventual sensitivity of such domains, as well as the black-box nature of the algorithms, has motivated the need for methods that explain why some prediction was made. For example, if a person's loan is rejected as a result of a model decision, the bank must be able to explain why. In such a context, it might be interesting to provide an explanation of what that person should change to influence the model's decision. As suggested by Wachter et al. [27], one way to build this type of explanation is through the use of counterfactual explanations. A counterfactual is defined as the smallest modification of feature values that changes the prediction of a model to a given output. In addition, the explanation also provides important feedback to the user. In the context of a denied credit, a counterfactual is a close individual for whom his credit is accepted and the feature modifications suggested by a counterfactual acts as recourse for the user. For privacy reason or simply because there is no similar individual with an opposite decision, we aim to generate synthetic individuals as counterfactuals.

In order to provide a meaningful recourse, the counterfactual is expected to be realistic, i.e. close to the existing examples and respecting the observed correlation among features. Furthermore, in order to be representative of its predicted class, it is interesting to obtain a counterfactual close to the existing examples but relative to the counterfactual class. A counterfactual instance is usually found by iteratively perturbing the input characteristics of the original example until the desired prediction is achieved, which is like minimizing a loss function using an optimization algorithm. Many methods proceed in this way, but differ in their definition of the loss function and optimization method [17]. These approaches appear to be computationally expensive. Indeed, for each instance to explain, a new optimisation problem has to be solved. Most of the counterfactual methods apply to already trained decision models and treat them as black boxes in the post-hoc paradigm. However, dissociating the prediction of the model from its explanation can lead to an explanation of poor quality [22].

Self-explaining models which incorporate an explanation generation module into their architecture, such that they provide explanations for their own predictions, can be an alternative to the previous approaches. In general, the predictor and explanation generator are trained jointly, hence the presence of the explanation generator is influencing the training of the predictor [7]. In this spirit, Guo et al. [9] propose CounterNet, a neural network based architecture for the prediction and counterfactual generation along with a novel variant of back propagation to ensure the stabilization of the training process. Compared to a post-hoc approach, they are able to produce counterfactuals with higher validity. A counterfactual is said to be valid if it succeeds in reaching a different prediction. A limitation of the CounterNet approach is that counterfactuals it generates may lack realism w.r.t. the data points of the class where they are positioned. The proposed approach, VCNet, tackles this issue: similarly to CounterNet, it combines a predictor and a counterfactual generator that are jointly trained. The difference lies in the counterfactual generator based on conditional variational autoencoder (cVAE) whose latent space can be controlled and tweaked to generate realistic counterfactuals. Our approach is inspired from John et al. work about learning disentangled latent spaces in the context of text style transfer [10].

Our main contribution is the proposal of a cVAE for counterfactual generation in order to generate realistic counterfactuals. Our second contribution is a self-explainable architecture of a classifier that embeds a cVAE, used as a counterfactual generator. In this architecture, both the model and the cVAE, are jointly trained with an efficient single back-propagation procedure. After recalling the properties of the variational autoencoder models (Section 3) that interest us in the context of this paper, we describe our proposal (Section 4). Then we present extensive experimental studies on synthetic and real data (Section 5). We compare the quality of the counterfactuals produced with those of CounterNet on different datasets through state-of-the-art metrics. The focus is on tabular data but we also show that our architecture is interesting on images.

2 Related Work

Our work is concerned with the search for counterfactual explanation that is usually found by iteratively perturbing the input features of the instance of interest until the desired prediction is reached. In practice, the search of counterfactuals is usually posed as an optimization problem. It consists of minimizing an objective function which encodes desirable properties of the counterfactual instance with respect to the perturbations. Wachter et al. [27] propose the generation of counterfactual instances by minimizing the distance between the instance to be explained and the counterfactual while pushing the new prediction toward the desired class. Other algorithms optimize other aspects with additional terms in the objective function such as actionability [25], diversity [18, 23], realism [26].

All aforementioned techniques search for counterfactual example by solving a separate optimization problem for each instance to be explained. This optimization problem is computationally intensive, making it impractical for large numbers of instances. To address this issue, several frameworks based on generative models have been proposed. A generative counterfactual model is trained to predict the counterfactual perturbations or instances directly. Many of these frameworks use the latent space of variational autoencoder models to generate counterfactuals with linear interpolation [2], latent feature selection [6], perturbation [21] or incorporation of the target class in the latent space [19]).

All these counterfactual generation techniques are post-hoc as they assume that the explaining task is done after the decision task with a fixed black-box model. In this post-hoc paradigm, the counterfactual search process is completely uninformed from the decision process. Post-hoc explanations may be the only option for already-trained models.

Another approach is to design models that optimize both the decision and an explanation of that decision during the learning process [1]. In the context of counterfactual explanations, such a strategy has been recently proposed by Guo et al. [9] with CounterNet, a framework in which prediction and explanation are jointly learned. The optimization of the counterfactual example generation only once together with the predictive model allows a better alignment between predictions and counterfactual explanations. This leads to explanations of better quality and significantly reduces the generation process time. Our architecture is inspired by theirs but we have chosen to use a conditional variational autoencoder as a generative model of counterfactuals allowing us to exploit text style-transfer techniques [10, 19].

3 Backgrounds

In this section, we introduce some notations, problematic and backgrounds necessary for the presentation of the VCNet architecture in the next section.

Let $\mathcal{X} \subseteq \mathbb{R}^p$ represents the p-dimensional feature space and $l \geq 2$ a number of classes. A training dataset, denoted $D = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$, is such that $\boldsymbol{x_i} \in \mathcal{X}$ and $y_i \in \{1, \ldots, l\}$ for each $i \in \{1, \ldots, n\}$. For a new example $\boldsymbol{x}$, the prediction consists in deciding to which class $\hat{y}$ the example $\boldsymbol{x}$ belongs. For more generality, we consider probabilistic prediction: the prediction is a probability vector, denoted $\hat{\boldsymbol{p}} \in [0, 1]^p$. Then, the predicted class is the most likely class according to $\hat{\boldsymbol{p}}$. The counterfactual generation yields an example $\boldsymbol{x'}$ which is close to $\boldsymbol{x}$ and whose prediction $\hat{y}'$ is different from $\hat{y}$ in case the counterfactual is valid.

Our problem is both to learn from D an accurate predictor, denoted f, and a generator of counterfactual, denoted g.

Now that we have presented our problem, we introduce the notion of VAE [12] and cVAE [24] which our proposal relies on.

3.1 Variational Autoencoder (VAE)

A variational autoencoder is a generative model where a latent parameterized distribution is learned. If samples are drawn in the latent space according to this distribution, the decoded samples are expected to be distributed according to the training data distribution (an approximate distribution of the training data distribution is learned) [12]. Formally, let $\boldsymbol{z}$ be a latent vector (drawn from the latent distribution) and $\boldsymbol{x}$ be an example, we denote by $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ the encoder distribution and by $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ the decoder distribution. Training a VAE is finding the parameters θ and ϕ that minimize the following objective function, i.e. the opposite of the Evidence Lower

Bound (ELBO):

$$\mathcal{L}_{\text{VAE}}(\theta, \phi) = -\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log(p_\theta\left(\boldsymbol{x} \mid \boldsymbol{z}\right))\right] + \mathrm{D}_{KL}\left[q_\phi\left(\boldsymbol{z} \mid \boldsymbol{x}\right) \middle\| p(\boldsymbol{z})\right] \tag{1}$$

Generally, distributions are chosen to be Gaussian, meaning that $q_\phi\left(\boldsymbol{z} \mid \boldsymbol{x}\right) \sim \mathcal{N}\left(\mu_\phi, \Sigma_\phi\right)$ and $p_\theta\left(\boldsymbol{x} \mid \boldsymbol{z}\right) \sim \mathcal{N}\left(\boldsymbol{x} \mid \mu_\theta, \Sigma_\theta\right)$ and distribution parameters are estimated thanks to back-propagation. The first term of Eq. 1 encourages reconstructing $\boldsymbol{x}$ at the output of the decoder ($\hat{\boldsymbol{x}}$). The second term encourages the regularization of the latent space by pushing $q_\phi\left(\boldsymbol{z} \mid \boldsymbol{x}\right)$ to a Gaussian prior $p \sim \mathcal{N}(0, I)$.

3.2 Conditional Variational Autoencoder (cVAE)

A conditional variational autoencoder is a VAE where distributions are conditioned on a given variable c [24]. The architecture is the same as a standard VAE but c is given as input of the encoder and also as input of the decoder. Then the objective function of Eq. 1 can be rewritten as:

$$\mathcal{L}_{\text{cVAE}}\left(\theta, \phi\right) = -\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x},c)}\left[\log(p_\theta\left(\boldsymbol{x} \mid \boldsymbol{z}, c\right))\right] + \mathrm{D}_{KL}\left[q_\phi\left(\boldsymbol{z} \mid \boldsymbol{x}, c\right) \middle\| p(\boldsymbol{z})\right] \tag{2}$$

The encoder distribution becomes $q_\phi\left(\boldsymbol{z} \mid \boldsymbol{x}, c\right)$ and is pushed to the Gaussian prior $p \sim \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{0}, \boldsymbol{I})$ by the regularization term regardless of the value of c. The decoder reconstructs $\boldsymbol{x}$ from the concatenation of $\boldsymbol{z}$ with c.

The initial objective of conditional variational autoencoder is to enrich the expressiveness of the model in supervised settings. In this article, we use a property of the cVAE to disentangle the class specification from the content of the data [13].[1] Intuitively, the latent variable $\boldsymbol{z}$ does not need to model the example category, then it can focus on modeling the content of the examples, which is shared by all the categories. To illustrate the effective disentanglement of category and content, Kingma et al. show that the decoder $p_\theta\left(\boldsymbol{x} \mid \boldsymbol{z}, c\right)$ can be used to generate images of the ten digits with the same shared content (let say the handwriting) by changing the class c and keeping the same random values for $\boldsymbol{z}$. The same property has been applied to text style transfer [10]. In this context, the style is the category, and the content is the wording. For tabular data, the notion of "content" and "style" can be illustrated in the context of the loan decision. The "style" characterizes the category of people loan (*accepted* or *rejected*) and the "content" characterizes the other features. More specifically, the later models correlations between variables that are independent from the loan decisions.

In our proposal, we exploit the modeling properties of a cVAE to generate counterfactuals. Considering that the cVAE disentangles the category and the content, the decoder of a cVAE can be tweaked in a flexible way. For $\boldsymbol{z}$ the encoding of an example $\boldsymbol{x}$ of class c, $p_\theta\left(\boldsymbol{x} \mid \boldsymbol{z}, c'\right)$ generates examples of a category $c' \neq c$. In addition, $(\boldsymbol{z}, c')$ is likely the element of class c that is the closest to $(\boldsymbol{z}, c)$ in the latent space. As this space is regularized, $p_\theta\left(\boldsymbol{x} \mid \boldsymbol{z}, c'\right)$ generates examples that are similar to $\boldsymbol{x}$, but belonging to a different class. This fits exactly the expectations of counterfactuals and will be assessed in Section 5.1.

[1] For Kingma et al [13], what we call the "content" in this paper is denoted the "style". It refers to the writing style of digits in MNIST-like datasets.

4 A Join Training Model

VCNet is a self-explainable [2] model through counterfactual generation. Inspired by CounterNet [9], the VCNet model is made of a predictor, $f(\cdot)$, and a counterfactual generator, $g(\cdot, \cdot)$, that are jointly trained. In the inference phase, each part can be used on demand: on the one side, to get the prediction $f(\boldsymbol{x})$ for some new example $\boldsymbol{x}$ and, on the other side, to generate its counterfactual $g(\boldsymbol{x}, c)$ for another class c. VCNet can be used as a self-explainable model and generates $(f(\boldsymbol{x}), g(\boldsymbol{x}, c))$, i.e. the couple of the prediction and its counterfactual.

The trick of VCNet is to not train a counterfactual generator, but a supervised autoencoder, i.e. a cVAE. The cVAE is trained as an autoencoder and used as a counterfactual generator in inference.

We start by presenting the architecture of our network, then we detail the training problem by defining the loss which is optimized and finally, we detail how the trained model is used to generate counterfactuals.

4.1 VCNet Architecture

VCNet is a neural network architecture. Figure 1 illustrates this architecture made of three main blocks:

1. Shared layers, s_β, that transform the input into a dense latent representation. We use fully connected layers with ELU activation functions.

2. A predictor network f_α that takes the shared latent representation and returns a probability vector corresponding to the probabilistic prediction. We use fully connected layers with ELU activation functions.

3. A conditional variational autoencoder that takes as input the shared latent representation and also the probability vector given by the predictor. The cVAE reconstructs examples and integrates additional layers to handle categorical variables (see details below).

During training, an example $\boldsymbol{x}_i$ is passed through the shared layers to generate a dense latent representation $\boldsymbol{h}_i = s_\beta(\boldsymbol{x}_i)$. This representation is then shared with both the predictor network and the autoencoder. On the one hand, $\boldsymbol{h}_i$ is passed through the predictor f_α in order to obtain the probability vector $\hat{\boldsymbol{p}}_i$. Then, the prediction of an example $\boldsymbol{x}_i$ is obtained by the function $f_{\alpha,\beta}(\boldsymbol{x}_i) = f_\alpha \circ s_\beta(\boldsymbol{x}_i)$. On the other hand, $\boldsymbol{h}_i$ is passed through the cVAE. It is first concatenated with $\hat{\boldsymbol{p}}_i$ and then is passed through the encoder of the cVAE and samples a latent vector $\boldsymbol{z}_i$. This latent vector concatenated with $\hat{\boldsymbol{p}}_i$ is passed through the decoder, so as to obtain a reconstructed example $\hat{\boldsymbol{x}}_i$.

It can be noticed that the cVAE slightly differs from the original cVAE [24]. Indeed, formally, the encoder of an end-to-end cVAE includes the shared layer, s_β. In our architecture, the condition is introduced at an intermediary latent representation of the examples. The idea behind this architecture is to enforce the dense latent representation to be as independent of the class as possible.

In addition, we adopt the same preprocessing as Guo et al. [9] to handle categorical variables. At the input of the network, each categorical variable is encoded with a one-hot. At the output of the cVAE, we add a softmax activation function for each one-hot categorical variable in order to obtain a one-hot encoding format by taking the *argmax*. Finally, continuous variables are scaled to have all variables in $[0, 1]$.

[2] By Self-explainable model here we mean that the predictor is constrained by the counterfactual generator during training but the explanation is not directly used to produce model output as in [1].

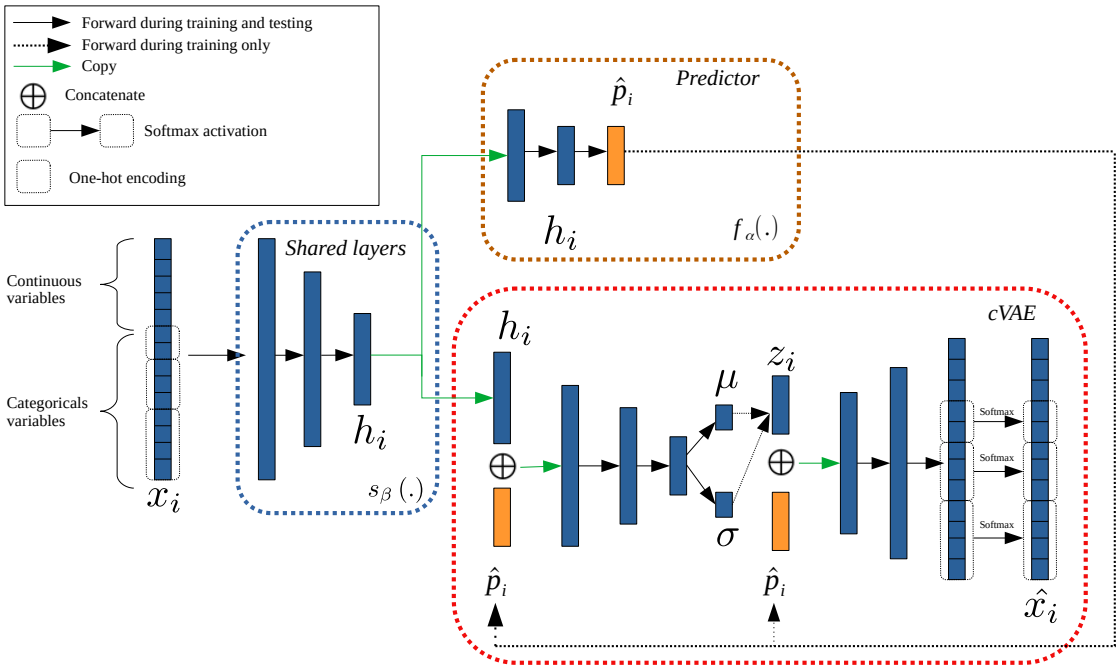

Figure 1: VCNet architecture is composed of three blocks: Shared layers that transform the input into a latent representation (blue square), a predictor that outputs the prediction (brown square), and a conditional variational autoencoder that acts as a counterfactual generator during testing (red square).

4.2 Loss Function and Training Procedure

The objective of the training is to jointly learn the predictor and the cVAE. Then, the loss to minimize is made of two terms.

The first term is derived from the loss of a cVAE defined in Eq. 2. In our case, the cVAE is conditioned by the probability vector obtained at the output of the predictor. Then we can rewrite the objective function as:

$$\mathcal{L}_{cVAE}(\theta, \phi, \beta; \boldsymbol{x}_i) = -\lambda_3 \mathbb{E}_{q_\phi(\boldsymbol{z}_i | s_\beta(\boldsymbol{x}_i), \hat{\boldsymbol{p}}_i)} \left[\log(p_\theta(\boldsymbol{x}_i \mid \boldsymbol{z}_i, \hat{\boldsymbol{p}}_i))\right] + \lambda_1 \mathrm{D}_{KL}\left[q_\phi(\boldsymbol{z}_i \mid s_\beta(\boldsymbol{x}_i), \hat{\boldsymbol{p}}_i) \,\middle\|\, p(\boldsymbol{z}_i)\right] \quad (3)$$

λ_1 and λ_3 are weights to control the impact of each term during the training phase.

The second term enables us to learn the predictor. We use cross-entropy as classification loss between the output of the predictor $\hat{\boldsymbol{p}}_i = f_\alpha \circ s_\beta(\boldsymbol{x}_i)$ and the true label y_i:

$$\mathcal{L}_{pred}(\alpha, \beta; \boldsymbol{x}_i, y_i) = \sum_{k=1}^{l} -\mathbb{1}_{[y_i=k]} \log\left([f_\alpha \circ s_\beta(\boldsymbol{x}_i)]_k\right) \quad (4)$$

where $[f_\alpha \circ s_\beta(\boldsymbol{x}_i)]_k$ denotes the predicted probability that $\boldsymbol{x}_i$ belongs to the k-th class.

Then, the loss function on the training set (D) is a weighted sum of the losses from Eq. 4 and

Eq. 3 as follows:

$$\mathcal{L}(\theta, \alpha, \beta, \phi; D) = \sum_{i=1}^{n} \mathcal{L}_{cVAE}(\theta, \phi, \beta; \boldsymbol{x}_i) + \lambda_2 \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{pred}(\alpha, \beta; \boldsymbol{x}_i, y_i)$$

As mentioned at the beginning of this section, it is worth noticing that our problem is not to learn to generate counterfactuals. Then, contrary to CounterNet that has divergent objectives to optimize, the minimization of $\mathcal{L}$ is a simple optimization problem solved by back-propagation.

Note that λ_1,λ_2,λ_3 are hyperparameters to tune for training.

4.3 Counterfactual Generation

Since our model does not directly produce counterfactuals, some modifications are needed for inference. An example $\boldsymbol{x}_i$ is passed through the prediction network to obtain both its predicted probability vector ($\hat{\boldsymbol{p}}_i$) and its dense vector representation ($\boldsymbol{h}_i$). This dense vector representation ($\boldsymbol{h}_i$) and the predicted probability vector ($\hat{\boldsymbol{p}}_i$) are given to the encoder of the cVAE to produce a latent vector $\boldsymbol{z}_i$. Then, the decoder of the cVAE plays the role of a counterfactual generator. Because we want to generate an example with a different predicted class we need a probability vector $\boldsymbol{p}_c$ such that the class with maximum probability is different from the one of the prediction, i.e. $\arg\max(\hat{\boldsymbol{p}}_i) \neq \arg\max(\boldsymbol{p}_c)$. In a binary-classification problem setup, we decided to use a one-hot vector where the probability is 0 for the predicted class of x_i and 1 for the opposite class. The reason for this choice is that we want to generate counterfactuals for which the confidence in the predicted class is the highest for the predictor. In the case of a multi-class classification problem, we propose to select the class with the second-highest probability in $\hat{\boldsymbol{p}}_i$ and to switch the values with the predicted class. For example, if we obtain a probability vector $\hat{\boldsymbol{p}}_i = [0.6, 0.3, 0.1]$ then $\boldsymbol{p}_c = [0.3, 0.6, 0.1]$. An alternative solution would be to let the user select the class for which he/she is interested in having a counterfactual.

This vector $\boldsymbol{p}_c$ and the dense latent representation $\boldsymbol{z}_i$ are passed through the cVAE decoder in order to infer a new predicted class to obtain a counterfactual [3] $\boldsymbol{x}_c$. As explained in Section 3.2, the intuition is to benefit from the disentanglement of the latent space of a cVAE: $\boldsymbol{z}_i$ contains non-class-specific information about $\boldsymbol{x}_i$ and $\boldsymbol{p}_c$ encodes information for the desired class. As such, the decoder generates a new example $\boldsymbol{x}_c$ that is similar to $\boldsymbol{x}_i$ and that belongs to a different class.

5 Experiments and Results

Our experiments are organized in four steps. Our main objective is to show that VCNet generates counterfactuals that are both valid (counterfactuals actually belong to another class) and realistic (counterfactuals are close to examples of the same class). In the first set of experiments, we present results of a cVAE on a synthetic dataset to confirm the actual disentanglement of the content and the class. These experiments also aim at providing some intuition about the reason why VCNet works. In the second set of experiments, we compare the results of VCNet with CounterNet, the state-of-the-art algorithm for self-explainable counterfactual generation. The reader interested in the results of more counterfactual generation systems may refer to the original article of Guo et al. [9]. Our experiments use the same datasets and data preprocessing. In the third experiment, we evaluate the impact of join training on the quality of counterfactuals. We propose a post-hoc version of our framework and compare the results obtained with the jointly trained VCNet. Finally, we present some qualitative experiments on the MNIST dataset. We choose to present experiments

[3] Note that the quality of the generated counterfactual depends on the quality of the learned latent space.

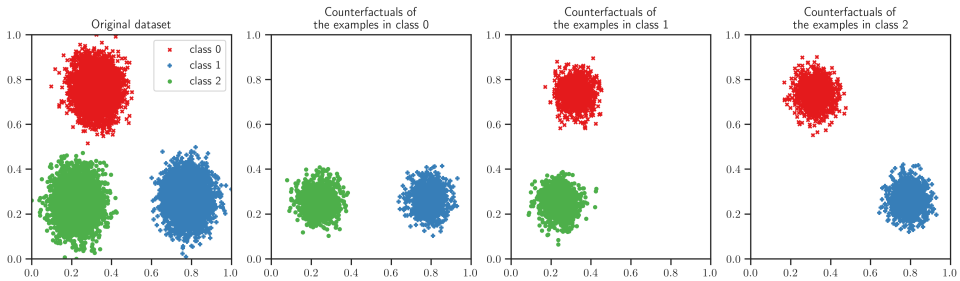

Figure 2: Comparison of the examples/counterfactuals distributions for synthetic data. All graphics represent the projection of the examples or counterfacturals on the first two dimensions of the examples space ($\mathbb{R}^8$). The graphic on the left represents the original dataset. The three other graphics represent counterfactuals generated from the examples of the test set belonging to each class. For this three rightmost graphics, the same examples have been used to generate counterfactuals with the two other classes. The color/shape of the point represents a class information: the class an example belongs to (graphic on the left) and the class requested for counterfactual generation (3 graphics on the right).

on MNIST firstly because it has been widely used in the field of counterfactual generation and, secondly, because it illustrates that VCNet may be applied on different types of data (tabular, images, time series, ...).

The hyperparameters and architectures of the models used in these experiments are detailed in Supplementary material. The code to reproduce the results of this section is also provided in supplementary material.

5.1 cVAE for counterfactual generation

Our proposal is based on using a cVAE to generate counterfactuals. It relies on the capability of a cVAE to actually disentangle the class and its content such that the decoder can be used to generate counterfactual examples by changing the class conditioning.In this experiment, we generate a synthetic dataset of examples in $\mathbb{R}^8$ with three classes. Each class is distributed according to a multidimensional Gaussian distribution (see Figure 2 on the left).

A cVAE, i.e. a couple of an encoder $\varphi(\boldsymbol{x}, c)$ and a decoder $\psi(\boldsymbol{z}, c)$, is trained on a set of $10k$ examples. The complete code of these experiments is available in supplementary material for the sake of reproducibility.

Figure 2 illustrates the capability of a cVAE to generate realistic examples when changing the class that conditions the decoder. The figure on the left illustrates the dataset. Each colored group of point corresponds to a class. The three figures on the right illustrate datasets that have been generated $\boldsymbol{x}' = \psi(\varphi(\boldsymbol{x}, c), c')$ where $\boldsymbol{x}$ is an example from the test that belongs to class c (origin class). Each figure corresponds to one origin class, the colors of the point correspond to the conditioning class (c'). We observe that all figures look similar. This means that taking $\psi(\varphi(\boldsymbol{x}, c), c')$ generates an example that looks similar to an example of class c' whatever the origin class of $\boldsymbol{x}$. Thus, it demonstrates that cVAE can be used to generate realistic counterfactuals. Moreover, $\boldsymbol{x}'$ is a good counterfactual if it is similar to $\boldsymbol{x}$. The question is whether $\boldsymbol{z} = \varphi(\boldsymbol{x}, c)$ is a better choice to generate an example $\psi(\boldsymbol{z}, c')$ than any other example $\psi(\boldsymbol{z}', c')$ (which also likely belongs to c'). To assess this behavior, we randomly generate 10 latent representations $\boldsymbol{z}' = \boldsymbol{z} + \boldsymbol{\delta}$ for each $\boldsymbol{x}$, and compute the Euclidean distance between $\boldsymbol{x}' = \psi(\boldsymbol{z}', c')$ and $\boldsymbol{x}$.

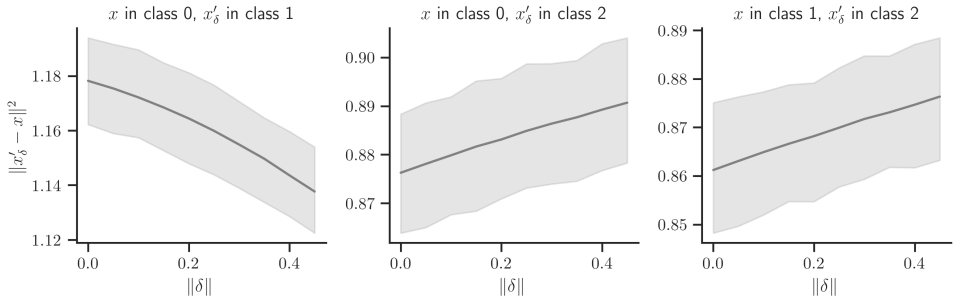

Figure 3: Distance between $\boldsymbol{x}$, an example of class c to explain, and $\boldsymbol{x'} = \psi(\boldsymbol{z} + \boldsymbol{\delta}, c')$, a counterfactual for class c' perturbed in latent space by $\boldsymbol{\delta}$. Each graphic illustrates this distance with respect to $\|\boldsymbol{\delta}\|$ depending on the class c and c' (on the right: $c = 0$ and $c' = 1$; in the middle: $c = 0$ and $c' = 2$; on the left: $c = 1$ and $c' = 2$). The mean and variance are computed on 10 random δ.

Figure 3 shows three graphs: one per couple of classes (the class of the example to explain and the class requested as counterfactual). Each graph illustrates the mean Euclidean distance between $\boldsymbol{x'}$ and $\boldsymbol{x}$ with respect to $\|\boldsymbol{\delta}\|$. When $\|\boldsymbol{\delta}\| = 0$, it uses the latent representation of $\boldsymbol{x}$ as input of the cVAE decoder. Intuitively, we expect to have $\psi(\varphi(\boldsymbol{x}, c), c')$ closer to $\boldsymbol{x}$ than $\psi(\varphi(\boldsymbol{x}, c) + \boldsymbol{\delta}, c')$ and thus, that the higher $\|\boldsymbol{\delta}\|$, the higher the mean distance to the original example. The two graphics on the right illustrates the expected behavior. In these cases, the latent representation of the example seems to generate a counterfactual that is the most similar among the examples of the opposite class. Nonetheless, we observe that it is not always the case. We can note that the distance decreases when perturbing the latent representation of examples in class 0 and regenerating counterfactual examples in class 1. This may be explained by the proximity between the two classes.

5.2 Comparison between VCNet and CounterNet

This section compares the quality of counterfactuals of VCNet against CounterNet. We conduct evaluations on six binary-classification datasets with various properties: Adult [14], HELOC [8], OULAD [15], Breast Cancer Wisconsin [3], Student performance [4] and Titanic [11]. Some of these datasets contain only numerical variables but some others, such as Adult, contain both numerical and categorical variables. More details about the datasets can be found in the supplementary material of the article.

To evaluate the counterfactual quality, we used the following four metrics that are classical in the literature.

Prediction gain: The prediction gain is given by the difference between the predicted probability of the counterfactual $\boldsymbol{x'}$ and the predicted probability of the example $\boldsymbol{x}$, according to the counterfactual class [20].

$$\text{Gain} = [f_{\text{pred}}(\boldsymbol{x'})]_{y_i} - [f_{\text{pred}}(\boldsymbol{x})]_{y_i}$$

where y_i denotes the predicted class for the counterfactual. A higher prediction gain means being more confident in the class change of the counterfactual.

Validity: A counterfactual is valid if it achieves to obtain a different predicted class [18, 5]. Then:

$$\text{Validity} = \begin{cases} 0, \text{ if } y_i = y_0 \\ 1, \text{ if } y_i \neq y_0 \end{cases}$$

where y_i denotes the predicted class for the counterfactual and y_0 the predicted class for the example to explain.

Proximity: The proximity is the L_1 distance between an example, $\boldsymbol{x}$ and its counterfactual, $\boldsymbol{x}'$ [18, 27].

$$\text{Proximity} = \|\boldsymbol{x}' - \boldsymbol{x}\|_1 = \|\boldsymbol{\delta}\|_1$$

A low value indicates fewer changes of features to apply to the original example to obtain the counterfactual.

Proximity score: This metric is inspired from Laugel et al. [16] to quantify the distance of a counterfactual to examples of the same predicted class:

$$P_s(\boldsymbol{x}') = \frac{d\left(\boldsymbol{x}', a_0\right)}{\frac{1}{\|H\|(\|H\|-1)/2} \sum_{\boldsymbol{x}_i, \boldsymbol{x}_j \in H} d\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)}$$

where $d\left(\boldsymbol{x}', a_0\right)$ is the Euclidean distance of the counterfactual to the closest example that has the same predicted class (a_0) and H is the set of existing examples that have the same predicted class as $\boldsymbol{x}'$. The insight behind this metric is that the counterfactual should be close to an existing example of the same predicted class relative to the rest of the data. Note that to be evaluated, this metric requires a set of m examples $X \in \mathbb{R}^{m \times p}$.

For each dataset, we choose a random sample of size 25% for counterfactual generation. Then, we compute the mean and standard deviation of each metric for every selected random sample.

Table 1 provides results of VCNet and CounterNet [9]. More information on the reproducibility of CounterNet results is available in Supplementary material. It is worth noting that Table 1 contains additional results for post-hoc VCNet that will be discussed in Section 5.3.

Counterfactual quality: VCNet achieves perfect validity for 4 of the 6 datasets, and a lower validity of respectively 4.5% and 7.5% for the 2 other datasets (Student and Titanic) compared to CounterNet.

As far as prediction gain and proximity score are concerned, VCNet outperforms CounterNet for all the 6 datasets. The higher the prediction gain, the more confidence one can have in the prediction related to the class change of the counterfactual. At the same time, a low proximity score reflects the achievement of counterfactuals that are close to real examples belonging to the same class as predicted for the counterfactual.

For the last evaluated metric that is proximity, we observe that VCNet achieves higher values than CounterNet on 5 of the 6 datasets. A larger proximity value means that the counterfactuals are obtained at the cost of larger changes in the input space.

Predictive accuracy: Both CounterNet and VCNet are self-explainable models, and if the previous results show that VCNet generates better counterfactuals, it can not be at the expense of the prediction accuracy. Thus, Table 1 also presents model accuracies. We observe that VCNet achieved similar performances on 3 of the 6 datasets. On the other hand, the accuracies for the other datasets are lower by 0.4% (HELOC) to 2% (Student), which shows that our method still performs very well in terms of prediction.

5.3 Impact of join-training on counterfactual quality

We derived our architecture to a post-hoc version (see Figure 4). Its training procedure is the following: 1) we first train a prediction model. For our comparisons here, it is composed of the concatenation of the shared layers block and the predictor block of VCNet, but in practice it can be any machine learning model that outputs a probability score. Once the model is trained, we obtain a probability vector $\hat{\boldsymbol{p}}_i$ by forwarding an example $\boldsymbol{x}_i$ to the model. 2) then we train a cVAE model conditionally to the probability vector ($\hat{\boldsymbol{p}}_i$) output by the predictor learned during Step 1. The cVAE is composed of the same shared layers block than VCNet, but it is not shared with the predictor.

Table 1: Comparison of quality metrics of counterfactuals and predictive accuracy for three methods : VCNet, CounterNet and Post-hoc VCNet. Bold items give the best metric values among the three methods.

Datasets	Metrics	Methods		
		VCNet	**CounterNet**	**Post-hoc VCNet**
Adult	**Validity**	**1.0**	0.99	0.84
	Proximity	7.71 ± 2.11	**7.16± 2.13**	7.28± 2.23
	Prediction gain	**0.76 ± 0.15**	0.61±0.17	0.47±0.35
	Proximity score	**0.04 ± 0.11**	0.31± 0.28	0.06±0.14
	Accuracy	**0.83**	**0.83**	**0.83**
OULAD	**Validity**	**1.0**	0.99	0.74
	Proximity	11.66±2.46	11.96±2.40	**11.22±2.54**
	Prediction gain	**0.93±0.12**	0.74±0.13	0.66±0.44
	Proximity score	**0.38±0.18**	0.46±0.16	**0.38±0.18**
	Accuracy	**0.93**	**0.93**	**0.93**
HELOC	**Validity**	**1.0**	0.99	0.77
	Proximity	5.60±2.11	**4.41±1.80**	5.09±1.71
	Prediction gain	**0.64±0.13**	0.56±0.15	0.24±0.25
	Proximity score	**0.23±0.21**	0.49±0.35	0.40±0.32
	Accuracy	0.71	**0.72**	0.71
Student	**Validity**	0.96	**1.0**	0.46
	Proximity	19.90±3.21	19.86±2.78	**19.68 ±3.03**
	Prediction gain	**0.86±0.27**	0.76±0.05	0.41±0.46
	Proximity score	**0.70±0.08**	0.73±0.06	0.75±0.08
	Accuracy	0.90	**0.92**	0.90
Titanic	**Validity**	0.92	**0.99**	0.38
	Proximity	15.43±3.79	**15.15±4.05**	15.56±5.23
	Prediction gain	**0.69±0.31**	0.66±0.15	0.26±0.36
	Proximity score	**0.71±0.21**	0.80± 0.16	1.21±0.26
	Accuracy	0.82	**0.83**	0.82
Breast-cancer	**Validity**	**1.0**	**1.0**	0.59
	Proximity	5.27±1.47	**1.51±1.01**	7.71±1.67
	Prediction gain	**0.95 ± 0.11**	0.69±0.15	0.60±0.45
	Proximity score	**0.28±0.03**	0.72±0.48	0.94± 0.07
	Accuracy	**0.96**	**0.96**	**0.96**

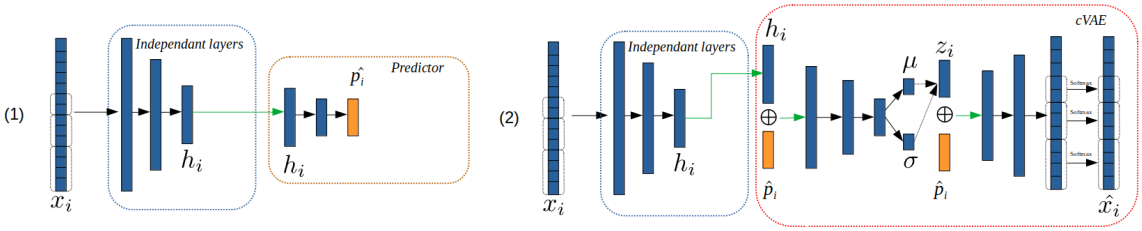

Figure 4: Post-hoc version of VCNet. (1) is the prediction model and (2) is the cVAE model.

We compare VCNet with its post-hoc version (Post-hoc VCNet) in order to study the impact of join-training on counterfactuals. Table 1 provides the results of the post-hoc version compared to those obtained previously with the join-training approach. We observe a drop of validity for every dataset, which justifies that the join-training approach allows a better alignment of the explanation task with the prediction task. We also observe a significant decrease in prediction gain for all datasets, which means less changes between an example to explain and its counterfactual. Besides, proximity is higher for 3 datasets (Adult, OULAD, Breast-cancer) and lower by respectively 9%, 1% and 0.8% on the remaining datasets (HELOC, Student, Titanic). Thus, we can argue that this drop of prediction gain does not benefit to closer counterfactuals w.r.t. examples to explained. In terms of model accuracies, training the prediction model alone leads to comparable results, which

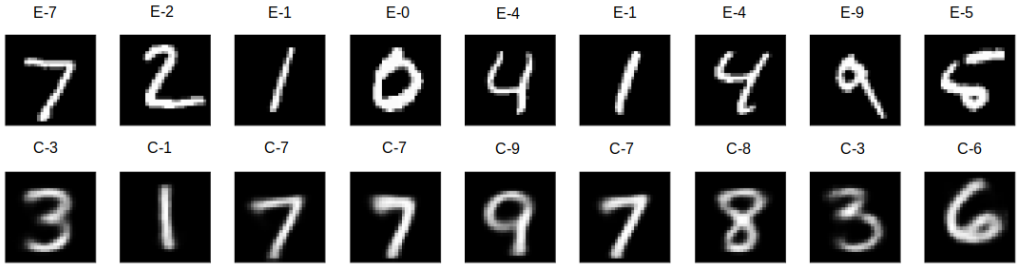

Figure 5: Counterfactuals obtained with VCNet for the MNIST dataset. The top line corresponds to the examples to explain, the bottom to the corresponding counterfactuals.

indicates that the join-training approach is not at the cost of a lower accuracy.

5.4 Qualitative results on MNIST dataset

We evaluated VCNet on the MNIST dataset[4] with metrics suggested in Section 5.2. This experiment illustrates that VCNet is adaptable to several types of data including image datasets. As CounterNet was not applied to image data in the original paper, we do not offer a comparison with this model here. This avoids a poor adaptation of CounterNet and an unfair comparison.

VCNet gives a mean validity of 0.99, a mean prediction gain of 0.98 and an accuracy of 0.98. The counterfactual quality metrics on MNIST show that VCNet is a good model to generate realistic and valid counterfactuals and, at the same time, to make accurate predictions. These results suggest that VCNet has also good capabilities to generate counterfactuals for images, and not only for tabular data. Figure 5 presents some examples to explain (in the first line) and corresponding counterfactuals generated with VCNet (second line). Each example to explain and its counterfactual is complemented by the predicted class, for instance **E-7** means an example predicted class 7 and **C-3** means a counterfactual predicted class 3. We first notice that each counterfactual "looks like" an example that matches the predicted class. Moreover, we observe that the orientation of the digits is often preserved, for example **E-1** is converted in **C-7** by keeping the orientation of **E-1**. **C-6** is interesting as it shows that VCNet is able to provide a realistic counterfactual even if the class of the example to explain is not trivial.

6 Conclusion

In this article, we propose VCNet, a new architecture for self-explainable classification based on counterfactuals examples. Our architecture generates at the same time the decision and a counterfactual that can be used by the analyst to understand the algorithm decision. The first advantage of VCNet is to generate explanations and decisions in a simple feed forward pass of the examples. Contrary to post-hoc counterfactuals explanation, VCNet does not require expensive optimization to generate counterfactuals.

The main contribution of this article is the use of a cVAE as a counterfactual generator in our model. The choice of a cVAE yields realistic counterfactuals and it is simple to train jointly with the prediction model.

We extensively evaluated the quality of the counterfactuals and compared them to CounterNet. We conclude that VCNet generates valid and realistic counterfactuals. The VCNet counterfactuals

[4] http://yann.lecun.com/exdb/mnist/

are realistic in the sense that they are close to existing examples of the same predicted class (VCNet has better proximity scores than CounterNet) and also the result of a higher confidence in the class change (VCNet has better prediction gains than CounterNet).

Finally, VCNet is simple to train because the training procedure is not based on counterfactuals directly, but on the disentanglement of the class and the content of examples by the cVAE. It allows proposing a simple optimisation procedure which makes our approach easier to put in practice. This is illustrated by its successful application to a dataset of images. In addition, it is also assessed in terms of model accuracy. Our experiments show that our join-training approach keeps its competitive prediction performance against CounterNet. A future work is to compare VCNet performances against state-of-the-art post-hoc counterfactuals methods and also to include actionability constraints.

References

[1] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 7786–7795, 2018.

[2] Brian Barr, Matthew R Harrington, Samuel Sharpe, and C Bayan Bruss. Counterfactual explanations via latent space projection and interpolation. preprint arXiv:2112.00890, 2021.

[3] Catherine Blake. UCI repository of machine learning databases. http://www.ics.uci.edu/mlearn/MLRepository.html, 1998.

[4] Paulo Cortez and Alice Maria Gonçalves Silva. Using data mining to predict secondary school student performance. In *Proceedings of Annual Future Business Technology Conference*, pages 5–12, 2008.

[5] Raphael Mazzine Barbosa de Oliveira and David Martens. A framework and benchmarking study for counterfactual generating methods on tabular data. *Applied Sciences*, 11(16):7274, 2021.

[6] Michael Downs, Jonathan L Chu, Yaniv Yacoby, Finale Doshi-Velez, and Weiwei Pan. Cruds: Counterfactual recourse using disentangled subspaces. In *ICML Workshop on Human Interpretability in Machine Learning (WHI)*, pages 1–23, 2020.

[7] Daniel C. Elton. Self-explaining AI as an alternative to interpretable AI. In *Proceedings of the International Conference on Artificial General Intelligence (AGI)*, pages 95–106, 2020.

[8] FICO. Explainable machine learning challenge. https://community.fico.com/s/explainable-machine-learning-challenge, 2018.

[9] Hangzhi Guo, Thanh Nguyen, and Amulya Yadav. CounterNet: End-to-end training of counterfactual aware predictions. In *ICML Workshop on Algorithmic Recourse*, 2021.

[10] Vineet John, Lili Mou, Hareesh Bahuleyan, and Olga Vechtomova. Disentangled representation learning for non-parallel text style transfer. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 424–434, 2019.

[11] Kaggle. Titanic – machine learning from disaster. https://www.kaggle.com/c/titanic/overview, 2018.

[12] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[13] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Proceedings of International Conference on neural information processing systems (NIPS)*, pages 3581–3589, 2014.

[14] R Kohavi and B Becker. UCI machine learning repository: Adult data set, 1996.

[15] Jakub Kuzilek, Martin Hlosta, and Zdenek Zdrahal. Open university learning analytics dataset. *Scientific data*, 4:170171, 2017.

[16] Thibault Laugel, Marie-Jeanne Lesot, Christophe Marsala, Xavier Renard, and Marcin Detyniecki. The dangers of post-hoc interpretability: Unjustified counterfactual explanations. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2801–2807, 2019.

[17] Christoph Molnar. *Interpretable Machine Learning*. C. Molnar, 2nd edition, 2022.

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

[19] Sharmila Reddy Nangi, Niyati Chhaya, Sopan Khosla, Nikhil Kaushik, and Harshit Nyati. Counterfactuals to control latent disentangled text representations for style transfer. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 40–48, 2021.

[20] Daniel Nemirovsky, Nicolas Thiebaut, Ye Xu, and Abhishek Gupta. CounteRGAN: Generating realistic counterfactuals with residual generative adversarial nets. preprint arXiv:2009.05199, 2020.

[21] Martin Pawelczyk, Klaus Broelemann, and Gjergji Kasneci. Learning model-agnostic counterfactual explanations for tabular data. In *Proceedings of The Web Conference (WWW'20)*, pages 3126–3132, 2020.

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

[23] Chris Russell. Efficient search for diverse coherent explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT)*, New York, NY, USA, 2019. Association for Computing Machinery.

[24] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 3483–3491, 2015.

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

[26] Arnaud Van Looveren and Janis Klaise. Interpretable counterfactual explanations guided by prototypes. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, pages 650–665, 2021.

Table 2: Datasets details

Dataset	Size	Continuous	Categorical
Adult	32,561	2	6
Student	649	2	14
Titanic	891	2	24
HELOC	10,459	21	2
OULAD	32,593	23	8
Breast Cancer	569	30	0

[27] Sandra Wachter, Brent Daniel Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law and Technology*, 31(2):841–887, 2018.

A Dataset Details

A.1 Tabular Data

We used 6 binary-classification datasets for the comparison of VCNet with CounterNet. Table 2 describe the size of each dataset as well as the number of categorical and continuous variables.

A.2 MNIST Dataset

MNIST is composed of 60000 examples in train and 10000 examples in test. We used all of the test set for counterfactual generation.

B Implementation Details

B.1 VCNet Details

We choose a binary cross entropy loss as the reconstruction error for the cVAE part and a cross entropy loss for the predictor part. We apply a grid search to tune hyperparameters that are specific for each datasets, these values are report in Table 3. We used an Adam optimizer for every dataset. Table 4 describe the model architecture for every dataset. It contains the dimensions for each block of VCNet. For example, with the adult dataset, the shared encoding transforms an example $\boldsymbol{x}_i$ of size **29** into a vector $\boldsymbol{h}_i$ of size **15**. Then the encoder of the cVAE part takes a concatenation of $\boldsymbol{h}_i$ and the prediction vector $\hat{\boldsymbol{p}}_i$ to produce a vector of size **16**. This vector is transform by the encoder to a lower representation of size **8** and finally a vector of size **5**.

B.2 Post-hoc VCNet For Tabular Data

For the post-hoc version of VCNet, the architecture is the same as detailed in Table 4. Nonetheless, hyperparameters for training change. Table 5 and 6 gives training hyperparameters for the cVAE model and prediction model respectively.

B.3 Additional Information on CounterNet Reproducibility

We used the CounterNet code that was available at this link https://github.com/bkghz-orange-blue/CounterNet. For fair comparison, we have computed counterfactuals from available trained models and used the same data processing for VCNet.

Table 3: Hyperparameters details for VCNet

Dataset	λ_1	λ_2	λ_3	Learning Rate	Epochs	Batch-size
Adult	1	1	0.001	0.001	250	128
Student	1	0.1	0.01	0.001	50	30
Titanic	1	1	0.001	0.001	100	30
HELOC	0.1	1	0.01	0.001	200	64
OULAD	1	1	0.01	0.001	40	128
Breast Cancer	1	0.1	0.001	0.001	100	30
MNIST	1	8	0.2	0.001	100	30

Table 4: Architecture details for VCNet

Dataset	Shared Encoding Dims	Encoding Dims	Predictor Dims	CF Generator Dims
Adult	[29,15]	[16,8,5]	[6,8,15,29]	[15,15,1]
Student	[85,50]	[51,20,10]	[11,20,50,85]	[50,50,1]
Titanic	[57,20]	[21,10,5]	[6,10,20,57]	[20,20,1]
HELOC	[35,15]	[16,8,5]	[6,8,15,35]	[15,15,1]
OULAD	[127,200]	[201,100,10]	[11,100,200,127]	[200,200,1]
Breast Cancer	[30,15]	[16,8,5]	[6,8,15,30]	[15,15,1]
MNIST	[784,400,40]	[50,400,20]	[30,400,784]	[40,10]

Table 5: Hyperparameters details for post-hoc VCNet on tabular data for the cVAE part

Dataset	λ_2	λ_3	Learning Rate	Epochs	Batch-size
Adult	1	0.001	0.001	10	128
Student	1	0.01	0.001	50	30
Titanic	1	0.001	0.001	20	30
HELOC	0.1	0.01	0.001	100	64
OULAD	0.1	0.01	0.001	40	128
Breast Cancer	1	0.001	0.001	30	30

Table 6: Hyperparameters details for post-hoc VCNet on tabular data for the prediction part

Dataset	Learning Rate	Epochs	Batch-size
Adult	0.001	50	128
Student	0.001	50	30
Titanic	0.001	80	30
HELOC	0.001	100	64
OULAD	0.001	150	128
Breast Cancer	0.001	10	30