Interpretable Counterfactual Explanations Guided by Prototypes

Arnaud Van Looveren(✉)[0000−0002−8347−5305] and Janis Klaise[0000−0002−7774−8047]

Seldon Technologies, 41 Luke St, London EC2A 4AR, UK
{avl,jk}@seldon.io

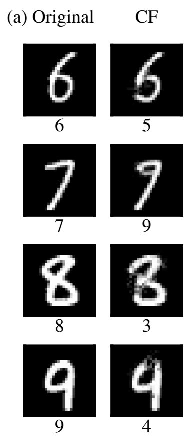
Abstract. We propose a fast, model agnostic method for finding interpretable counterfactual explanations of classifier predictions by using class prototypes. We show that class prototypes, obtained using either an encoder or through class specific k-d trees, significantly speed up the search for counterfactual instances and result in more interpretable explanations. We quantitatively evaluate interpretability of the generated counterfactuals to illustrate the effectiveness of our method on an image and tabular dataset, respectively MNIST and Breast Cancer Wisconsin (Diagnostic). Additionally, we propose a principled approach to handle categorical variables and illustrate our method on the Adult (Census) dataset. Our method also eliminates the computational bottleneck that arises because of numerical gradient evaluation for *black box* models.

Keywords: Interpretation · Transparency / Explainability · Counterfactual explanations

1 Introduction

Humans often think about how they can alter the outcome of a situation. *What do I need to change for the bank to approve my loan?* or *Which symptoms would lead to a different medical diagnosis?* are common examples. This form of counterfactual reasoning comes natural to us and explains how to arrive at a desired outcome in an interpretable manner. Moreover, examples of counterfactual instances resulting in a different outcome can give powerful insights of what is important to the underlying decision process, making it a compelling method to explain predictions of machine learning models (Figure 1).

In the context of predictive models, given a test instance and the model's prediction, a counterfactual instance describes the necessary change in input features that alter the prediction to a predefined output [21]. For classification models the predefined output can be any target class or prediction probability distribution. Counterfactual instances can then be found by iteratively perturbing the input features of the test instance until the desired prediction is reached. In practice, the counterfactual search is posed as an optimization problem—we want to minimize an objective function which encodes desirable properties of the counterfactual instance with respect to the perturbations. The key insight of this



(b)	Original	CF
Workclass	Private	State-gov
Education	High school	Bachelors
Marital Status	Married	Married
Occupation	Blue-Collar	Blue-Collar
Relationship	Husband	Husband
Race	White	White
Sex	Male	Male
Country	United-States	United-States
Age	46	46
Capital Gain	0	0
Capital Loss	0	0
Hours p/w	40	40
Prediction	$\leq \$50k/y$	$> \$50k/y$

Fig. 1. (a) Examples of original and counterfactual instances on the MNIST dataset along with predictions of a CNN model. (b) A counterfactual instance on the Adult (Census) dataset highlighting the feature changes required to alter the prediction of an NN model.

formulation is the need to design an objective function that allows us to generate high quality counterfactual instances. A counterfactual instance x_{cf} should have the following desirable properties:

1. The model prediction on x_{cf} needs to be close to the predefined output.
2. The perturbation δ changing the original instance x_0 into $x_{\text{cf}} = x_0 + \delta$ should be sparse.
3. The counterfactual x_{cf} needs to be interpretable. We consider an instance x_{cf} interpretable if it lies close to the model's training data distribution. This definition does not only apply to the overall data set, but importantly also to the *training instances that belong to the counterfactual class*. Let us illustrate this with an intuitive example. Assume we are predicting house prices with features including the square footage and the number of bedrooms. Our house is valued below £500,000 and we would like to know what needs to change about the house in order to increase the valuation above £500,000. By simply increasing the number of bedrooms and leaving the other features unchanged, the model predicts that our *counterfactual house* is now worth more than £500,000. This sparse counterfactual instance lies fairly close to the overall training distribution since only one feature value was changed. The counterfactual is however out-of-distribution with regards to the subset of houses in the training data valued above £500,000 because other relevant features like the square footage still resemble a typical house valued below £500,000. As a result, we do not consider this counterfactual to be very interpretable. We show in the experiments that there is often a trade-off between sparsity and interpretability.

4. The counterfactual instance x_{cf} needs to be found fast enough to ensure it can be used in a real life setting.

An overly simplistic objective function may return instances which satisfy properties 1. and 2., but where the perturbations are not interpretable with respect to the counterfactual class.

In this paper we propose using class prototypes in the objective function to guide the perturbations quickly towards an interpretable counterfactual. The prototypes also allow us to remove computational bottlenecks from the optimization process which occur due to numerical gradient calculation for black box models. In addition, we propose two novel metrics to quantify interpretability which provide a principled benchmark for evaluating interpretability at the instance level. We show empirically that prototypes improve the quality of counterfactual instances on both image (MNIST) and tabular (Wisconsin Breast Cancer) datasets. Finally, we propose using pairwise distance measures between categories of categorical variables to define meaningful perturbations for such variables and illustrate the effectiveness of the method on the Adult (Census) dataset.

2 Related Work

Counterfactual instances—synthetic instances of data engineered from real instances to change the prediction of a machine learning model—have been suggested as a way of explaining individual predictions of a model as an alternative to feature attribution methods such as LIME [23] or SHAP [19].

Wacther et al. [27] generate counterfactuals by minimizing an objective function which sums the squared difference between the predictions on the perturbed instance and the desired outcome, and a scaled L_1 norm of the perturbations. Laugel et al. [15] find counterfactuals through a heuristic search procedure by growing spheres around the instance to be explained. The above methods do not take local, class specific interpretability into account. Furthermore, for black box models the number of prediction calls during the search process grows proportionally to either the dimensionality of the feature space [27] or the number of sampled observations [15, 9], which can result in a computational bottleneck. Dhurandhar et al. [7, 9] propose the framework of *Contrastive Explanations* which find the minimal number of features that need to be changed/unchanged to keep/change a prediction.

A key contribution of this paper is the use of prototypes to guide the counterfactual search process. Kim et al. [14], Gurumoorthy et al. [11] use prototypes as example-based explanations to improve the interpretability of complex datasets. Besides improving interpretability, prototypes have a broad range of applications like clustering [13], classification [4, 26], and few-shot learning [25]. If we have access to an encoder [24], we follow the approach of [25] who define a class prototype as the mean encoding of the instances which belong to that class. In the absence of an encoder, we find prototypes through class specific k-d trees [3].

To judge the quality of the counterfactuals we introduce two novel metrics which focus on local interpretability with respect to the training data distribution.

This is different from [8] who define an interpretability metric relative to a target model. Kim et al. [14] on the other hand quantify interpretability through a human pilot study measuring the accuracy and efficiency of the humans on a predictive task. Luss et al. [20] also highlight the importance of good local data representations in order to generate high quality explanations.

Another contribution of this paper is a principled approach to handling categorical variables during the counterfactual generation process. Some previously proposed solutions are either computationally expensive [27] or do not take relationships between categories into account [9, 22]. We propose using pairwise distance measures to define embeddings of categorical variables into numerical space which allows us to define meaningful perturbations when generating counterfactuals.

3 Methodology

3.1 Background

The following section outlines how the prototype loss term is constructed and why it improves the convergence speed and interpretability. Finding a counterfactual instance $x_{\text{cf}} = x_0 + \delta$, with both x_{cf} and $x_0 \in \mathcal{X} \subseteq \mathbb{R}^D$ where $\mathcal{X}$ represents the D-dimensional feature space, implies optimizing an objective function of the following form:

$$\min_{\delta} c \cdot f_{\kappa}(x_0, \delta) + f_{\text{dist}}(\delta). \tag{1}$$

$f_{\kappa}(x_0, \delta)$ encourages the predicted class i of the perturbed instance x_{cf} to be different than the predicted class t_0 of the original instance x_0. Similar to [7], we define this loss term as:

$$\begin{aligned} L_{\text{pred}} &:= f_{\kappa}(x_0, \delta) \\ &= \max([f_{\text{pred}}(x_0 + \delta)]_{t_0} - \max_{i \neq t_0}[f_{\text{pred}}(x_0 + \delta)]_i, -\kappa), \end{aligned} \tag{2}$$

where $[f_{\text{pred}}(x_0 + \delta)]_i$ is the i-th class prediction probability, and $\kappa \geq 0$ caps the divergence between $[f_{\text{pred}}(x_0 + \delta)]_{t_0}$ and $[f_{\text{pred}}(x_0 + \delta)]_i$. The term $f_{\text{dist}}(\delta)$ minimizes the distance between x_0 and x_{cf} with the aim to generate sparse counterfactuals. We use an elastic net regularizer [28]:

$$f_{\text{dist}}(\delta) = \beta \cdot \|\delta\|_1 + \|\delta\|_2^2 = \beta \cdot L_1 + L_2. \tag{3}$$

While the objective function (1) is able to generate counterfactual instances, it does not address a number of issues:

1. x_{cf} does not necessarily respect the training data manifold, resulting in out-of-distribution counterfactual instances. Often a trade off needs to be made between sparsity and interpretability of x_{cf}.

2. The scaling parameter c of $f_\kappa(x_0, \delta)$ needs to be set within the appropriate range before a potential counterfactual instance is found. Finding a good range can be time consuming.

[7] aim to address the first issue by adding in an additional loss term L_{AE} which represents the L_2 reconstruction error of x_{cf} evaluated by an autoencoder AE which is fit on the training set:

$$L_{\mathrm{AE}} = \gamma \cdot \|x_0 + \delta - \mathrm{AE}(x_0 + \delta)\|_2^2. \tag{4}$$

The loss L to be minimized now becomes:

$$L = c \cdot L_{\mathrm{pred}} + \beta \cdot L_1 + L_2 + L_{\mathrm{AE}}. \tag{5}$$

The autoencoder loss term L_{AE} penalizes out-of-distribution counterfactual instances, but does not take the data distribution for each prediction class i into account. This can lead to sparse but uninterpretable counterfactuals, as illustrated by Figure 2. The first row of Figure 2(b) shows a sparse counterfactual 3 generated from the original 5 using loss function (5). Both visual inspection and reconstruction of the counterfactual instance using AE in Figure 2(e) make clear however that the counterfactual lies closer to the distribution of a 5 and is not interpretable as a 3. The second row adds a prototype loss term to the objective function, leading to a less sparse but more interpretable counterfactual 6.

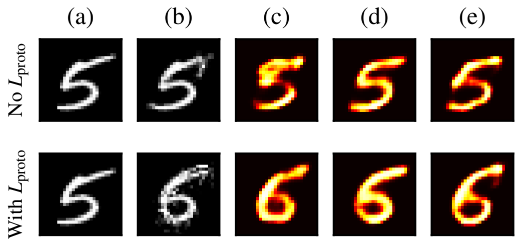


Fig. 2. First row: (a) original instance and (b) uninterpretable counterfactual 3. (c), (d) and (e) are reconstructions of (b) with respectively AE_3, AE_5 and AE. Second row: (a) original instance and (b) interpretable counterfactual 6. (c), (d) and (e) are reconstructions of (b) with respectively AE_6, AE_5 and AE.

The L_{AE} loss term also does not consistently speed up the counterfactual search process since it imposes a penalty on the distance between the proposed x_{cf} and its reconstruction by the autoencoder without explicitly guiding x_{cf} towards an interpretable solution. We address these issues by introducing an additional loss term, L_{proto}.

3.2 Prototype loss term

By adding in a prototype loss term L_{proto}, we obtain the following objective function:

$$L = c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 + L_{\text{AE}} + L_{\text{proto}}, \tag{6}$$

where L_{AE} becomes optional. The aim of L_{proto} is twofold:

1. Guide the perturbations δ towards an interpretable counterfactual x_{cf} which falls in the distribution of counterfactual class i.
2. Speed up the counterfactual search process without too much hyperparameter tuning.

To define the prototype for each class, we can reuse the encoder part of the autoencoder from L_{AE}. The encoder $\text{ENC}(x)$ projects $x \in \mathcal{X}$ onto an E-dimensional latent space $\mathbb{R}^E$. We also need a representative, unlabeled sample of the training dataset. First the predictive model is called to label the dataset with the classes predicted by the model. Then for each class i we encode the instances belonging to that class and order them by increasing L_2 distance to $\text{ENC}(x_0)$. Similar to [25], the class prototype is defined as the average encoding over the K nearest instances in the latent space with the same class label:

$$\text{proto}_i := \frac{1}{K} \sum_{k=1}^{K} \text{ENC}(x_k^i) \tag{7}$$

for the ordered $\{x_k^i\}_{k=1}^K$ in class i. It is important to note that the prototype is defined in the latent space, not the original feature space.

The Euclidean distance is part of a class of distance functions called *Bregman divergences*. If we consider that the encoded instances belonging to class i define a cluster for i, then proto_i equals the cluster mean. For Bregman divergences the cluster mean yields the minimal distance to the points in the cluster [1]. Since we use the Euclidean distance to find the closest class to x_0, proto_i is a suitable class representation in the latent space. When generating a counterfactual instance for x_0, we first find the nearest prototype proto_j of class $j \neq t_0$ to the encoding of x_0:

$$j = \underset{i \neq t_0}{\operatorname{argmin}} \|\text{ENC}(x_0) - \text{proto}_i\|_2. \tag{8}$$

The prototype loss L_{proto} can now be defined as:

$$L_{\text{proto}} = \theta \cdot \|\text{ENC}(x_0 + \delta) - \text{proto}_j\|_2^2, \tag{9}$$

where $\text{ENC}(x_0 + \delta)$ is the encoding of the perturbed instance. As a result, L_{proto} explicitly guides the perturbations towards the nearest prototype $\text{proto}_{j \neq t_0}$, speeding up the counterfactual search process towards the average encoding of class j. This leads to more interpretable counterfactuals as illustrated by the experiments. Algorithm 1 summarizes this approach.

Algorithm 1 Counterfactual search with encoded prototypes

```
1: Parameters: $\beta, \theta$ (required) and $c, \kappa$ and $\gamma$ (optional)
2: Inputs: AE (optional) and ENC models. A sample $X = \{x_1, \dots, x_n\}$ from training
   set. Instance to explain $x_0$.
3: Label $X$ and $x_0$ using the prediction function $f_{\text{pred}}$:
   $X^i \leftarrow \{x \in X \mid \text{argmax}\, f_{\text{pred}}(x) = i\}$ for each class $i$ $t_0 \leftarrow \text{argmax}\, f_{\text{pred}}(x_0)$
4: Define prototypes for each class $i$:
   $\text{proto}_i \leftarrow \frac{1}{K}\sum_{k=1}^{K} \text{ENC}(x_k^i)$ for $x_k^i \in X^i$ where $x_k^i$ is ordered by increasing
   $\|\text{ENC}(x_0) - \text{ENC}(x_k^i)\|_2$ and $K \leq |X^i|$
5: Find nearest prototype $j$ to instance $x_0$ but different from original class $t_0$:
   $j \leftarrow \text{argmin}_{i \neq t_0} \|\text{ENC}(x_0) - \text{proto}_i\|_2$.
6: Optimize the objective function:
   $\delta^* \leftarrow \text{argmin}_{\delta \in \mathcal{X}}\, c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 + L_{\text{AE}} + L_{\text{proto}}$ where $L_{\text{proto}} = \theta \cdot \|\text{ENC}(x_0 +$
   $\delta) - \text{proto}_j\|_2^2$.
7: Return $x_{\text{cf}} = x_0 + \delta^*$
```

Algorithm 2 Counterfactual search with k-d trees

```
1: Parameters: $\beta, \theta, k$ (required) and $c, \kappa$ (optional)
2: Input: A sample $X = \{x_1, \dots, x_n\}$ from training set. Instance to explain $x_0$.
3: Label $X$ and $x_0$ using the prediction function $f_{\text{pred}}$:
   $X^i \leftarrow \{x \in X \mid \text{argmax}\, f_{\text{pred}}(x) = i\}$ for each class $i$ $t_0 \leftarrow \text{argmax}\, f_{\text{pred}}(x_0)$
4: Build separate k-d trees for each class $i$ using $X_i$
5: Find nearest prototype $j$ to instance $x_0$ but different from original class $t_0$:
   $j \leftarrow \text{argmin}_{i \neq t_0} \|x_0 - x_{i,k}\|_2$ where $x_{i,k}$ is the $k$-th nearest item to $x_0$ in the k-d
   tree of class $i$.
   $\text{proto}_j \leftarrow x_{j,k}$
6: Optimize the objective function:
   $\delta^* \leftarrow \text{argmin}_{\delta \in \mathcal{X}}\, c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 + L_{\text{proto}}$ where $L_{\text{proto}} = \theta \cdot \|x_0 + \delta - \text{proto}_j\|_2^2$.
7: Return $x_{\text{cf}} = x_0 + \delta^*$
```

3.3 Using k-d trees as class representations

If we do not have a trained encoder available, we can build class representations using k-d trees [3]. After labeling the representative training set by calling the predictive model, we can represent each class i by a separate k-d tree built using the instances with class label i. This approach is similar to [12] who use class specific k-d trees to measure the agreement between a classifier and a modified nearest neighbour classifier on test instances. For each k-d tree $j \neq t_0$, we compute the Euclidean distance between x_0 and the k-nearest item in the tree $x_{j,k}$. The closest $x_{j,k}$ across all classes $j \neq t_0$ becomes the class prototype proto_j. Note that we are now working in the original feature space. The loss term L_{proto} is equal to:

$$L_{\text{proto}} = \theta \cdot \|x_0 + \delta - \text{proto}_j\|_2^2. \tag{10}$$

Algorithm 2 outlines the k-d trees approach.

3.4 Categorical variables

Creating meaningful perturbations for categorical data is not straightforward as the very concept of perturbing an input feature implies some notion of rank and distance between the values a variable can take. We approach this by inferring pairwise distances between categories of a categorical variable based on either model predictions (Modified Value Distance Metric) [6] or the context provided by the other variables in the dataset (Association-Based Distance Metric) [16]. We then apply multidimensional scaling [5] to project the inferred distances into one-dimensional Euclidean space, which allows us to perform perturbations in this space. After applying a perturbation in this space, we map the resulting number back to the closest category before evaluating the classifier's prediction.

3.5 Removing L_{pred}

In the absence of L_{proto}, only L_{pred} encourages the perturbed instance to predict class $i \neq t_0$. In the case of black box models where we only have access to the model's prediction function, L_{pred} can become a computational bottleneck. This means that for neural networks, we can no longer take advantage of automatic differentiation and need to evaluate the gradients numerically. Let us express the gradient of L_{pred} with respect to the input features x as follows:

$$\frac{\partial L_{\text{pred}}}{\partial x} = \frac{\partial f_\kappa(x)}{\partial x} = \frac{\partial f_\kappa(x)}{\partial f_{\text{pred}}} \frac{\partial f_{\text{pred}}}{\partial x}, \tag{11}$$

where f_{pred} represents the model's prediction function. The numerical gradient approximation for f_{pred} with respect to input feature k can be written as:

$$\frac{\partial f_{\text{pred}}}{\partial x_k} \approx \frac{f_{\text{pred}}(x + \epsilon_k) - f_{\text{pred}}(x - \epsilon_k)}{2\epsilon}, \tag{12}$$

where ϵ_k is a perturbation with the same dimension as x and taking value ϵ for feature k and 0 otherwise. As a result, the prediction function needs to be evaluated twice for each feature per gradient step just to compute $\frac{\partial f_{\text{pred}}}{\partial x_k}$. For a 28×28 MNIST image, this translates into a batch of $28 \cdot 28 \cdot 2 = 1568$ prediction function calls. Eliminating L_{pred} would therefore speed up the counterfactual search process significantly. By using the prototypes to guide the counterfactuals, we can remove L_{pred} and only call the prediction function once per gradient update on the perturbed instance to check whether the predicted class i of $x_0 + \delta$ is different from t_0. This eliminates the computational bottleneck while ensuring that the perturbed instance moves towards an interpretable counterfactual x_{cf} of class $i \neq t_0$.

3.6 FISTA optimization

Like [7], we optimize our objective function by applying a fast iterative shrinkage-thresholding algorithm (FISTA) [2] where the solution space for the output

$x_{\mathrm{cf}} = x_0 + \delta$ is restricted to $\mathcal{X}$. The optimization algorithm iteratively updates δ with momentum for N optimization steps. It also strips out the $\beta \cdot L_1$ regularization term from the objective function and instead shrinks perturbations $|\delta_k| < \beta$ for feature k to 0. The optimal counterfactual is defined as $x_{\mathrm{cf}} = x_0 + \delta^{n^*}$ where $n^* = \mathrm{argmin}_{n \in 1,\ldots,N}\, \beta \cdot \|\delta^n\|_1 + \|\delta^n\|_2^2$ and the predicted class on x_{cf} is $i \neq t_0$.

4 Experiments

The experiments are conducted on an image and tabular dataset. The first experiment on the MNIST handwritten digit dataset [17] makes use of an autoencoder to define and construct prototypes. The second experiment uses the Breast Cancer Wisconsin (Diagnostic) dataset [10]. The latter dataset has lower dimensionality so we find the prototypes using k-d trees. Finally, we illustrate our approach for handling categorical data on the Adult (Census) dataset [10].

4.1 Evaluation

The counterfactuals are evaluated on their interpretability, sparsity and speed of the search process. The sparsity is evaluated using the elastic net loss term $\mathrm{EN}(\delta) = \beta \cdot \|\delta\|_1 + \|\delta\|_2^2$ while the speed is measured by the time and the number of gradient updates required until a satisfactory counterfactual x_{cf} is found. We define a satisfactory counterfactual as the optimal counterfactual found using FISTA for a fixed value of c for which counterfactual instances exist.

In order to evaluate interpretability, we introduce two interpretability metrics IM1 and IM2. Let AE_i and AE_{t_0} be autoencoders trained specifically on instances of classes i and t_0, respectively. Then IM1 measures the ratio between the reconstruction errors of x_{cf} using AE_i and AE_{t_0}:

$$\mathrm{IM1}(\mathrm{AE}_i, \mathrm{AE}_{t_0}, x_{\mathrm{cf}}) := \frac{\|x_0 + \delta - \mathrm{AE}_i(x_0 + \delta)\|_2^2}{\|x_0 + \delta - \mathrm{AE}_{t_0}(x_0 + \delta)\|_2^2 + \epsilon}. \tag{13}$$

A lower value for IM1 means that x_{cf} can be better reconstructed by the autoencoder which has only seen instances of the counterfactual class i than by the autoencoder trained on the original class t_0. This implies that x_{cf} lies closer to the data manifold of counterfactual class i compared to t_0, which is considered to be more interpretable.

The second metric IM2 compares how similar the reconstructed counterfactual instances are when using AE_i and an autoencoder trained on all classes, AE. We scale IM2 by the L_1 norm of x_{cf} to make the metric comparable across classes:

$$\mathrm{IM2}(\mathrm{AE}_i, \mathrm{AE}, x_{\mathrm{cf}}) := \frac{\|\mathrm{AE}_i(x_0 + \delta) - \mathrm{AE}(x_0 + \delta)\|_2^2}{\|x_0 + \delta\|_1 + \epsilon}. \tag{14}$$

A low value of IM2 means that the reconstructed instances of x_{cf} are very similar when using either AE_i or AE. As a result, the data distribution of the counterfactual class i describes x_{cf} as good as the distribution over all classes.

This implies that the counterfactual is interpretable. Figure 2 illustrates the intuition behind IM1 and IM2.

The uninterpretable counterfactual 3 ($x_{\text{cf},1}$) in the first row of Figure 2(b) has an IM1 value of 1.81 compared to 1.04 for $x_{\text{cf},2}$ in the second row because the reconstruction of $x_{\text{cf},1}$ by AE_5 in Figure 2(d) is better than by AE_3 in Figure 2(c). The IM2 value of $x_{\text{cf},1}$ is higher as well—0.15 compared to 0.12 for $x_{\text{cf},2}$)—since the reconstruction by AE in Figure 2(e) yields a clear instance of the original class 5.

Finally, for MNIST we apply a multiple model comparison test based on the maximum mean discrepancy [18] to evaluate the relative interpretability of counterfactuals generated by each method.

4.2 Handwritten digits

The first experiment is conducted on the MNIST dataset. The experiment analyzes the impact of L_{proto} on the counterfactual search process with an encoder defining the prototypes for K equal to 5. We further investigate the importance of the L_{AE} and L_{pred} loss terms in the presence of L_{proto}. We evaluate and compare counterfactuals obtained by using the following loss functions:

$$\begin{aligned}
A &= c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 \\
B &= c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 + L_{\text{AE}} \\
C &= c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 + L_{\text{proto}} \\
D &= c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 + L_{\text{AE}} + L_{\text{proto}} \\
E &= \beta \cdot L_1 + L_2 + L_{\text{proto}} \\
F &= \beta \cdot L_1 + L_2 + L_{\text{AE}} + L_{\text{proto}}
\end{aligned} \tag{15}$$

For each of the ten classes, we randomly sample 50 numbers from the test set and find counterfactual instances for 3 different random seeds per sample. This brings the total number of counterfactuals to 1,500 per loss function.

The model used to classify the digits is a convolutional neural network with 2 convolution layers, each followed by a max-pooling layer. The output of the second pooling layer is flattened and fed into a fully connected layer followed by a softmax output layer over the 10 possible classes. For objective functions B to F, the experiment also uses a trained autoencoder for the L_{AE} and L_{proto} loss terms. The autoencoder has 3 convolution layers in the encoder and 3 deconvolution layers in the decoder. Full details of the classifier and autoencoder, as well as the hyperparameter values used can be found in the supplementary material.

Results Table 1 summarizes the findings for the speed and interpretability measures.

Speed Figure 3(a) shows the mean time and number of gradient steps required to find a satisfactory counterfactual for each objective function. We also show

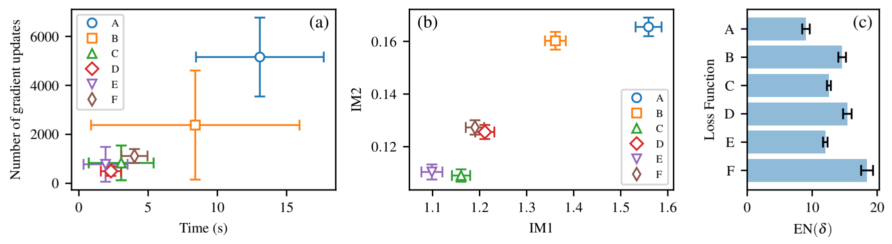


Fig. 3. (a) Mean time in seconds and number of gradient updates needed to find a satisfactory counterfactual for objective functions A to F across all MNIST classes. The error bars represent the standard deviation to illustrate variability between approaches. (b) Mean IM1 and IM2 for objective functions A to F across all MNIST classes (lower is better). The error bars represent the 95% confidence bounds. (c) Sparsity measure EN(δ) for loss functions A to F. The error bars represent the 95% confidence bounds.

Table 1. Summary statistics with 95% confidence bounds for each loss function for the MNIST experiment.

Method	**Time (s)**	**Gradient steps**	**IM1**	**IM2 (×10)**
A	13.06 ± 0.23	5158 ± 82	1.56 ± 0.03	1.65 ± 0.04
B	8.40 ± 0.38	2380 ± 113	1.36 ± 0.02	1.60 ± 0.03
C	3.06 ± 0.11	835 ± 36	1.16 ± 0.02	1.09 ± 0.02
D	2.31 ± 0.04	497 ± 10	1.21 ± 0.02	1.26 ± 0.03
E	1.93 ± 0.10	777 ± 44	1.10 ± 0.02	1.10 ± 0.03
F	4.01 ± 0.05	1116 ± 14	1.19 ± 0.02	1.27 ± 0.03

the standard deviations to illustrate the variability between the different loss functions. For loss function A, the majority of the time is spent finding a good range for c to find a balance between steering the perturbed instance away from the original class t_0 and the elastic net regularization. If c is too small, the L_1 regularization term cancels out the perturbations, but if c is too large, x_{cf} is not sparse anymore.

The aim of L_{AE} in loss function B is not to speed up convergence towards a counterfactual instance, but to have x_{cf} respect the training data distribution. This is backed up by the experiments. The average speed improvement and reduction in the number of gradient updates compared to A of respectively 36% and 54% is significant but very inconsistent given the high standard deviation. The addition of L_{proto} in C however drastically reduces the time and iterations needed by respectively 77% and 84% compared to A. The combination of L_{AE} and L_{proto} in D improves the time to find a counterfactual instance further: x_{cf} is found 82% faster compared to A, with the number of iterations down by 90%.

So far we have assumed access to the model architecture to take advantage of automatic differentiation during the counterfactual search process. L_{pred} can however form a computational bottleneck for black box models because numerical gradient calculation results in a number of prediction function calls proportionate to the dimensionality of the input features. Consider A' the equivalent of loss function A where we can only query the model's prediction function. E and F remove L_{pred} which results in approximately a 100x speed up of the counterfactual search process compared to A'. The results can be found in the supplementary material.

Quantitative interpretability IM1 peaks for loss function A and improves by respectively 13% and 26% as L_{AE} and L_{proto} are added (Figure 3(b)). This implies that including L_{proto} leads to more interpretable counterfactual instances than L_{AE} which explicitly minimizes the reconstruction error using AE. Removing L_{pred} in E yields an improvement over A of 29%. While L_{pred} encourages the perturbed instance to predict a different class than t_0, it does not impose any restrictions on the data distribution of x_{cf}. L_{proto} on the other hand implicitly encourages the perturbed instance to predict $i \neq t_0$ while minimizing the distance in latent space to a representative distribution of class i.

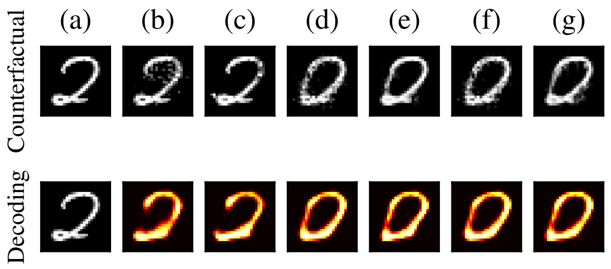


Fig. 4. (a) Shows the original instance, (b) to (g) on the first row illustrate counterfactuals generated by using loss functions A to F. (b) to (g) on the second row show the reconstructed counterfactuals using AE.

The picture for IM2 is similar. Adding in L_{proto} brings IM2 down by 34% while the combination of L_{AE} and L_{proto} only reduces the metric by 24%. For large values of K the prototypes are further from $\text{ENC}(x_0)$ resulting in larger initial perturbations towards the counterfactual class. In this case, L_{AE} ensures the overall distribution is respected which makes the reconstructed images of AE_i and AE more similar and improves IM2. The impact of K on IM1 and IM2 is illustrated in the supplementary material. The removal of L_{pred} in E and F has little impact on IM2. This emphasizes that L_{proto}—optionally in combination with L_{AE}—is the dominant term with regards to interpretability.

Finally, performing kernel multiple model comparison tests [18] indicates that counterfactuals generated by methods not including the prototype term (A and B) result in high rejection rates for faithfully modelling the predicted class distribution (see supplementary material).

Visual interpretability Figure 4 shows counterfactual examples on the first row and their reconstructions using AE on the second row for different loss functions. The counterfactuals generated with A or B are sparse but uninterpretable and are still close to the manifold of a 2. Including L_{proto} in Figure 4(d) to (g) leads to a clear, interpretable 0 which is supported by the reconstructed counterfactuals on the second row. More examples can be found in the supplementary material.

Sparsity The elastic net evaluation metric $\text{EN}(\delta)$ is also the only loss term present in A besides L_{pred}. It is therefore not surprising that A results in the most sparse counterfactuals (Figure 3(c)). The relative importance of sparsity in the objective function goes down as L_{AE} and L_{proto} are added. L_{proto} leads to more sparse counterfactuals than L_{AE} (C and E), but this effect diminishes for large K.

4.3 Breast Cancer Wisconsin (Diagnostic) Dataset

The second experiment uses the Breast Cancer Wisconsin (Diagnostic) dataset which describes characteristics of cell nuclei in an image and labels them as *malignant* or *benign*. The real-valued features for the nuclei in the image are the mean, error and worst values for characteristics like the radius, texture or area of the nuclei. The dataset contains 569 instances with 30 features each. The first 550 instances are used for training, the last 19 to generate the counterfactuals. For each instance in the test set we generate 5 counterfactuals with different random seeds. Instead of an encoder we use k-d trees to find the prototypes. We evaluate and compare counterfactuals obtained by using the following loss functions:

$$\begin{aligned} A &= c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 \\ B &= c \cdot L_{\text{pred}} + \beta \cdot L_1 + L_2 + L_{\text{proto}} \\ C &= \beta \cdot L_1 + L_2 + L_{\text{proto}} \end{aligned} \tag{16}$$

The model used to classify the instances is a 2 layer feedforward neural network with 40 neurons in each layer. More details can be found in the supplementary material.

Results Table 2 summarizes the findings for the speed and interpretability measures.

Speed L_{proto} drastically reduces the time and iterations needed to find a satisfactory counterfactual. Loss function B finds x_{cf} in 13% of the time needed compared to A while bringing the number of gradient updates down by 91%. Removing L_{pred} and solely relying on the prototype to guide x_{cf} reduces the search time by 92% and the number of iterations by 93%.

Quantitative interpretability Including L_{proto} in the loss function reduces IM1 and IM2 by respectively 55% and 81%. Removing L_{pred} in C results in similar improvements over A.

Table 2. Summary statistics with 95% confidence bounds for each loss function for the Breast Cancer Wisconsin (Diagnostic) experiment.

Method	**Time (s)**	**Gradient steps**	**IM1**	**IM2 ($\times$10)**
A	2.68 ± 0.20	2752 ± 203	2.07 ± 0.16	7.65 ± 0.79
B	0.35 ± 0.03	253 ± 33	0.94 ± 0.10	1.47 ± 0.15
C	0.22 ± 0.02	182 ± 30	0.88 ± 0.10	1.41 ± 0.15

Sparsity Loss function A yields the most sparse counterfactuals. Sparsity and interpretability should however not be considered in isolation. The dataset has 10 attributes (e.g. radius or texture) with 3 values per attribute (mean, error and worst). B and C which include L_{proto} perturb relatively more values of the same attribute than A which makes intuitive sense. If for instance the worst radius increases, the mean should typically follow as well. The supplementary material supports this statement.

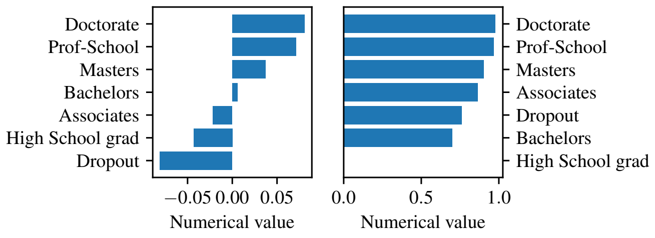


Fig. 5. Left: Embedding of the categorical variable “Education” in numerical space using association based distance metric (ABDM). Right: Frequency based embedding.

4.4 Adult (Census) Dataset

The Adult (Census) dataset consists of individuals described by a mixture of numerical and categorical features. The predictive task is to determine whether a person earns more than \$50k/year. As the dataset contains categorical features, it is important to use a principled approach to define perturbations over these features. Figure 5 illustrates our approach using the association based distance metric [16](ABDM) to embed the feature “Education” into one dimensional numerical space over which perturbations can be defined. The resulting embedding defines a natural ordering of categories in agreement with common sense for this interpretable variable. By contrast, the frequency embedding method as proposed by [9] does not capture the underlying relation between categorical values.

Since ABDM infers distances from other variables by computing dissimilarity based on the K-L divergence, it can break down if there is independence between categories. In such cases one can use MVDM [6] which uses the difference between the conditional model prediction probabilities of each category. A counterfactual example changing categorical features is shown in Figure 1.

5 Discussion

In this paper we introduce a model agnostic counterfactual search process guided by class prototypes. We show that including a prototype loss term in the objective results in more interpretable counterfactual instances as measured by two novel interpretability metrics. We demonstrate that prototypes speed up the search process and remove the numerical gradient evaluation bottleneck for black box models thus making our method more appealing for practical applications. By fixing selected features to the original values during the search process we can also obtain *actionable counterfactuals* which describe concrete steps to take to change a model's prediction. To facilitate the practical use of counterfactual explanations we provide an open source library with our implementation of the method.

References

1. Banerjee, A., Merugu, S., Dhillon, I.S., Ghosh, J.: Clustering with Bregman divergences. Journal of Machine Learning Research **6**, 1705–1749 (Dec 2005)
2. Beck, A., Teboulle, M.: A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM Journal on Imaging Sciences **2**(1), 183–202 (Mar 2009). https://doi.org/10.1137/080716542
3. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9), 509–517 (Sep 1975). https://doi.org/10.1145/361002.361007
4. Bien, J., Tibshirani, R.: Prototype selection for interpretable classification. The Annals of Applied Statistics **5**(4), 2403–2424 (12 2011). https://doi.org/10.1214/11-AOAS495
5. Borg, I., Groenen, P.: Modern Multidimensional Scaling: Theory and Applications. Springer (2005)
6. Cost, S., Salzberg, S.: A weighted nearest neighbor algorithm for learning with symbolic features. Machine Learning **10**(1), 57–78 (Jan 1993). https://doi.org/10.1023/A:1022664626993
7. Dhurandhar, A., Chen, P.Y., Luss, R., Tu, C.C., Ting, P., Shanmugam, K., Das, P.: Explanations based on the missing: Towards contrastive explanations with pertinent negatives. In: Advances in Neural Information Processing Systems 31, pp. 592–603 (2018)
8. Dhurandhar, A., Iyengar, V., Luss, R., Shanmugam, K.: Tip: Typifying the interpretability of procedures. arXiv preprint arXiv:1706.02952 (2017)
9. Dhurandhar, A., Pedapati, T., Balakrishnan, A., Chen, P.Y., Shanmugam, K., Puri, R.: Model agnostic contrastive explanations for structured data. arXiv preprint arXiv:1906.00117 (2019)
10. Dua, D., Graff, C.: UCI machine learning repository (2017)

11. Gurumoorthy, K.S., Dhurandhar, A., Cecchi, G.: Protodash: fast interpretable prototype selection. arXiv preprint arXiv:1707.01212 (2017)
12. Jiang, H., Kim, B., Guan, M., Gupta, M.: To trust or not to trust a classifier. In: Advances in Neural Information Processing Systems 31, pp. 5541–5552 (2018)
13. Kaufmann, L., Rousseeuw, P.: Clustering by means of medoids. Data Analysis based on the L1-Norm and Related Methods pp. 405–416 (01 1987)
14. Kim, B., Khanna, R., Koyejo, O.O.: Examples are not enough, learn to criticize! criticism for interpretability. In: Advances in Neural Information Processing Systems 29, pp. 2280–2288 (2016)
15. Laugel, T., Lesot, M.J., Marsala, C., Renard, X., Detyniecki, M.: Comparison-based inverse classification for interpretability in machine learning. In: Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations. pp. 100–111. Springer International Publishing (2018)
16. Le, S.Q., Ho, T.B.: An association-based dissimilarity measure for categorical data. Pattern Recognition Letters **26**(16), 2549 – 2557 (2005). https://doi.org/https://doi.org/10.1016/j.patrec.2005.06.002
17. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010)
18. Lim, J.N., Yamada, M., Schölkopf, B., Jitkrittum, W.: Kernel stein tests for multiple model comparison. In: Advances in Neural Information Processing Systems 32, pp. 2240–2250. Curran Associates, Inc. (2019)
19. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Advances in Neural Information Processing Systems 30, pp. 4765–4774 (2017)
20. Luss, R., Chen, P.Y., Dhurandhar, A., Sattigeri, P., Shanmugam, K., Tu, C.C.: Generating contrastive explanations with monotonic attribute functions. arXiv preprint arXiv:1905.12698 (2019)
21. Molnar, C.: Interpretable Machine Learning (2019), `https://christophm.github.io/interpretable-ml-book/`; accessed 22-January-2020
22. Mothilal, R.K., Sharma, A., Tan, C.: Explaining machine learning classifiers through diverse counterfactual explanations. Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency (Jan 2020). https://doi.org/10.1145/3351095.3372850
23. Ribeiro, M.T., Singh, S., Guestrin, C.: "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. pp. 1135–1144 (2016). https://doi.org/10.1145/2939672.2939778
24. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1, pp. 318–362. MIT Press, Cambridge, MA, USA (1986)
25. Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. In: Advances in Neural Information Processing Systems 30, pp. 4077–4087 (2017)
26. Takigawa, I., Kudo, M., Nakamura, A.: Convex sets as prototypes for classifying patterns. Engineering Applications of Artificial Intelligence **22**(1), 101 – 108 (2009). https://doi.org/https://doi.org/10.1016/j.engappai.2008.05.012
27. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: Automated decisions and the GDPR. Harvard journal of law & technology **31**, 841–887 (04 2018)
28. Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society. Series B (Statistical Methodology) **67**(2), 301–320 (2005)