Interpretable Counterfactual Explanations Guided by Prototypes

Arnaud Van Looveren
Seldon Technologies Ltd
avl@seldon.io

Janis Klaise
Seldon Technologies Ltd
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 the search for counterfactual instances and result in more interpretable explanations. We introduce two novel metrics to quantitatively evaluate local interpretability at the instance level. We use these metrics to illustrate the effectiveness of our method on an image and tabular dataset, respectively MNIST and Breast Cancer Wisconsin (Diagnostic). The method also eliminates the computational bottleneck that arises because of numerical gradient evaluation for *black box* models.[1]

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 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 (Molnar 2019). 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 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:

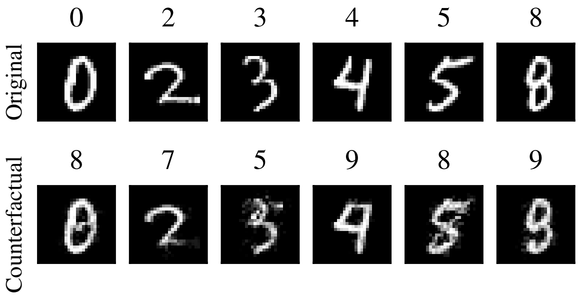

Figure 1: Examples of original and counterfactual instances on the MNIST dataset along with predictions of a CNN model. The counterfactuals are found using class prototypes.

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

[1] An open source implementation of the algorithm can be found at https://github.com/SeldonIO/alibi.

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.

2 Related Work

The problem of local, instance level model explanations for classification can be approached from various angles. Feature attribution methods assign importance to each input feature for a given prediction. Attribution methods can be fully model agnostic (Ribeiro, Singh, and Guestrin 2016; Lundberg and Lee 2017) or require knowledge of the architecture of the underlying model (Bach et al. 2015; Montavon et al. 2017; Kindermans et al. 2018). Alternatively, we can also assess the impact of individual training data instances on a specific prediction by using influence functions (Koh and Liang 2017; Koh et al. 2019; Khanna et al. 2019).

Another approach is to determine which features should remain the same so the prediction does not change. These unchanged features can be translated into *if-then rules* called *Anchors* (Ribeiro, Singh, and Guestrin 2018). Anchors are complementary to counterfactual reasoning and concepts from both approaches have been combined in the form of *Contrastive Explanations* which consist of *Pertinent Positives* and *Pertinent Negatives* (Dhurandhar et al. 2018; Dhurandhar et al. 2019). Similar to Anchors, Pertinent Positives detect the minimal and sufficient subset of features that are needed to leave the prediction unchanged. Pertinent Negatives on the other hand find feature values that should be minimally and necessarily absent in order to keep the original prediction and resemble counterfactual reasoning. Contrastive Explanations rely on the concept of neutral background values for each feature, which are often difficult to obtain. (Luss et al. 2019) tackle this issue by introducing learned monotonic attribute functions representing meaningful concepts. These high-level interpretable concepts can be learned either through labeled examples (Kim et al. 2018) or in an unsupervised fashion via disentangled representations (Kumar, Sattigeri, and Balakrishnan 2018). In order to generate realistic Contrastive Explanations, the perturbed instance needs to lie on the training data manifold modeled by generative adversarial networks (Goodfellow et al. 2014; Karras et al. 2018) or variational autoencoders (Kingma and Welling 2014).

(Wachter, Mittelstadt, and Russell 2018) suggest to 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. This leads to sparse but potentially uninterpretable counterfactuals. (Laugel et al. 2018) 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 (Wachter, Mittelstadt, and Russell 2018) or the number of sampled observations (Laugel et al. 2018; Dhurandhar et al. 2019), which can result in a computational bottleneck.

One of the key contributions of this paper is the use of prototypes to guide the counterfactual search process. (Kim, Khanna, and Koyejo 2016; Gurumoorthy, Dhurandhar, and Cecchi 2017) 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 (Kaufmann and Rousseeuw 1987), classification (Bien and Tibshirani 2011; Takigawa, Kudo, and Nakamura 2009), and few-shot learning (Snell, Swersky, and Zemel 2017). If we have access to an encoder (Rumelhart, Hinton, and Williams 1986), we follow the approach of (Snell, Swersky, and Zemel 2017) 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 (Bentley 1975).

Incorporating prototypes in the objective function leads to more interpretable counterfactuals. We introduce two novel metrics which focus on local interpretability with respect to the training data distribution. This is different from (Dhurandhar et al. 2017) who define an interpretability metric relative to a target model. (Kim, Khanna, and Koyejo 2016) 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. 2019) also highlight the importance of good local data representations in order to generate high quality explanations.

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 (Dhurandhar et al. 2018), 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 (Zou and Hastie 2005):

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

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.

(Dhurandhar et al. 2018) 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:

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

The loss L to be minimized now becomes:

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

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 7 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 7 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 9.

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:

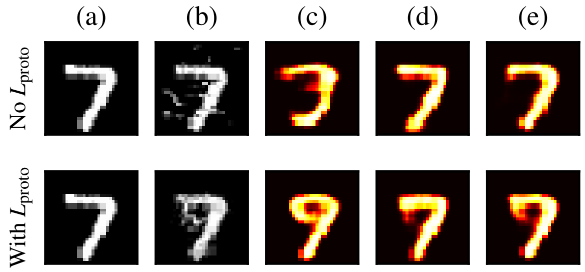

Figure 2: First row: (a) original instance and (b) uninterpretable counterfactual 3. (c), (d) and (e) are reconstructions of (b) with respectively AE_3, AE_7 and AE. Second row: (a) original instance and (b) interpretable counterfactual 9. (c), (d) and (e) are reconstructions of (b) with respectively AE_9, AE_7 and AE.

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

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. Similar to (Snell, Swersky, and Zemel 2017), the class prototype is defined as the average encoding over the instances with the same class label:

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

for $\{x_k^i\}_{k=1}^{N_i}$ 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 (Banerjee et al. 2005). 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:

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

Algorithm 1 Counterfactual search with encoded prototypes

```
Parameters: $\beta, \theta$ (required) and $c, \kappa$ and $\gamma$ (optional)
Inputs: AE (optional) and ENC models. A sample $X = \{x_1, \dots, x_n\}$ from training set. Instance to explain $x_0$.
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 $i$
    $t_0 \leftarrow \text{argmax}\, f_{\text{pred}}(x_0)$
Define prototypes for each class $i$:
    $\text{proto}_i \leftarrow \frac{1}{N_i} \sum_{k=1}^{N_i} \text{ENC}(x_k^i)$ for $x_k^i \in X^i$, $N_i = |X^i|$
Find nearest prototype $j$ to instance $x_0$ but different from original class $t_0$:
    $j \leftarrow \text{argmin}_{i \neq t_0} \|\text{ENC}(x_0) - \text{proto}_i\|_2$.
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 $L_{\text{proto}} = \theta \cdot \|\text{ENC}(x_0 + \delta) - \text{proto}_j\|_2^2$.
Return $x_{\text{cf}} = x_0 + \delta^*$
```

Algorithm 2 Counterfactual search with k-d trees

```
Parameters: $\beta, \theta, k$ (required) and $c, \kappa$ (optional)
Input: A sample $X = \{x_1, \dots, x_n\}$ from training set. Instance to explain $x_0$.
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 $i$
    $t_0 \leftarrow \text{argmax}\, f_{\text{pred}}(x_0)$
Build separate k-d trees for each class $i$ using $X_i$
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 $i$.
    $\text{proto}_j \leftarrow x_{j,k}$
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 $L_{\text{proto}} = \theta \cdot \|x_0 + \delta - \text{proto}_j\|_2^2$.
Return $x_{\text{cf}} = x_0 + \delta^*$
```

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.

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 (Bentley 1975). 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 (Jiang et al. 2018) 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 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 $i \neq t_0$.

3.5 FISTA optimization

Like (Dhurandhar et al. 2018), we optimize our objective function by applying a fast iterative shrinkage-thresholding algorithm (FISTA) (Beck and Teboulle 2009) where the solution space for the output $x_{\text{cf}} = x_0 + \delta$ 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 $x_{\text{cf}} = x_0 + \delta^{n^*}$ where $n^* = \text{argmin}_{n \in 1, \dots, N}\, \beta \cdot \|\delta^n\|_1 + \|\delta^n\|_2^2$ 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 (LeCun and Cortes 2010) makes use of an autoencoder to define and construct prototypes. The second experiment uses the Breast Cancer Wisconsin (Diagnostic) dataset (Dua and Graff 2017). The latter dataset has lower dimensionality so we find the prototypes using k-d trees.

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}:

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

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:

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

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 2.41 compared to 1.17 for $x_{\text{cf},2}$ in the second row because the reconstruction of $x_{\text{cf},1}$ by AE_7 in Figure 2(d) is much better than by AE_3 in Figure 2(c). The IM2 value of $x_{\text{cf},1}$ is higher as well—0.17 compared to 0.09 for $x_{\text{cf},2}$)—since the reconstruction by AE in Figure 2(e) yields a clear instance of the original class 7.

4.2 Handwritten digits

The first experiment is conducted on the MNIST dataset which contains 70,000 labeled 28×28 images of handwritten digits between 0 and 9. The experiment analyzes the impact of L_{proto} on the counterfactual search process with an encoder defining the prototypes. 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 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 82% and 85% 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 84% faster compared to A, with the number of iterations down by 90%.

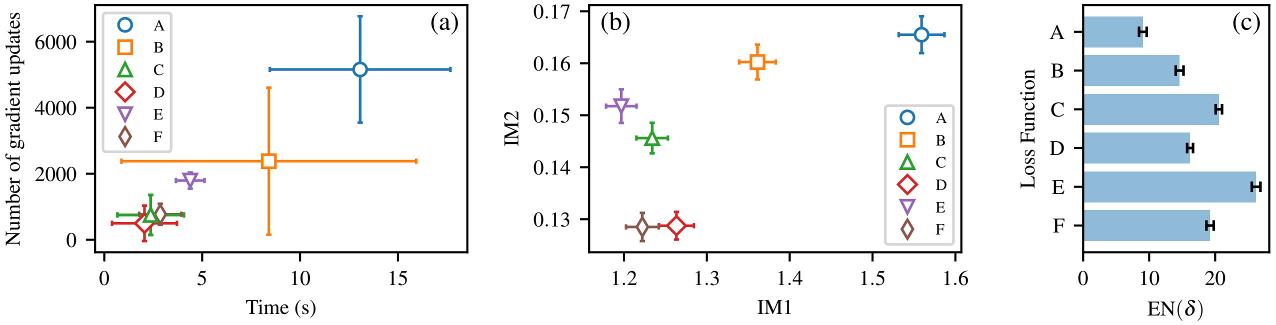

Figure 3: (a) Mean time in seconds and number of gradient updates needed to find a satisfactory counterfactual for objective functions A to F across all MNIST classes. The error bars represent the standard deviation to illustrate variability between approaches. (b) Mean IM1 and IM2 for objective functions 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.

Method	**Time (s)**	**Gradient steps**	**IM1**	**IM2 ($\times$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	2.37 ± 0.09	751 ± 31	1.23 ± 0.02	1.46 ± 0.03
D	2.05 ± 0.08	498 ± 27	1.26 ± 0.02	1.29 ± 0.03
E	4.39 ± 0.04	1794 ± 12	1.20 ± 0.02	1.52 ± 0.03
F	2.86 ± 0.06	773 ± 16	1.22 ± 0.02	1.29 ± 0.03

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

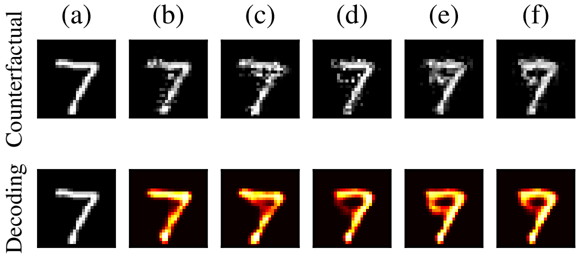

Figure 4: (a) Shows the original instance, (b) to (f) on the first row illustrate counterfactuals generated by using loss functions A, B, C, D and F. (b) to (f) on the second row show the reconstructed counterfactuals using AE.

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 calculcation 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 21% 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 23%. 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.

The picture for IM2 is slightly different. Adding in L_{proto} brings IM2 down by 12%. The combination of L_{AE} and L_{proto} however reduces the metric by 22%. L_{proto} generates an instance x_{cf} closer to the distribution of class i while L_{AE} ensures the overall distribution is respected which makes the reconstructed images of AE$_i$ and AE more similar and improves IM2. 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.

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 7. Including L_{proto} in Figure 4(d) to (f) leads to a clear, interpretable 9 which is supported by the reconstructed the counterfactuals on the second row. More counterfactual examples can be found in the supplementary material.

Sparsity The elastic net evaluation metric EN(δ) 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. Steering x_{cf} towards $\text{proto}_{i \neq t_0}$ encourages less sparse but more interpretable solutions.

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

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

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 feed-forward neural network with 40 neurons in each layer. More details about the setup 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.

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.

5 Discussion

Counterfactual reasoning comes natural to humans and specific counterfactual instances highlight the necessary changes to alter the outcome of a decision making process. As such they are applicable to a wide range of use cases for putting the decisions of machine learning models in context (e.g. a necessary change in the income level to result in a positive loan application or a change in symptoms to result in an alternative medical diagnosis). In this paper we introduced a model agnostic counterfactual search process guided by class prototypes. We showed that including a prototype loss term in the objective results in more interpretable counterfactual instances as measured by two novel interpretability metrics. In addition, 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 of model interpretability.

One of the drawbacks of this and other techniques for finding counterfactual instances (Wachter, Mittelstadt, and Russell 2018; Laugel et al. 2018; Dhurandhar et al. 2018) is that they don't work out of the box for categorical variables. As an example, assume that one-hot encoding is applied to a categorical feature with N categories. The perturbations for each of the N encoded categories should then be restricted to $\{0, 1\}$ or $\{-1, 0\}$ depending on the initial feature value. Moreover, two different encoded categories i and j are typically mutually exclusive so perturbations in the encoded space are not independent. While L_{AE} and L_{proto} would punish out-of-distribution counterfactuals that violate the mutually exclusive category condition, there is no mechanism to strictly enforce it. The categorical feature perturbation needs to be interpretable as well. A small perturbation δ_k to a continuous numerical feature x_k implies that $x_k + \delta_k$ is close to x_k in the feature space. There are no such guarantees when changing the category of a categorical feature from i to j.

(Dhurandhar et al. 2019) propose to transform categories into real valued features in $[0, 1]$ based on category frequency. This defines a natural ordering between categories which does not necessarily correspond to the underlying relationship between them. Consider for instance an income prediction model on the US population with a person's education level as a feature. According to (U.S. Census Bureau 2019) there are roughly 10% of high school dropouts among adults above the age of 25, similar to the 13% of the population with an advanced degree such as a Master's. As a result, the frequency mapping assigns a very similar real value to both the *high school drop out* and *advanced degree* categories, which does not reflect the actual meaning of the categories. Extending the counterfactual search method to handle categorical features in a principled way is an open research direction, necessary for many applications.

In summary, our work takes into account various *desiderata* to generate interpretable counterfactual instances in a fraction of the time compared to other methods. We believe these properties make for an appealing and practical model explanation method.

References

Bach, S.; Binder, A.; Montavon, G.; Klauschen, F.; Mller, K.-R.; and Samek, W. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE* 10(7):1–46.

Banerjee, A.; Merugu, S.; Dhillon, I. S.; and Ghosh, J. 2005. Clustering with bregman divergences. *Journal of Machine Learning Research* 6:1705–1749.

Beck, A., and Teboulle, M. 2009. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences* 2(1):183–202.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18(9):509–517.

Bien, J., and Tibshirani, R. 2011. Prototype selection for interpretable classification. *The Annals of Applied Statistics* 5(4):2403–2424.

Dhurandhar, A.; Iyengar, V.; Luss, R.; and Shanmugam, K. 2017. Tip: Typifying the interpretability of procedures. *arXiv preprint arXiv:1706.02952*.

Dhurandhar, A.; Chen, P.-Y.; Luss, R.; Tu, C.-C.; Ting, P.; Shanmugam, K.; and Das, P. 2018. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. In *Advances in Neural Information Processing Systems 31*. 592–603.

Dhurandhar, A.; Pedapati, T.; Balakrishnan, A.; Chen, P.-Y.; Shanmugam, K.; and Puri, R. 2019. Model agnostic contrastive explanations for structured data. *arXiv preprint arXiv:1906.00117*.

Dua, D., and Graff, C. 2017. UCI machine learning repository.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*. 2672–2680.

Gurumoorthy, K. S.; Dhurandhar, A.; and Cecchi, G. 2017. Protodash: fast interpretable prototype selection. *arXiv preprint arXiv:1707.01212*.

Jiang, H.; Kim, B.; Guan, M.; and Gupta, M. 2018. To trust or not to trust a classifier. In *Advances in Neural Information Processing Systems 31*. 5541–5552.

Karras, T.; Aila, T.; Laine, S.; and Lehtinen, J. 2018. Progressive growing of GANs for improved quality, stability, and variation. In *6th International Conference on Learning Representations*.

Kaufmann, L., and Rousseeuw, P. 1987. Clustering by means of medoids. *Data Analysis based on the L1-Norm and Related Methods* 405–416.

Khanna, R.; Kim, B.; Ghosh, J.; and Koyejo, S. 2019. Interpreting black box predictions using fisher kernels. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 3382–3390.

Kim, B.; Wattenberg, M.; Gilmer, J.; Cai, C. J.; Wexler, J.; Viégas, F. B.; and Sayres, R. 2018. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In *Proceedings of the 35th International Conference on Machine Learning*, 2673–2682.

Kim, B.; Khanna, R.; and Koyejo, O. O. 2016. Examples are not enough, learn to criticize! criticism for interpretability. In *Advances in Neural Information Processing Systems 29*. 2280–2288.

Kindermans, P.-J.; Schtt, K. T.; Alber, M.; Mller, K.-R.; Erhan, D.; Kim, B.; and Dhne, S. 2018. Learning how to explain neural networks: PatternNet and PatternAttribution. In *International Conference on Learning Representations*.

Kingma, D. P., and Welling, M. 2014. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations*.

Koh, P. W., and Liang, P. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 1885–1894.

Koh, P. W.; Ang, K.-S.; Teo, H. H.; and Liang, P. 2019. On the accuracy of influence functions for measuring group effects. *arXiv preprint arXiv:1905.13289*.

Kumar, A.; Sattigeri, P.; and Balakrishnan, A. 2018. Variational inference of disentangled latent concepts from unlabeled observations. In *6th International Conference on Learning Representations*.

Laugel, T.; Lesot, M.-J.; Marsala, C.; Renard, X.; and Detyniecki, M. 2018. Comparison-based inverse classification for interpretability in machine learning. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*, 100–111. Springer International Publishing.

LeCun, Y., and Cortes, C. 2010. MNIST handwritten digit database.

Lundberg, S. M., and Lee, S.-I. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*. 4765–4774.

Luss, R.; Chen, P.-Y.; Dhurandhar, A.; Sattigeri, P.; Shanmugam, K.; and Tu, C.-C. 2019. Generating contrastive explanations with monotonic attribute functions. *arXiv preprint arXiv:1905.12698*.

Molnar, C. 2019. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/; accessed 24-June-2019.

Montavon, G.; Lapuschkin, S.; Binder, A.; Samek, W.; and Mller, K.-R. 2017. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition* 65:211 – 222.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "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*, 1135–1144.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2018. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence*.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. Cambridge, MA, USA: MIT Press. 318–362.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30*. 4077–4087.

Takigawa, I.; Kudo, M.; and Nakamura, A. 2009. Convex sets as prototypes for classifying patterns. *Engineering Applications of Artificial Intelligence* 22(1):101 – 108.

U.S. Census Bureau. 2019. Educational attainment in the united states: 2018. https://www.census.gov/data/tables/2018/demo/education-attainment/cps-detailed-tables.html; accessed 24-June-2019.

Wachter, S.; Mittelstadt, B.; and Russell, C. 2018. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard journal of law & technology* 31:841–887.

Zou, H., and Hastie, T. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 67(2):301–320.

Supplementary Material

Breast Cancer Wisconsin experiment details

The classification model used to classify the cell nuclei into the *malignant* or *benign* categories is a 2 layer feedforward neural network with 40 neurons and ReLU activations in each layer. The model is trained on standardized features with stochastic gradient descent for 500 epochs with batch size 128 and reaches 100% accuracy on the test set.

The class specific autoencoders used to evaluate IM1 and IM2 consist of 3 dense layers in the encoder with respectively 20, 10 and 6 neurons for each layer. The first 2 layers have ReLU activations whilst the last one has a linear activation. The dense layers in the decoder contain 10 and 20 neurons followed by a linear layer projecting the reconstructed instance back to the input feature space. The autoencoders are optimized with Adam and trained for 500 epochs on batches of 128 instances with the mean squared error between the original and reconstructed instance as the loss function.

Similar to the MNIST experiment, parameters c, κ and β are kept constant throughout the experiments at 1, 0 and 0.1. The impact of different values for hyperparameters θ and k is visualized in Figures 6 and 7. The figures show that a broad range of values for both θ and k work well.

All experiments were run on a Thinkpad T480 with an Intel Core i7-8550U Processor.

Breast Cancer Wisconsin results

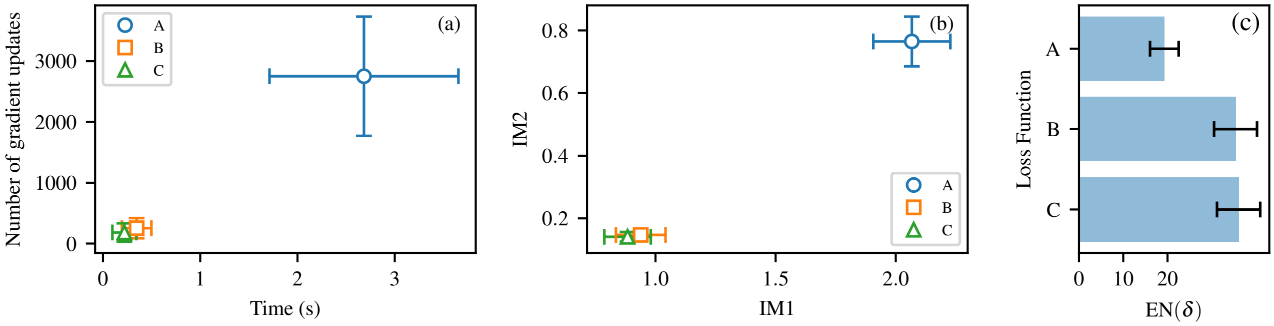

Figure 5: (a) Mean time in seconds and number of gradient updates needed to find a satisfactory counterfactual for objective functions A, B and C (16) for the Breast Cancer Wisconsin dataset. The error bars represent the standard deviation to illustrate variability between approaches. (b) Mean IM1 and IM2 for objective functions A, B and C (lower is better). The error bars represent the 95% confidence bounds. (c) Sparsity measure EN(δ) for loss functions A, B and C. The error bars represent the 95% confidence bounds.

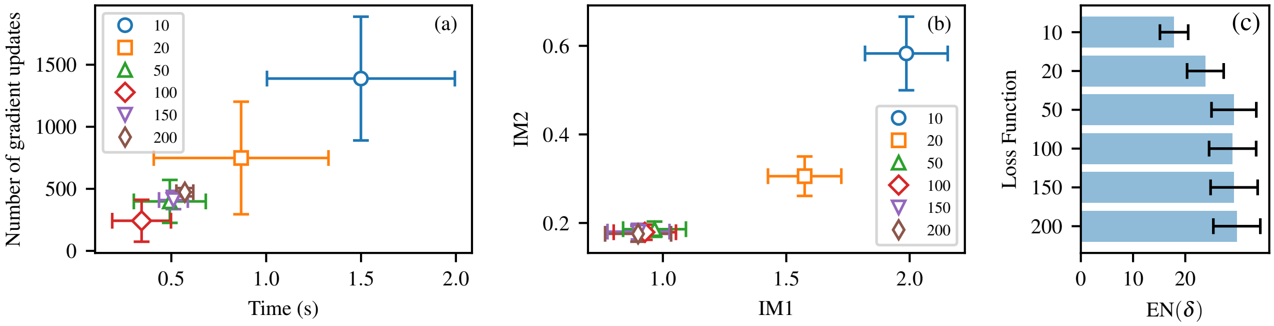

Figure 6: Impact of θ. (a) Mean time in seconds and number of gradient updates needed to find a satisfactory counterfactual for objective function B with different values of θ (10, 20, 50, 100, 150, 200) for the Breast Cancer Wisconsin dataset. The error bars represent the standard deviation to illustrate variability between approaches. (b) Mean IM1 and IM2 for objective function B for different values of θ (lower is better). The error bars represent the 95% confidence bounds. (c) Sparsity measure EN(δ) for loss functions B and different θ values. The error bars represent the 95% confidence bounds.

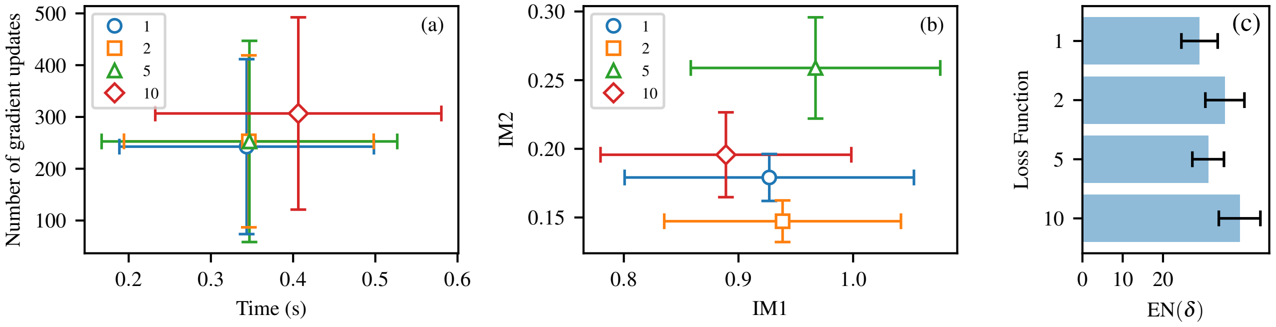

Figure 7: Impact of k. (a) Mean time in seconds and number of gradient updates needed to find a satisfactory counterfactual for objective function B with different values for the kth nearest instance in each class (k set to 1, 2, 5 and 10) which is used to define the prototype for the Breast Cancer Wisconsin dataset. The error bars represent the standard deviation to illustrate variability between approaches. (b) Mean IM1 and IM2 for objective function B for different values of k (lower is better). The error bars represent the 95% confidence bounds. (c) Sparsity measure EN(δ) for loss functions B and different k values. The error bars represent the 95% confidence bounds.

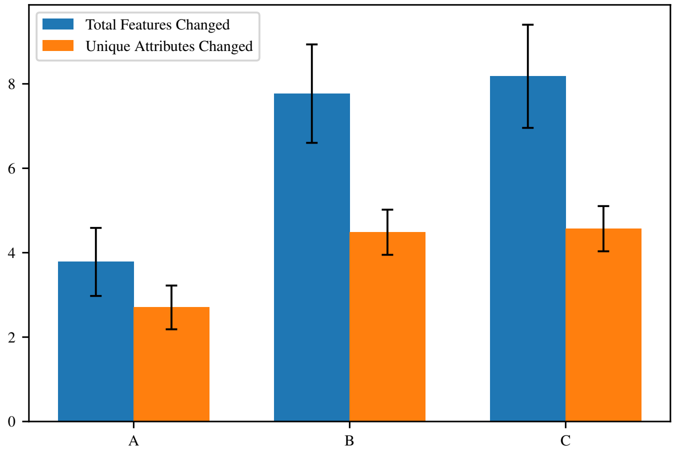

Figure 8: Total number of features and unique number of attributes changed by more than 1 standard deviation in x_{cf} compared to x_0 for loss functions A, B and C (16). The error bars represent the 95% confidence bound. A leads to sparser counterfactuals than B and C but perturbs relatively more unique attributes (e.g. radius or texture) while B and C perturb relatively more features of the same attribute (e.g. mean or worst value of the attribute).

MNIST experiment details

The classification model consists of 2 convolutional layers with respectively 64 and 32 2×2 filters and ReLU activations. Each convolutional layer is followed by a 2×2 max-pooling layer. Dropout with fraction 30% is applied during training. The output of the second pooling layer is flattened and fed into a fully connected layer of size 256 with ReLU activation and 50% dropout. This dense layer is followed by a softmax output layer over the 10 classes. The model is trained with an Adam optimizer for 3 epochs with batch size 64 on MNIST images scaled to $[-0.5, 0.5]$ and reaches a test accuracy of 98.6%.

The autoencoder used in objective functions B to F (15) has 3 convolutional layers in the encoder. The first 2 contain 16 3×3 filters and ReLU activations and are followed by a 2×2 max-pooling layer which feeds into a convolution layer with 1 3×3 filter and linear activation. The decoder takes the encoded instance as input and feeds it into a convolutional layer with 16 3×3 filters and ReLU activations, followed by a 2×2 upsampling layer and again the same convolutional layer. The final convolutional is similar to the last layer in the encoder. All the convolutions have *same* padding. The autoencoder is trained with an Adam optimizer for 4 epochs with batch size 128 and uses the mean squared error between the original and reconstructed instance as the loss function.

The class specific autoencoders used to evaluate IM1 and IM2 consist of 3 convolutional layers with 3×3 filters and ReLU activations in the encoder, each followed by 2×2 max-pooling layers. The first one contains 16 filters while the others have 8 filters. The decoder follows the same architecture in reversed order and with upsampling instead of max-pooling. The autoencoder is trained with an Adam optimizer for 30 epochs and batch size 128.

Parameters c, κ, β and γ are kept constant throughout the experiments at 1, 0, 0.1 and 100. Both L_{AE} and L_{proto} are reconstruction errors, but L_{AE} works on the full input feature space while L_{proto} operates on the compressed latent space. θ is therefore set at 200 for loss function C and 100 if used in combination with L_{AE} in D. As the only loss term besides the elastic net regularizer, θ is increased for E to 500.

All experiments were run on a Thinkpad T480 with an Intel Core i7-8550U Processor.

MNIST results

Method	Time (s)
A’	54.64 ± 1.28
E	0.53 ± 0.04
F	0.72 ± 0.01

Table 3: Mean time in seconds needed to compute 100 optimization steps for objective functions A', E and F with 95% confidence bounds. A' is the equivalent of A without access to the model architecture. As a result, we can only query the prediction function and need to evaluate gradients numerically. One test instance is used for each class in MNIST.

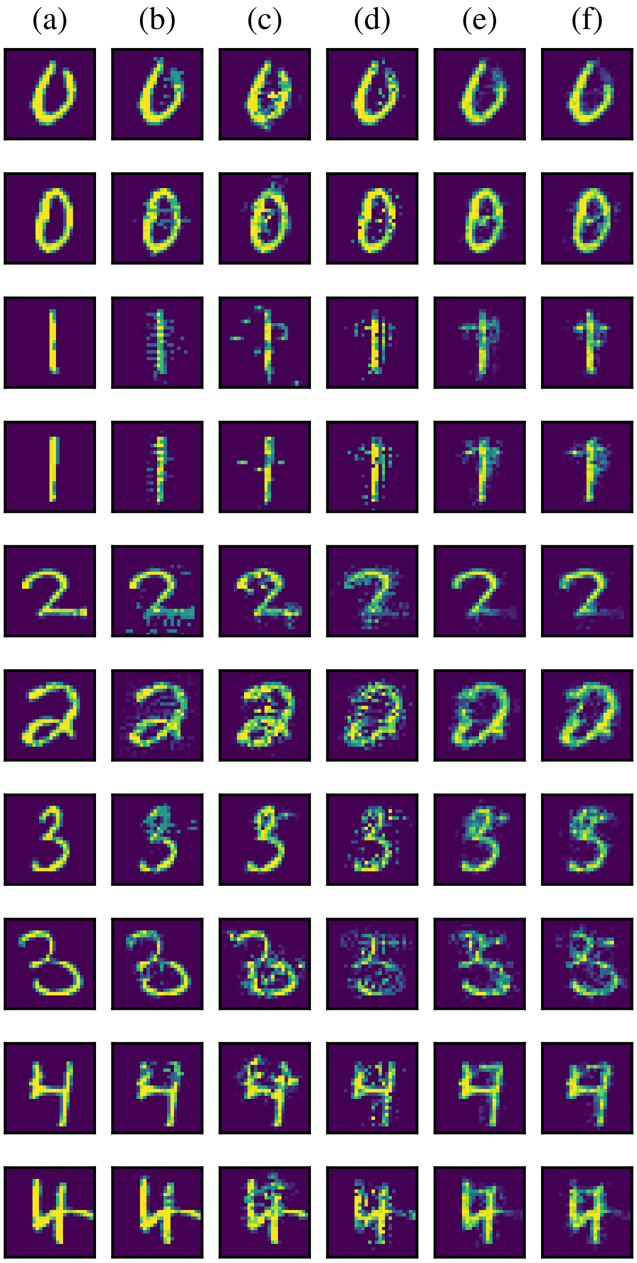

Figure 9: (a) Shows the original instance, (b) to (f) illustrate counterfactuals generated by using loss functions A, B, C, D and F.

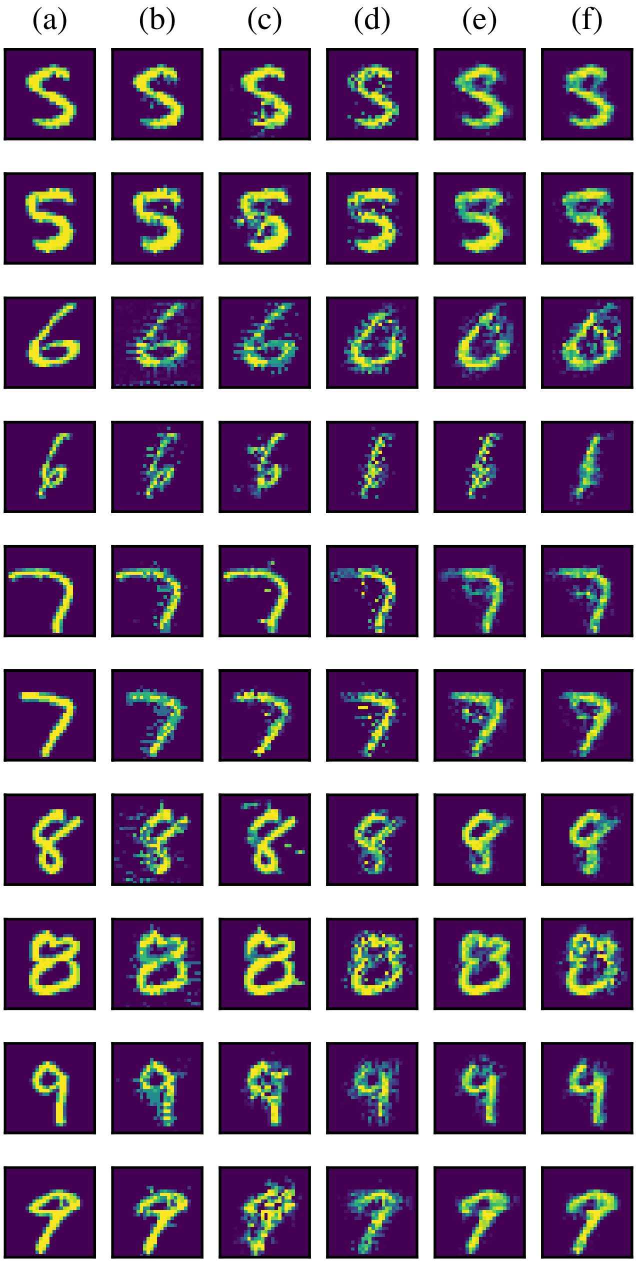

Figure 10: (a) Shows the original instance, (b) to (f) illustrate counterfactuals generated by using loss functions A, B, C, D and F.