Learning Time Series Counterfactuals via Latent Space Representations

Zhendong Wang[1](✉), Isak Samsten[1], Rami Mochaourab[2], and Panagiotis Papapetrou[1]

[1] Stockholm University, Stockholm, Sweden
{zhendong.wang,samsten,panagiotis}@dsv.su.se
[2] RISE Research Institutes of Sweden, Stockholm, Sweden
rami.mochaourab@ri.se

Abstract. Counterfactual explanations can provide sample-based explanations of features required to modify from the original sample to change the classification result from an undesired state to a desired state; hence it provides interpretability of the model. Previous work of `LatentCF` presents an algorithm for image data that employs auto-encoder models to directly transform original samples into counterfactuals in a latent space representation. In our paper, we adapt the approach to time series classification and propose an improved algorithm named `LatentCF++` which introduces additional constraints in the counterfactual generation process. We conduct an extensive experiment on a total of 40 datasets from the UCR archive, comparing to current state-of-the-art methods. Based on our evaluation metrics, we show that the `LatentCF++` framework can with high probability generate valid counterfactuals and achieve comparable explanations to current state-of-the-art. Our proposed approach can also generate counterfactuals that are considerably closer to the decision boundary in terms of margin difference.

Keywords: Time series classification · Interpretability · Counterfactual explanations · Deep learning

1 Introduction

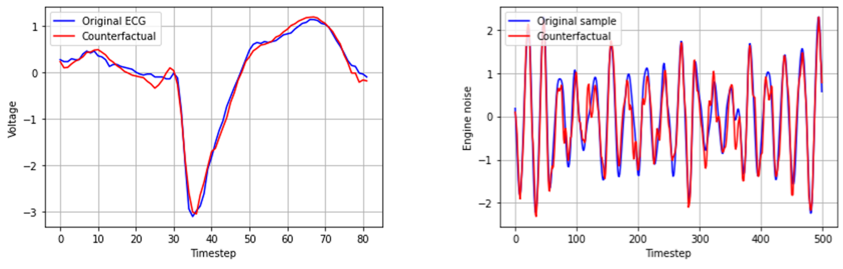
Machine learning (ML) is developing rapidly to address real-world classification problems and automate decisions in different fields. Especially, time series classification (TSC) has gained popularity in many critical applications, such as Electrocardiogram (ECG) signal classification [9], sensor signal classification [19], and stream monitoring [16]. Nevertheless, most ML methods remain opaque, although model interpretability is crucial to gaining trust from practitioners. Recent governmental regulations, such as the EU General Data Protection Regulation (GDPR), indicate that the public is entitled to receive "*meaningful information*" from automated decision-making processes [18]. Towards that direction, Wachter et al. [18] suggested *counterfactual explanations* as a solution to provide sample-based explanations, aligning with the data protection regulations from GDPR. Counterfactuals provide information on which features of the original test example are required to be modified and how to change the classification result from an undesired state (e.g., “abnormal” ECG) to a desired state (e.g., “normal” ECG), without opening the “black-box” classifier.

Several earlier approaches towards time series counterfactuals have appeared in the literature, with the main highlights including two techniques presented by Karlsson et al. [11]. The two techniques are model specific and they are defined for the random shapelet forest (RSF) classifier [10] as well as the k-NN classifier. Nonetheless, both techniques focus on specific evaluation metrics, i.e., *compactness*, referring to the fraction of time series points that need to alter in order to create a counterfactual, and *cost*, referring to the distance of the original time series to its counterfactual. Despite their promising performance on a large set of collection of time series datasets, both techniques are hampered by the chosen evaluation metrics as they mostly focus on minimizing the two selected metrics, which fail to assess whether the generated counterfactuals are compliant with the data distribution and they fall within the dataset manifold.

A recent approach that attempts to address some of the above limitations for image data has been proposed by Balasubramanian et al. [3], where the `LatentCF` framework was proposed to generate counterfactuals by means of a representative latent space using auto-encoders. The framework established a simple baseline for counterfactual explanations using latent spaces representations. However, the authors solely evaluated their method on image data, while we observed the limitation of ineffective counterfactual generation when we replicated their experiments. In this paper, we re-formulate this problem for the time series domain, and present an adaptation of the original approach, which we refer to as `LatentCF++`, by integrating Adam optimization and additional constraints on the latent space perturbation to generate more robust counterfactuals. Additionally, we demonstrate the generalizability considering several deep learning models serving as components to construct effective `LatentCF++` instantiations.

To highlight the importance of the problem we solve in this paper, consider the example in Fig. 1, where we provide two examples of time series counterfactuals generated by `LatentCF++` using two datasets from the UCR time series repository: *TwoLeadECG* (left) and *FordA* (right). Illustrated in blue are the original time series and in red the generated counterfactuals of the opposite class. By inspecting these counterfactuals, domain experts can not only gain improved understandings of the classifier decision boundary, but also can gain insight on how these predictions can be reversed.

Related Work. There is a wide range of TSC algorithms proposed using different techniques in recent years, such as elastic distance measures, intervals, shapelets and ensemble methods [2]. It is different from traditional ML classification problems due to the feature dependency of ordered data. Shapelet-based methods (e.g. shapelet transformations) identify shapelets (subsequences of whole time series) used as discriminatory features to train classifiers such as SVM and random forest [2,10]. More recently, researchers have introduced sev-



(a) ECG example (b) FordA example

Fig. 1. Examples of generated counterfactuals using `LatentCF++` on *TwoLeadECG* (left) and *FordA* (right). Illustrated in blue are the original time series and in red the generated counterfactuals of the opposite class. (Color figure online)

eral breakthrough algorithms with comparable benchmark metrics in TSC that are considered state-of-the-art, e.g., InceptionTime [7] and ROCKET [5]. Nevertheless, most of these methods are considered black-box models. As such, they lack model transparency and prediction interpretability.

Interpretability is crucial to help uncover domain findings in opaque machine learning models and has recently attained increased attention[13]. Counterfactual explanations have surged in the last few years in high-stake applications among the wide range of interpretable methods [17]. For TSC problems, Ates et al. [1] presented a counterfactual explanation solution on sample-based predictions using CORELS, focusing on multivariate time series datasets. Similarly, Karlsson et al. [11] proposed local and global solutions for counterfactual explanation problems on univariate time series data utilizing RSF and k-NN classifiers. However, both local and global approaches were proposed to provide model-specific explanations, since they cannot be applied in conjunction with other classifiers, e.g. neural networks or other state-of-the-art models in TSC.

A large number of counterfactual approaches were proposed that can provide model-agnostic explanations for any black-box classifier [17]. By utilizing a variational auto-encoder (VAE) architecture, Pawelczyk et al. [15] conducted an experiment to generate counterfactuals uniformly around spheres of the original data representation for tabular data. Joshi et al. [8] applied a VAE as the generation model, a linear softmax classifier as the classification model, to sample the set of counterfactuals with high probability paths of changes to change the outcome. The `LatentCF` approach was presented to apply gradient descent on the latent representations of input samples and transform into counterfactuals using an auto-encoder (AE) [3]. To the best of our knowledge, these counterfactual solutions using latent representations mainly focused on tabular or image data, and none of them has been generalized in the TSC domain.

Contributions. We propose a novel time series counterfactual generation framework, named `LatentCF++`, that ensures the robustness and closeness of the generated counterfactuals. Our specific contributions of this paper are summarized as follows:

- we formulate the time series counterfactual explanation problem for univariate time series data and provide a novel solution `LatentCF++` to solve the problem by employing a latent representation for generating counterfactuals;
- we demonstrate two instantiations of our proposed solution, where we incorporate classifiers and auto-encoders based on convolutional neural networks and LSTM, together with an extension of the model based on a composite auto-encoder;
- we conduct an in-depth experiment comparing our solution and the original framework, together with current state-of-the-art `RSF`, `k-NN` and `FGD` as baseline models. We show that our proposed framework outperforms other methods in terms of *validity* and *margin difference*; while it achieves the comparable performance of *proximity* compared to `RSF`.

2 Problem Formulation

Similar to the definition of counterfactual time series explanation problem for multivariate time series classification [1], we define univariate time series counterfactual explanations as follows: given a black-box ML model that predicts the binary output probability from a univariate time series sample, the counterfactual method shows the modifications of the input sample that are required to change the classification result from an undesired state (e.g. negative) to a desired state (e.g. positive). We assume a given classifier in our formulation, and a pre-trained auto-encoder that can transform the time series into the latent representation and back to the original feature space. Note that we do not need to access internal structures of the classifier (e.g. weights); only the prediction function is required. Let $\mathcal{Y}$ defines the set of target class labels, and we consider a binary classification problem where $\mathcal{Y} = \{\text{`}+\text{'}, \text{`}-\text{'}\}$. The formal definition of the problem is as follows:

Problem 1. **Univariate time series counterfactual explanations.** Give any given classifier $f(\cdot)$ and a time series sample x containing t timesteps, such that the output represents as $f(x) = \text{`}-\text{'}$ with probability $\hat{y}$. In the problem, $\hat{y}$ is smaller than the decision boundary threshold τ (i.e. $\hat{y} < \tau$) as it determines negative. Our goal is to utilize an auto-encoder composed of an encode function $E(\cdot)$ and a decode function $D(\cdot)$ to find the generated counterfactual x' with desired positive outcome. We want to perturb the encoded latent representation $z = E(x)$ through a gradient descent optimization iteratively to generate a new time series sample $x' = D(z)$ such that the output target $f(x') = \text{`}+\text{'}$. Finally, we aim to minimize the objective function denoting the loss between $\hat{y}$ and τ.

For example, given a classifier $f(\cdot)$ and an auto-encoder with functions $E(\cdot)$ and $D(\cdot)$ trained on time series of ECG measurements, we intend to generate counterfactuals through both $E(\cdot)$ and $D(\cdot)$ functions. An exemplified counterfactual x' with the desired prediction $f(x') = \text{`}+\text{'}$ (i.e. *normal*) for a time series sample $x$ with an undesired prediction $f(x) = \text{`}-\text{'}$ (i.e. *abnormal*).

3 The LatentCF++ Time Series Counterfactuals Framework

The `LatentCF` framework was presented as a simple baseline for counterfactual explanations, which employs an auto-encoder model to directly transform original samples into counterfactuals using gradient descent to search in the latent space [3]. Due to the low efficiency of gradient descent in the original implementation, it requires an extensive search for the proper learning rate to avoid getting stuck in a local optimum. As such, `LatentCF` often fails to generate valid counterfactuals.

Instead, we propose to improve the counterfactual generation process by integrating an adequate gradient-based optimization algorithm based on adaptive moment estimation (Adam) [12], making the counterfactual generation more robust. Adam optimization can be considered a combination of two momentum-based gradient descent algorithms AdaGrad and RMS-Prop. Employing the advantages of both allows the algorithm to deal with sparse features and non-stationary objectives and obtain faster convergence [12]. To further improve the validity of the counterfactual explanations, we add constraints (see Line 5 in Algorithm 1) to ensure that the class probability of the generated counterfactual has crossed the decision boundary threshold τ. We call the improved version of the `LatentCF` framework, `LatentCF++`.

Algorithm 1: Counterfactual explanations for time series classification using `LatentCF++`

```
input  : A time series sample $x$, threshold of decision boundary $\tau$, learning
         rate $\alpha$, loss tolerance $tol$, maximum iteration $max\_iter$
output: A generated counterfactual $x'$ with desired target class $y'$
1  $z \leftarrow$ Encode($x$)
2  $y_{pred} \leftarrow$ Predict( Decode($z$))
3  $loss \leftarrow$ MSE($y_{pred} -\ \tau$)
4  $iter \leftarrow 0$
5  while $loss > tol \wedge y_{pred} < \tau \wedge iter < max\_iter$ do
6      $z \leftarrow$ AdamOptimize($z, loss, \alpha$)
7      $y_{pred} \leftarrow$ Predict( Decode($z$))
8      $loss \leftarrow$ MSE( $y_{pred} -\ \tau$)
9      $iter \leftarrow iter + 1$
10 $x' \leftarrow$ Decode($z$)
11 return $x'$
```

Given an input of time series sample x, a pre-trained encoder encodes it into latent representation z, followed by a decoder that reconstructs it back to the original feature space. Consecutively, a predictor function estimates the class probability of the reconstructed sample. On Line 5, the constraints are validated to guarantee that the loss iteratively decreases and that the output probability

y_{pred} crosses the threshold of decision boundary τ. The loss is measured using the mean of squared error between y_{pred} and τ. Subsequently, on Line 6 the `AdamOptimize()` function is used to update the latent representation z using Adam. The output counterfactual is the decoded result x' when the while loop condition breaks (i.e., either $loss$ is lower than tol, y_{pred} is larger than or equal to τ, or the while loop reaches the maximum number of allowed iterations).

Finally, as we can observe from the architecture in Algorithm 1, the algorithm requires two main components: a pre-trained classifier and a pre-trained auto-encoder which can decompose into an encoder and a decoder.

4 Experimental Evaluation

We conduct our experiments using the UCR time series archive [4]. We mainly focus on the problem of counterfactuals for binary univariate time classification. After filtering, a subset of 40 datasets from the UCR archive is selected, containing representations from different data sources. For example, *TwoLeadECG* represents ECG measurements in the medical domain and *Wafer* exemplifies sensor data in semiconductor manufacturing. In terms of time series length, it varies from 24 (*ItalyPowerDemand*) to 2709 timesteps (*HandOutlines*) in our chosen subset. For the evaluation, we choose to apply a standard stratified split of 80% for training and the remainder for testing, for all the datasets. Moreover, to compensate for the imbalanced target classes, we apply an up-sampling technique that resamples the minority class during training.

4.1 Experiment Setup

There are three main deep neural network architectures that have been adopted for time series classification tasks in recent years: multi-layer perceptron (MLP), convolutional neural networks (CNN) and recurrent neural networks (RNN) [6]. In the experiment, we choose to train separate CNN and long short-term memory (LSTM, a variant of RNN) classification and auto-encoder models, as main components to apply the `LatentCF` and `LatentCF++` frameworks. Despite the fact that more recent state-of-the-art architectures have been proposed in the

Table 1. Summary of model components and hyperparameters for each instantiation in the experiment.

Method	Instantiation	Auto-encoder	Classifier	Optimization	Threshold
`LatentCF++`	`1dCNN`	`1dCNN-AE`	`1dCNN-CLF`	Adam	0.5
	`LSTM`	`LSTM-AE`	`LSTM-CLF`	Adam	0.5
	`1dCNN-C`	`1dCNN-C`		Adam	0.5
`LatentCF`	`1dCNN`	`1dCNN-AE`	`1dCNN-CLF`	Vanilla GD	No
	`LSTM`	`LSTM-AE`	`LSTM-CLF`	Vanilla GD	No

Table 2. Summary of architectures and hyperparameters for the deep learning models, representing different components.

Instan.	Components		**#Layers**	**#Conv**	**#LSTM**	**Norm**	**Pooling**	**Output**
`1dCNN`	`1dCNN-AE`		5	5	0	No	Max	Linear
	`1dCNN-CLF`	*(shallow)*	3	1	0	Yes	Max	Sigmoid
		(deep)	4	3	0	Yes	Max	Sigmoid
`LSTM`	`LSTM-AE`		5	0	4	No	Max	Sigmoid
	`LSTM-CLF`	*(shallow)*	2	0	1	Yes	None	Sigmoid
		(deep)	3	0	2	Yes	None	Sigmoid
`1dCNN-C`	`1dCNN-C`		8	6	0	No	Max	Lin.+Sig.

literature, we opt to use these simpler architectures to highlight the explainable power of latent space representations.

Thus, we construct two instantiations for both `LatentCF` and `LatentCF++` in our experiment: `1dCNN` and `LSTM`. We show a detailed comparison of different components and hyperparameters for each instantiation in Table 1. For illustration, the `1dCNN` instantiation comprises a `1dCNN-AE` auto-encoder and a `1dCNN-CLF` classifier, together with an Adam optimization and the probability threshold of 0.5. Additionally, `LatentCF++` is extended with one composite auto-encoder structure (`1dCNN-C`) instead of utilizing the two components.

More specifically, to evaluate `LatentCF` and `LatentCF++` for each dataset, we first train five deep learning models representing different components in the framework: CNN auto-encoder (`1dCNN-AE`), CNN classifier (`1dCNN-CLF`), LSTM auto-encoder (`LSTM-AE`), LSTM classifier (`LSTM-CLF`) and CNN composite auto-encoder (`1dCNN-C`). Since our goal is not to assess the performance of classifiers or auto-encoders, we apply a one-size-fits-all plan to utilize a standard set of model architectures for all the datasets. Table 2 shows architectures and hyper-parameters for all different deep learning models. From the table, we can see that each instantiation comprises two components - an auto-encoder and a classifier, e.g., `1dCNN` consists of `1dCNN-AE` and `1dCNN-CLF`. Besides, the model extension `1dCNN-C` contains only one component of composite auto-encoder. For each instantiation, we apply a parameter search for the learning rate between 0.001 and 0.0001, and then report the model metrics with the best validity.

Note that we have two slightly different structures (*shallow* and *deep*) for classifiers `1dCNN-CLF` and `LSTM-CLF`, due to different timestep sizes and varied amounts of available training data. During the training, either a shallow or a deep classifier is trained for both CNN and LSTM instantiations. We then evaluate on each specific dataset using the model with the best performance. Empirically, this strategy generalizes well for all datasets in our experiment.

CNN Models. For the detailed architecture of `1dCNN-AE`, the network contains four convolutional layers with 64, 32, 32 and 64 filters with kernel size 3 and activated using ReLU. The network's output consists of a final convolutional layer, which is corresponding to the reconstructed output. The first two layers are followed by a max pooling operation; while the next two layers are followed by

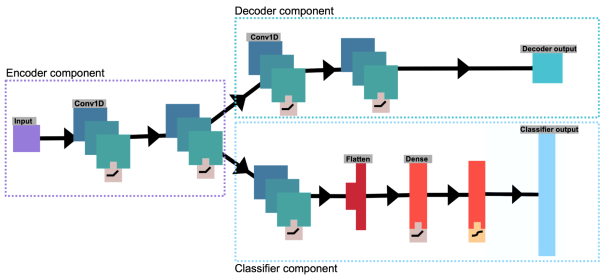


Fig. 2. Illustration of 1dCNN composite model architecture.

up-sampling transformations of size 2. The *deep* `1dCNN-CLF` model is composed of three convolutional layers followed by a final dense layer with sigmoid activation. For each convolutional layer, the number of filters is fixed to 64 and the kernel size is set to 3, with ReLU activation and batch normalization. Moreover, the *shallow* model contains only one convolutional layer compared to three in the deep model and finally a 128-unit dense layer after the input.

LSTM Models. `LSTM-AE` consists of four consecutive LSTM layers with respectively 64, 32, 32 and 64 units with tanh activation functions. The final layer is a dense layer with a sigmoid activation function. The *shallow* model for `LSTM-CLF` contains one 32-unit LSTM layer with tanh activation; while the *deep* model comprises two continuous LSTM layers with 64 and 16 units, respectively. Each LSTM layer is followed by a batch normalization operation. Finally, a dense layer with sigmoid activation function is fully connected to the previous layer, where the output represents the probability of the target prediction.

LatentCF++ Extension: Composite Model. In addition to the two instantiations, we intend to evaluate a model extension of `LatentCF++` with a CNN composite auto-encoder `1dCNN-C`. The model has a unique architecture compared to the previously described models, which is shown in Fig. 2. It has three elements: an *encoder* for encoding the input into the latent space, followed by a *decoder* and a *classifier*. Accordingly, it contains two different output layers in the architecture: one convolutional layer as the decoder and a sigmoid function for the classifier. The encoding and decoding components share the same setup of layers as `1dCNN-AE`: the encoding component has two convolutional layers with 64 and 32 filters followed by max pooling; the decoder comprises three convolutional layers with 32, 64 and 1 filters, respectively. While for the component of classifier, a convolutional layer (16 filters with kernel size 3) followed by a 50-unit dense layer are connected after the shared latent space; a final sigmoid dense layer is fully connected with the previous layer for output probability of the target class. In the implementation of `LatentCF++`, we directly apply the `Predict()` function on latent representation z to adjust for the `1dCNN-C` model, instead of using the `Decode()` function.

Implementation Details. All deep learning models are implemented in Keras[1]. For `1dCNN-AE`, `LSTM-AE`, and `1dCNN-C`, we set the training epochs to 50; while for classification models, the training epochs are set to 150 for both `1dCNN-CLF` and `LSTM-CLF`. To reduce overfitting and improve the generalizability of our networks, we employ early stopping during model training. Adam optimizer is employed for all networks with learning rates ranging from 0.0001 and 0.001 for different models. The batch size is set to 32 for all of them.

4.2 Baseline Models

We adopt the first baseline model from the original paper, `FGD` [3], which applies the `LatentCF` method with only a classifier `1dCNN-CLF` to perturb samples in the original feature space directly. In addition, we apply two extra baseline methods from local and global time series tweaking approaches [11] - random shapelet forest (`RSF`) and the k-NN counterfactual method (`k-NN`). Similar to the evaluation of `LatentCF` and `LatentCF++`, we apply the same parameter setup across all datasets for these baseline models. For `RSF`, we set the number of estimators to 50 and max depth to 5; while the other parameters are kept at their default values[2]. For `k-NN`, we first train a k-NN classifier with k equals to 5 and the distance metric set to Euclidean; then the trained classifier is utilized to find counterfactual samples for further evaluation.

4.3 Evaluation Metrics

To evaluate the performance of our proposed approach in terms of explainability, we present three metrics: validity, proximity and closeness. *Validity* is defined to measure whether the generated counterfactual explanations lead to a valid transformation of the desired target class [14,17]. More specifically, it reports the fraction of counterfactuals predicted as the opposite class (i.e. have crossed the decision boundary τ) according to the standalone classifier. It is defined as:

$$validity(y_{cf}, \tau) = \frac{\#(y_i \geq \tau, y_i \in y_{cf})}{\# y_{cf}}, \tag{1}$$

where y_{cf} is the output probability of all the generated counterfactual samples, and τ is a user-defined threshold of the decision boundary. In our evaluation, we set the threshold τ to be 0.5.

Proximity is applied to measure the feature-wise distance between generated counterfactuals and corresponding original samples [14]. Karlsson et al. [11] reported a similar metric named *cost* in the evaluation of local and global time series tweaking approaches. In our case, we define proximity as Euclidean distance between transformed and original time series samples:

$$proximity(x, x') = d(x, x'), \tag{2}$$

[1] https://keras.io.

[2] See https://github.com/isaksamsten/wildboar.

where $d(\cdot)$ is the Euclidean distance and x and x' are the original time series and generated counterfactual sample respectively. We report the average value of the proximity scores for each dataset.

To measure how close is the predicted probability of a generated counterfactual compared to the decision boundary, we propose a new metric which we denote as *margin difference*. The margin difference captures the amount of information that has been altered from the original class and is defined as:

$$margin\_diff(y_{cf}, \tau) = y_{cf} - \tau, \tag{3}$$

where y_{cf} is the output probability of counterfactual x', and τ is the target decision boundary. Note that this metric can be either positive or negative, indicating whether the counterfactual has crossed the decision boundary or not. We record both the mean and standard deviation of margin differences for all generated counterfactuals as metrics for each dataset.

In addition, we show the classification performance of all models report as the **balanced accuracy** in the results. Moreover, we report the reconstruction loss of the auto-encoder models.

4.4 Results

In this section, we first compare the validity of our presented explainable models from `LatentCF` and `LatentCF++`, as well as `FGD`, `RSF` and `k-NN` counterfactual methods. For a detailed comparison, we choose to report metrics from a subset of 20 datasets with the sample size larger than 500. Then we report a subset average for different explainable models; together, we also present the average score (denoted as **Total avg.**) across all 40 datasets in the experiment[3].

Table 3 shows the *validity*, which we considered as the main metric for the evaluation of interpretability. Again, we report the results with the value of decision boundary $\tau = 0.5$ in Eq. 1. Across three different groups, we found that the `RSF` method achieved the best metric of subset average of 1.0000; in contrast, `1dCNN` from the `LatentCF++` method obtained the highest validity (0.9920) in terms of the total average, which indicates that an average of 99.20% of the generated counterfactuals is considered valid. In comparison, `1dCNN` from the `LatentCF` method and `FGD` baseline both received validity that was lower than 0.2. This evidence suggests that our proposed `LatentCF++` method can ensure a high fraction of valid counterfactual transformations similar to `RSF`.

In Table 4, we show a comparison for individual mean scores of *margin difference* for the subset of 20 datasets, together with the average score at the bottom. In addition, we report the average of standard deviations for both the subset (**Subset std.**) and the total (**Total std.**). From the table, we observed that `1dCNN` from `LatentCF++` achieved the best metric to both subset average (0.0333) and total average (0.0168), which indicates that generated counterfactuals are considerably closer to the decision boundary compared to other methods. In terms of the total average, `LSTM` models from `LatentCF` and `LatentCF++`

[3] The full result table is available at our supporting website.

Table 3. Summary of validity for a subset of 20 different datasets in our experiment. The best score for each dataset is highlighted in bold.

Dataset	LatentCF++			LatentCF		Baseline		
	`1dCNN`	`LSTM`	`1dCNN-C`	`1dCNN`	`LSTM`	`FGD`	`k-NN`	`RSF`
Yoga	0.9973	**1.0000**	0.9912	0.4347	**1.0000**	0.4507	**1.0000**	**1.0000**
TwoLeadECG	0.9914	0.9052	0.9835	0.3966	**1.0000**	0.0000	**1.0000**	**1.0000**
ItalyPower	**1.0000**	**1.0000**	0.9912	0.0000	0.2456	0.0000	**1.0000**	**1.0000**
MoteStrain	**1.0000**	**1.0000**	0.9918	0.2017	0.0455	0.0000	**1.0000**	**1.0000**
Wafer	**1.0000**	0.8625	0.8117	0.2353	0.1875	0.0588	**1.0000**	**1.0000**
FreezerRegular	**1.0000**	**1.0000**	0.9666	0.5633	**1.0000**	0.4967	**1.0000**	**1.0000**
PhalangesOutlines	**1.0000**	**1.0000**	**1.0000**	0.3833	0.7048	0.2333	0.8629	**1.0000**
FreezerSmall	0.9791	**1.0000**	0.9790	0.6794	**1.0000**	0.5122	**1.0000**	**1.0000**
HandOutlines	0.9091	0.9661	**1.0000**	0.2727	0.0000	0.0000	0.9359	**1.0000**
FordA	**1.0000**	0.0000	0.9838	0.0055	0.0000	0.0000	0.8903	**1.0000**
FordB	**1.0000**	**1.0000**	**1.0000**	0.0027	**1.0000**	0.0000	0.9324	**1.0000**
SonyAIBORobot2	0.9916	0.8560	0.9917	0.0168	**1.0000**	0.0000	**1.0000**	**1.0000**
SemgHandGender	0.8358	**1.0000**	0.8310	0.5970	**1.0000**	0.6567	0.6528	**1.0000**
MiddlePhalanx	**1.0000**	**1.0000**	**1.0000**	0.1111	**1.0000**	0.0000	0.8431	**1.0000**
ProximalPhalanx	**1.0000**	**1.0000**	**1.0000**	0.0000	0.1282	0.0000	0.6842	**1.0000**
ECGFiveDays	0.9773	0.8864	0.9773	0.5114	**1.0000**	0.0000	**1.0000**	**1.0000**
DistalPhalanx	**1.0000**	**1.0000**	**1.0000**	0.0769	0.3036	0.0000	**1.0000**	**1.0000**
SonyAIBORobot1	**1.0000**	**1.0000**	0.9818	0.0182	**1.0000**	0.0000	**1.0000**	**1.0000**
Computers	**1.0000**	0.3824	**1.0000**	0.4839	0.5000	0.3548	0.5593	**1.0000**
Strawberry	**1.0000**	**1.0000**	0.9600	0.4146	0.0000	0.3333	**1.0000**	**1.0000**
Subset avg.	0.9841	0.8929	0.9720	0.2703	0.6058	0.1548	0.9180	**1.0000**
Total avg.	**0.9920**	0.8256	0.9615	0.1676	0.5779	0.0774	0.9496	0.9802

achieved the second and third best margin difference scores of 0.0520 and 0.0580, respectively. In comparison, `RSF` model received a less favourable metric of margin difference (0.0614) since it does not optimize towards the threshold. Instead, it tries to minimize the difference between the original samples and counterfactuals. `1dCNN-C`, `FGD` and `1dCNN` from `LatentCF` received negative scores according to total average, which means that they could not constantly guarantee counterfactuals to cross the decision boundary. Nonetheless, `1dCNN-C` outperformed all the other models in 10 out of the 20 individual datasets.

In addition, we observed that the standard deviation for `1dCNN` from the `LatentCF++` framework resulted in the lowest of 0.0175 while `k-NN` achieved the maximum of 0.1218 in Table 4. Thus, this evidence shows that `LatentCF++` can generate counterfactuals that are more stable in terms of margin difference. In other words, our proposed approach can better guarantee to produce counterfactuals that can cross the decision boundary and obtain more compact transformations.

From the previous comparison of validity, we found that several models received scores lower than 0.2, which means that these models cannot guarantee the fraction of valid counterfactuals. As our primary evaluation was based on the validity, we intended to compare the rest of the five best-performed models for

Table 4. Summary of margin difference for a subset of 20 different datasets in our experiment. The best score for each dataset is highlighted in bold.

Dataset	LatentCF++			LatentCF		Baseline		
	`1dCNN`	`LSTM`	`1dCNN-C`	`1dCNN`	`LSTM`	`FGD`	`k-NN`	`RSF`
Yoga	**0.0005**	0.1810	–0.0003	–0.0383	0.3115	–0.0531	0.4096	0.0199
TwoLeadECG	0.0005	**0.0002**	0.0006	–0.0347	0.4941	–0.1854	0.4914	0.1936
ItalyPower	**0.0005**	**0.0005**	0.0006	–0.3020	–0.0854	–0.4144	0.5000	0.2077
MoteStrain	0.0005	0.0005	**0.0004**	–0.0343	-0.0261	–0.3766	0.4704	0.0382
Wafer	**0.0005**	0.0034	–0.0486	–0.2831	–0.3181	–0.4545	0.5000	0.0334
FreezerRegular	**0.0005**	0.4995	–0.0028	0.0614	0.4999	0.0016	0.3916	0.1278
PhalangesOutlines	**0.0005**	**0.0005**	**0.0005**	–0.0004	–0.0002	–0.0044	0.2783	0.0573
FreezerSmall	0.4774	0.4636	–0.0081	0.1878	0.4766	**0.0156**	0.3683	0.1605
HandOutlines	–0.0220	0.0061	**0.0005**	–0.2385	–0.5000	–0.3531	0.4051	0.0379
FordA	0.0005	–0.0382	**0.0002**	–0.1302	–0.0568	–0.1816	0.2363	0.0196
FordB	**0.0005**	0.1443	**0.0005**	–0.2152	0.2182	–0.2709	0.2551	0.0198
SonyAIBORobot2	**0.0005**	0.0056	**0.0005**	–0.1871	0.5000	–0.4431	0.3413	0.0311
SemgHandGender	0.2016	0.1822	–0.0416	0.0863	0.1908	0.1640	0.1556	**0.0238**
MiddlePhalanx	0.0005	0.0005	**0.0001**	–0.0084	0.0030	–0.0088	0.2255	0.1015
ProximalPhalanx	0.0005	**0.0004**	0.0005	–0.2328	–0.0026	–0.0963	0.2088	0.1905
ECGFiveDays	0.0005	**0.0003**	0.0004	0.0018	0.4985	–0.4627	0.4934	0.0786
DistalPhalanx	0.0006	0.0005	**0.0004**	–0.1098	–0.0009	–0.1899	0.3151	0.0932
SonyAIBORobot1	**0.0004**	0.0005	**0.0004**	–0.2647	0.4795	–0.4233	0.4690	0.0425
Computers	0.0005	0.0203	**0.0001**	–0.0082	0.1415	–0.0772	0.0864	0.0202
Strawberry	**0.0005**	**0.0005**	–0.0028	–0.1076	–0.4322	–0.1311	0.4587	0.1245
Subset avg.	**0.0333**	0.0736	–0.0049	–0.0929	0.1196	–0.1973	0.3530	0.0811
Subset std.	0.0348	**0.0246**	0.0287	0.2072	0.0453	0.1848	0.1600	0.0623
Total avg.	**0.0168**	0.0580	–0.0120	–0.0896	0.0520	–0.1435	0.3608	0.0614
Total std.	**0.0175**	0.0234	0.0310	0.1193	0.0240	0.1081	0.1218	0.0576

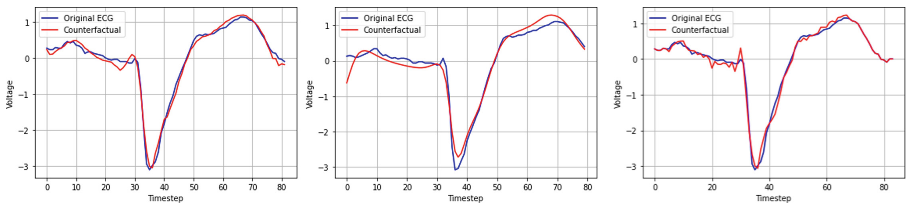
further investigation. Namely, we chose to exclude `1dCNN`, `LSTM` from `LatentCF` and `FGD` from baseline models in our comparison of proximity. Similar to the previous evaluation, we reported metrics from the subset of 20 datasets with respective subset and total average scores, as in Table 5.

In Table 5, we observed that `1dCNN` from our proposed `LatentCF++` framework achieved comparable proximity scores compared to the state-of-the-art method `RSF`, with a subset average of 0.3891 in comparison with 0.2873. This evidence indicates that the generated counterfactuals from `1dCNN` are relatively closer to the original samples. In contrast, `LSTM` received the worst average proximity of 2.5409 among all. When we checked the individual results, we found that both `1dCNN` and `1dCNN-C` from `LatentCF++` outperformed `RSF` in 9 out of 20 datasets, while `RSF` outperformed the others in 10 datasets. One of our key observations was that the proximity score was strongly related to the corresponding auto-encoder performance. Usually, if the `1dCNN-AE` model from `1dCNN` converged with a low reconstruction loss, then `1dCNN` would outperform the other methods. However, it was challenging to ensure the performance of the auto-encoder since we applied a unified structure of `1dCNN-AE` for all datasets.

Table 5. Summary of proximity for a subset of 20 different datasets in our experiment. The best score for each dataset is highlighted in bold (three methods are excluded from the comparison due to the low validity[†]).

	LatentCF++			LatentCF		Baseline		
Dataset	`1dCNN`	`LSTM`	`1dCNN-C`	`1dCNN`[†]	`LSTM`[†]	`FGD`[†]	`k-NN`	`RSF`
Yoga	**0.2049**	3.6447	0.7221	0.1759	3.6347	0.0277	1.0950	0.5281
TwoLeadECG	**0.1447**	1.4290	0.1839	0.1659	0.5681	0.0270	0.2655	0.1793
ItalyPower	0.4785	0.3986	0.2588	0.1609	0.3373	0.0066	0.3633	**0.2513**
MoteStrain	0.3167	0.4798	0.3884	0.2652	0.4798	0.0094	0.4673	**0.1503**
Wafer	**0.2157**	0.4416	0.2325	0.1062	0.3642	0.0069	0.5790	0.2547
FreezerRegular	0.2211	1.0497	0.1158	0.1812	0.6700	0.0409	0.0808	**0.0569**
PhalangesOutlines	**0.1449**	0.4592	0.2109	0.1060	0.5168	0.0190	0.2272	0.2238
FreezerSmall	1.9710	0.6919	0.1929	0.1978	0.5990	0.0319	0.0887	**0.0598**
HandOutlines	**0.4359**	23.9995	3.9276	0.2575	7.9567	0.0087	1.9754	1.1485
FordA	0.5670	2.3575	**0.4368**	0.2463	2.3087	0.0159	2.0811	0.4820
FordB	0.6099	2.2574	0.4261	0.2176	2.2092	0.0124	2.1105	**0.3934**
SonyAIBORobot2	0.3234	0.9515	0.3263	0.2293	0.9466	0.0054	0.6357	**0.2862**
SemgHandGender	0.4159	0.8741	0.2637	0.2676	0.8786	0.0351	0.2999	**0.1006**
MiddlePhalanx	**0.1749**	0.6519	0.3936	0.1496	0.6786	0.0000	0.2646	0.2568
ProximalPhalanx	0.2633	0.5222	0.2933	0.1359	0.5314	0.0145	**0.1577**	0.2539
ECGFiveDays	0.1654	0.9480	0.2829	0.1538	0.8912	0.0057	0.3459	**0.1083**
DistalPhalanx	0.2258	0.7235	**0.1850**	0.1450	0.7282	0.0133	0.3674	0.2724
SonyAIBORobot1	0.2719	0.5017	0.2440	0.1882	0.4951	0.0078	0.5752	**0.2260**
Computers	0.3759	1.3727	0.7971	0.3538	1.3243	0.0236	0.8305	**0.1774**
Strawberry	**0.2545**	1.1272	0.2947	0.2242	0.7843	0.0255	0.3734	0.3370
Subset avg.	0.3891	2.2441	0.5088	0.1963	1.3451	0.0169	0.6592	**0.2873**
Total avg.	0.8926	2.5409	0.9179	0.4415	1.9841	0.0087	1.4613	**0.5241**

For a detailed comparison, we investigated ECG examples of generated counterfactuals by `1dCNN`, `LSTM` from `LatentCF++`, and `RSF` (see Fig. 3). We observed that `1dCNN` and `RSF` generated similar counterfactuals for the ECG sample, although the counterfactual from `1dCNN` appeared slightly smoother in this case. In contrast, `LSTM`'s counterfactual poorly aligned with the original series and diverged in many timesteps. Moreover, we found that different classification models performed similarly in balanced accuracy. As to the auto-encoders, `LSTM-AE` achieved the highest reconstruction loss over most datasets while `1dCNN-AE` received the lowest reconstruction loss. This evidence explains why `LSTM` from `LatentCF` and `LatentCF++` attained the worst performance when comparing proximity. It appeared that the `LSTM` auto-encoder could not learn a representative latent space compared to other auto-encoders in the time series domain.



(a) `1dCNN` (b) `LSTM` (c) `RSF`

Fig. 3. Examples of generated counterfactuals by `1dCNN` and `LSTM` instantiations from LatentCF++, together with `RSF` in comparison. Illustrated in blue are the original time series and in red the generated counterfactuals of the opposite class. (Color figure online)

5 Conclusions

We presented a new counterfactual explanation framework named `LatentCF++` for time series counterfactual generation. Our experimental results on the UCR archive focusing on binary classification showed that `LatentCF++` substantially outperforms instantiations of its predecessor, `LatentCF`, and other baseline models. The results also suggest that `LatentCF++` can provide robust counterfactuals that firmly guarantee validity and are considerably closer margin difference to the decision boundary. Additionally, our proposed approach achieved comparable proximity compared to the state-of-the-art time series tweaking approach `RSF`. Furthermore, we found that it was challenging to leverage the power of deep learning models (both classifiers and auto-encoders) for datasets with the sample size of less than 500. Hence our one-size-fits-all plan to utilize unified structures of deep learning models as components for the framework did not address some specific datasets in the experiment. However, we still showed the generalizability of our proposed framework using a unified set of model components. For future work, we plan to extend our work to generalize `LatentCF++` to broader counterfactual problems using other types of data, e.g. multivariate time series, textual or tabular data. Also, we intend to conduct a qualitative analysis from domain experts to validate that the produced counterfactuals are meaningful in different application fields, such as ECG measurements in healthcare or sensor data from signal processing.

Reproducibility. All our code to reproduce the experiments is publicly available at our supporting website[4].

Acknowledgments. This work was supported by the EXTREMUM collaborative project (https://datascience.dsv.su.se/projects/extremum.html) funded by Digital Futures.

[4] https://github.com/zhendong3wang/learning-time-series-counterfactuals.

References

1. Ates, E., Aksar, B., Leung, V.J., Coskun, A.K.: Counterfactual Explanations for Machine Learning on Multivariate Time Series Data. arXiv:2008.10781 [cs, stat] (August 2020)
2. Bagnall, A., Lines, J., Bostrom, A., Large, J., Keogh, E.: The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min. Knowl. Disc. **31**(3), 606–660 (2016)
3. Balasubramanian, R., Sharpe, S., Barr, B., Wittenbach, J., Bruss, C.B.: Latent-CF: a simple baseline for Reverse Counterfactual Explanations. In: NeurIPS 2020 Workshop on Fair AI in Finance (December 2020)
4. Dau, H.A., et al.: Hexagon-ML: The ucr time series classification archive (October 2018). https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
5. Dempster, A., Petitjean, F., Webb, G.I.: ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. Data Min. Knowl. Disc. **34**(5), 1454–1495 (2020)
6. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.-A.: Deep learning for time series classification: a review. Data Min. Knowl. Disc. **33**(4), 917–963 (2019)
7. smail Fawaz, H., et al.: InceptionTime: finding AlexNet for time series classification. Data Min. Knowl. Disc. **34**(6), 1936–1962 (2020)
8. Joshi, S., Koyejo, O., Vijitbenjaronk, W., Kim, B., Ghosh, J.: Towards Realistic Individual Recourse and Actionable Explanations in Black-Box Decision Making Systems. arXiv: 1907.09615 (July 2019)
9. Kampouraki, A., Manis, G., Nikou, C.: Heartbeat time series classification with support vector machines. IEEE Trans. Inf Technol. Biomed. **13**(4), 512–518 (2009)
10. Karlsson, I., Papapetrou, P., Boström, H.: Generalized random shapelet forests. Data Min. Knowl. Disc. **30**(5), 1053–1085 (2016)
11. Karlsson, I., Rebane, J., Papapetrou, P., Gionis, A.: Locally and globally explainable time series tweaking. Knowl. Inf. Syst. **62**(5), 1671–1700 (2019)
12. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015) (January 2015)
13. Molnar, C.: Interpretable Machine Learning - A Guide for Making Black Box Models Explainable (2019)
14. Mothilal, R.K., Sharma, A., Tan, C.: Explaining machine learning classifiers through diverse counterfactual explanations. In: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, pp. 607–617 (January 2020)
15. Pawelczyk, M., Haug, J., Broelemann, K., Kasneci, G.: Learning model-agnostic counterfactual explanations for tabular data. In: Proceedings of The Web Conference, vol. 2020, pp. 3126–3132 (2020)
16. Rebbapragada, U., Protopapas, P., Brodley, C.E., Alcock, C.: Finding anomalous periodic time series. Mach. Learn **74**(3), 281–313 (2009)
17. Stepin, I., Alonso, J.M., Catala, A., Pereira-Fariña, M.: A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. IEEE Access **9**, 11974–12001 (2021)

18. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: automated decisions and the GDPR. SSRN Electron. J. (2017)
19. Yao, S., Hu, S., Zhao, Y., Zhang, A., Abdelzaher, T.: DeepSense: a unified Deep Learning Framework for Time-Series Mobile Sensing Data Processing. In: Proceedings of the 26th International Conference on World Wide Web. pp. 351–360 (April 2017)