Interpretable Latent Space to Enable Counterfactual Explanations

Francesco Bodria[1], Riccardo Guidotti[2], Fosca Giannotti[1], and Dino Pedreschi[2]

[1] Scuola Normale Superiore, Piazza dei Cavalieri, 7, Pisa, Italy
{name.surname}@sns.it
[2] University of Pisa, Largo Bruno Pontecorvo, 3, Pisa, Italy
{name.surname}@unipi.it

Abstract. Many dimensionality reduction methods have been introduced to map a data space into one with fewer features and enhance machine learning models' capabilities. This reduced space, called latent space, holds properties that allow researchers to understand the data better and produce better models. This work proposes an interpretable latent space that preserves the similarity of data points and supports a new way of learning a classification model that allows prediction and explanation through counterfactual examples. We demonstrate with extensive experiments the effectiveness of the latent space with respect to different metrics in comparison with several competitors, as well as the quality of the achieved counterfactual explanations.

1 Introduction

The booming research in eXplainable AI (XAI) of recent years has focused mainly on post-hoc explanation [12, 30], or how to add a transparency layer on top of an opaque machine learning model [16]. Post-hoc explanation methods have several shortcomings, including robustness and trustworthiness of the explanations [8, 13]. An emerging, more ambitious objective is to define novel Machine Learning (ML) methodologies to construct models that are *transparent-by-design*, i.e., models that natively deliver accurate classifications together with trustworthy explanations [31].

In this paper we propose a new approach to perform classification, named *ILS* for *Interpretable Latent Space*. ILS foresees the simultaneous construction of an Interpretable Latent Space and of a classifier trained on such latent space in the training phase. The latent space is built to obtain classification and retrieve explanations simultaneously. ILS uses a similarity loss to transform data from the real space to the latent space using a linear model. Then, from this latent space, a counterfactual explanation is extracted. We show how this approach enables a new use of the learned model such that, when applied to an instance x, besides the prediction, it also returns a *counterfactual example*, i.e., another instance x' with minimal changes to the features of x that is classified differently. For example, consider a ML model for credit approval and an application x classified as "declined". Our proposal also returns a counterfactual instance [11]

illustrating the minimal changes to x needed to obtain credit approval, e.g., *(Hours Per Week ! 53.5, occupation ! Prof Specialty)*.

Thus, the main contribution of this paper is twofold. First, an interpretable latent space is defined based on a linear encoding of the original data space. Second, we show how the newly defined interpretable latent space properties allow us to find a counterfactual example. We extensively evaluate our proposal with various tabular and image datasets. First, we observe that the interpretable latent space actually preserves similarities better than other approaches in the literature [3, 29]. Second, we assess qualitatively and quantitatively the counterfactuals provided by our method compared to others.

The rest of the paper is organized as follows. In Section 2, we analyze the existing works about latent space creation. After that, Section 3 illustrate the proposed methodology. Then, Section 4 reports the experimental results comparing our proposal against state-of-the-art methods. Conclusions and future research directions are discussed in Section 5.

2 Related Works

Latent Space Models. When it comes to compressing high dimensional data into a lower dimensional space, several options are available: dimensionality reduction methods that create features with linear combinations of original ones, and generative models that incorporate non-linear relationships. Principle Component Analysis (PCA) [3] is the most famous dimensionality reduction technique. It is defined as an orthogonal linear transformation that maps the data into a new coordinate system, reducing variance. Uniform Manifold Approximation and Projection (UMAP) [23] uses graph layout algorithms to arrange data in low-dimensional space and preserve the local structure of the data. TriMap [5] tries to balance the importance of the local and the global structure of points in the created latent space using triplet constraints of the kind: "point i is closer to point j than point k." Generative models are the most recent approaches to the problem of latent space creation. The most common architecture is called AutoEncoder [26], and it is composed of: an encoder that encodes the data into a reduced representation called latent space and a decoder that decodes the data from the latent space back to its original space. Both the encoder and the decoder are Neural Networks, and the training is performed by minimizing the reconstruction loss between the original data and the one generated by the decoder. Several variations have been proposed for years to improve data representation in the latent space. Variational Auto-Encoders (VAE) [29] improve the representation in the latent space by adding a constraint on the encoding network that forces it to generate latent vectors that follow a Gaussian unit distribution. The problem with autoencoders is the amount of parameters not trivial to select during training.

Latent Space Applications. Latent spaces are largely used in several domains ranging from healthcare [1, 24, 32], social network analysis [10, 15] and anomaly detection [6, 28]. Several studies have demonstrated that the model's latent space

Algorithm 1: $ILS(x, X, Y, K, f)$

```
Input  : x - instance to classify and explain X - training data, Y - labels, K - list of
         latent space dimensions, f - classifier training function
Output: y - classification, x^0 - counterfactual explanation
Train(X, Y, f, K):
1 M ← LearnBestLatentSpace(X, f, K);          //find best latent space
2 Z ← M(X);                                   //turn training data into latent space
3 b ← f(Z, Y);                                //train classifier on the latent space
4 return b, M;
Predict and Explain(x, b, Z, M):
5 z ← M(x);                                   //get latent representation
6 ŷ ← b(z);                                   //apply prediction
7 x^0 ← GetCounterfactual(z, b, Z, M)         //get counterfactual explanation
8 return ŷ, x^0;
```

can hold better clustering performances than the original feature space [25, 27, 39]. Other works have shown that classifiers benefit from latent spaces [40, 41].

Interpretation of the latent space. The latent space is always created using a generative model, and post-hoc analysis is performed to give insights into the dimension created. The most popular technique is to apply vector transformation in the latent space in order to control the generation of the data [2, 33, 38]. Style transfer is another way to use the latent space, the idea is to transfer the style of a data to another one with minimal changes [14]. Recent approaches have tried to explore the possibility of building a transparent latent space. The exploration is done by moving into the latent space and using the transparent proprieties to analyze the reconstructed data [17, 37].

3 Methodology

We introduce here the Interpretable Latent Space (ILS) method, and we show how it is able to return an explanation in the form of counterfactual instances besides the classification outcome. The idea of ILS is to create an interpretable latent space in which the position of a point can be explained exactly in terms of input characteristics. We claim that our space is interpretable since the linear mapping of the input features can be represented in the latent space in the form of vectors (Figure 1 (left)). Transparency can be exploited to obtain the counterfactual explanation of a classifier trained in such a space. Like most classification methods, ILS has a *train phase* and a *predict phase*. A novelty is that the latter is indeed a *predict & explain phase*.

The whole procedure, illustrated in Algorithm 1, starts by learning the latent space model M from the training set X, and with reference to a varying number of latent dimensions k chosen among the list of dimensions K (line 1). With M, we indicate a model able to turn the input dataset $X \in \mathbb{R}^n$ into a latent version $Z \in \mathbb{R}^k$. Details about the latent space learning are discussed and formalized in Section 3.1. Then, M is applied to X to obtain the latent representation of the dataset $Z \in \mathbb{R}^k$ (line 2). Finally, a classifier b is trained on Z through the training function f (line 3). After the latent space training, the predict

Algorithm 2: $LearnBestLatentSpace(X, K, f)$

```
Input  : X - training data, K - list of latent space dimensions, f - classifier training
         function
Output: M - Trained model
1  M ← ∅;                                                  //empty best transformation model
2  s ← 0;                                                  //init. best model score
3  for k ∈ K do
4  |  M^0 ← LearnLatentSpace(X, k);                       //learn latent space with k dimensions
5  |  Z ← M^0(X);                                         //get latent representation
6  |  b ← f(Z, Y)                                         //train classifier on Z
7  |  s^0 ← evaluate(b(Z), Y);                           //evaluate classification performance
8  |  if s < s^0 then
9  |  |  M ← M^0;                                       //take best transformer w.r.t classifier performance
10 return M;
```

& explain procedure works as follows. Given an instance x, x is turned into its latent representation z, and the classifier is applied to obtain its prediction (lines 4-5). Then, the prediction $\hat{y}$ is explained by exploiting the interpretability of the learned space, returning a counterfactual instance x'. Details on how the counterfactual is constructed are given in Section 3.2.

3.1 Interpretable Latent Space Learning

ILS is based on a linear transformation that enables the transparent mapping between the input and latent features. The idea is to create/learn a latent space by combining a similarity loss analogous to the one utilized in t-SNE [22] with the mapping reasoning of PCA: the data are mapped into the space based on the similarity between them. In addition to PCA and t-SNE, latent space is also created using black-box predictions, augmenting data information. Our objective is that *similar instances in the original input space should be close also in the latent space that we are trying to build.* Linear models have been proven in recent years [21] to be the best methodology to produce explanations, in the sense that it is possible to isolate the contribution of each feature to the prediction. In line with these insights, we propose a procedure to build an interpretable latent space using a linear mapping $\boldsymbol{M}$ that transforms the input space X of dimension n into a latent space of dimension k, i.e., $Z = \boldsymbol{M}(X)$ such that $z_j = w_0x_0 + w_1x_1 + \quad + w_ix_i + \quad + w_nx_n$, where w are the weights of the model $\boldsymbol{M}$, x is an instance belonging to the input space R^n, z its transformation to the latent space R^k, and k is the number of latent dimensions. In the literature, the k parameter is challenging to select and is usually provided heuristically. Hence, the objective of *LearnLatentSpace*, described by Algorithm 3, is to find the "best" weights w for the linear model $\boldsymbol{M}$, given a specific dimension k.

LearnLatentSpace starts by initializing the model $\boldsymbol{M}$ with random weights (line 3) and fitting it to the data X. Thus, we use gradient optimization [18] to minimize an unsupervised loss that encourages similarity between near points. We adopt the similarity probability loss introduced in [22], with a different purpose: instead of using the similarity loss for visualizing a space in two dimensions, we use it to create a new data space that enjoys the requested similarity property.

Algorithm 3: $LearnLatentSpace(X, k)$

```
Input  : X - training data, k - latent space dimension
Output: M - trained transformation model
1  i ← 0;                                              //init. iteration index
2  L_i ← ∞;                                            //initialize loss
3  M ← init();                                         //initialize model weights
4  S_X ← PairwiseSimilarity(X);                        //original similarity matrix
5  while L_{i-1} > L_i do                              //until the loss decreases
6  |  Z ← M(X)                                        //get latent representation
7  |  S_Z ← PairwiseSimilarity(Z);                     //latent similarity matrix
8  |  L_i ← KLD(S_X, S_Z);                            //compute Kullback-Leibler Divergence loss
9  |  M ← update(M, L_i);                             //Update the model using backpropagation
10 |  i ← i + 1
11 return M;
```

More in detail, the similarity of a point x_j to a point x_i is the probability that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian distribution centered at x_i. Formally, the probability is given by

$$\mathit{PairwiseSimilarity} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

where x_i and x_j are the two points, and σ is the variance of the Gaussian. Algorithm 3 computes the similarity probability of any two points x_i and x_j in the input space (lines 4 and 7). The more two points are similar, the higher this value. From a computational point of view, the issue with using this similarity loss is that it requires calculating the similarity between every pair of instances. However, since the fit is performed by gradient optimization, it is possible to divide the data into fixed-sized batches and compute the similarity matrix separately for any small batch of points. This operation can be done only once for the input data, but it needs to be repeated every time for the latent space since the position of the point in Z changes after every iteration. The two matrices must have similar distributions to enjoy the previously described similarity property. Therefore, the final loss function that we minimize is the Kullback–Leibler divergence [19] (line 8) between the matrices S_X and S_Z:

$$L(S_X, S_Z) = KL(S_X \| S_Z)$$

where S_X and S_Z are the similarity matrices computed respectively on the input space X and the latent space Z. This loss is back-propagated to update the weights w_i of the model M until convergence (line 9)[3].

The function *LearnBestLatentSpace* described by Algorithm 2 is designed to select the best space by varying the dimension of the latent space k. After initializing an empty model and setting its score to zero (lines 1-2), the following procedure is repeated for each dimension k (cycle for 3-10). Given k, it learns M', mapping the input space to a k-dimensional latent space (line 4). Next, X

[3] For the convergence problem, we used the early stopping technique.

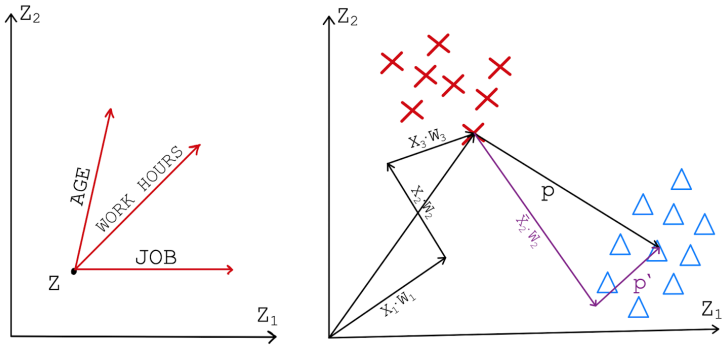

Fig. 1. Scheme of the vector model used for creating explanations. Left: representation of the input features in the latent space. Right: we illustrate a step in the ILS algorithm to modify the input features based on the position of the latent space, as explained in Section 3.2. The best update found by ILS is (X_2), p is the projector to the different class center, p^0 is the new projector for the next step

is mapped into its latent representation Z according to M', and a classifier b is trained on Z (line 6). The performance of the classifier is used to assess the goodness of M' (line 8). Finally, the latent space with the smallest size returning the highest classification performance is returned by *LearnBestLatentSpace* (lines 8-9). The *LearnBestLatentSpace* procedure is costly because it trains the classifier for every latent space dimension. We highlight that k is a crucial parameter, as its value can determine the goodness of the latent space.

3.2 Counterfactual Explanations

This section describes the methodology employed by ILS to extract counterfactual explanations exploiting the interpretability of the latent space. We refer here to binary classification, but the approach easily extends to multi-class classifiers.

Let $x = \mathbf{f}x_1, x_2, \ldots, x_n\mathbf{g}$ be an input data point for which we want to provide a prediction and its explanation. The first step of *GetCounterfactual*, illustrated in Algorithm 4 and used by ILS in line 7 of Algorithm 1, is to find the best counterfactual explanation by choosing the direction in the latent space. Given z as the latent representation of x (line 1), *GetCounterfactual* computes the position of the nearest centroid of the points in the latent space with opposite predictions of z's (line 2). This is realized using a clustering algorithm on those points in the latent space with opposite predictions with respect to z and by taking the centroid c of the cluster nearest to z (line 3). Taking the nearest sample is insufficient since we could move towards a single sample that could be wrongly classified. The direction to move in the latent space to change the outcome for z is expressed in line 7 by the projector $p = c \quad z$. The goal is to find the best feature x_i such that its new value $\bar{x}'_i$ moves the candidate counterfactual $\bar{x}$ towards the desired prediction. In particular, each input feature x_i is responsible for a

Algorithm 4: *GetCounterfactual*(x, b, Z, M)

Input : x - instance to classify and explain, b - classifier, Z - latent training set, M - latent transformation model,
Output: x^0 - counterfactual explanation

1 $z \leftarrow M(x)$; // get latent representation
2 $C \leftarrow Clustering(z^0 | z^0 \in Z_{\neq})$; // centroids with different prediction
3 $i \leftarrow \arg\min_i d(C_i, z)$ // find the centroid
4 $c \leftarrow C_i$
5 $x \leftarrow x$; // init. counterfactual
6 **while** $b(M(x)) = b(M(x))$ **do**
7 $p \leftarrow c - z$ // Find the vector projecting in the centroid direction
8 $u \leftarrow \emptyset$; // possible updates
9 **for** $i \in [1, n]$ **do**
10 $x_i^0 \leftarrow Equation2(x, i, p, M)$ //calculate update for feature i
11 $u_i \leftarrow x_i^0$; //store update
12 $i \leftarrow \arg\min_{i \in [1,n]} \{d_{euclidean}(u_i, c)\}$ // find best update
13 $x_i \leftarrow x_i^0$ // apply the best update
14 **return** x;

direction of movement in the latent space as shown in the example in Figure 1 (left)). This is repeated (lines 6-15) until the prediction for $\bar{x}$, the counterfactual candidate, is different from the prediction of the instance under analysis, i.e., until $b(M(x)) \neq b(M(\bar{x}))$.

The goal of ILS is to find the new value of x_i such that the projection p' of the instance point x' is perpendicular to the feature direction (Figure 1 (right)). More formally, this translate in $p' \cdot (W_i x'_i) = 0$ (Equation (1)), where $p' = c - z'$, z' is the position in the latent space by modifying the input feature i, and W_i is the vector of the i^{th} weight of the model M with dimension k.

$$0 = p'(W_i x'_i) = x'_i \left(\sum_{j=1}^{k} p'_j w_{ji} \right) = x'_i \sum_{j=1}^{k} (c_j - z'_j) w_{ji} = \sum_{j=1}^{k} \left(c_j - \sum_{l=1}^{n} x'_l w_{jl} \right) w_{ji} \tag{1}$$

$$\therefore x'_i = \frac{\sum_{j=1}^{k} c_j w_{ji} - \sum_{j=1}^{k} \left(\sum_{l \neq i}^{n} x_l w_{jl} \right) w_{ji}}{\sum_j w_{ji}^2} \tag{2}$$

By substituting the value of p', we obtain Equation (1). This equality would be valid only if the scalar product of p' and W_i part would be 0. Then, ILS substitutes the value of z' and extracts the x'_i value from the summation corresponding to the modification needed. The rest of the steps retrieves x'_i.

Going back to Algorithm 4, by applying the formula of Equation (2), ILS finds all the possible modifications of the i^{th} feature of x (line 10). Then, it selects the update x'_i that, if applied to $\bar{x}$ brings it to be more similar to c than

Table 1. Datasets statistics.

dataset	credit	adult	cover	clean1	clean2	isolet	madelon	sonar	soybean	anneal	mnist	fashion
instances	1,000	48,842	581,012	476	6,598	7,797	2,600	208	683	898	70,000	70,000
features	59	7	54	166	166	617	500	60	35	38	784	784
class values	2	2	7	2	2	26	2	2	19	6	10	10

the other possible updates analyzed (line 11). At this stage, the update is applied to the i^{th} feature (line 11). The procedure is iterated until a different prediction is obtained for $\bar{x}$. We underline that it is not said that the features to update are different in every iteration. Indeed, the best feature i to be updated may be the same that was already modified some iteration ago. This is due to the fact that the position in the latent space is changing at every iteration, and it is necessary for some refinement of the modification done before.

4 Experiments

We conducted two types of experiments. The first type of experiment is aimed to verify the goodness of the latent space created and compare it with other literature approaches. The second type of experiment is aimed at validating the counterfactual explanations produced.

Datasets. We ran experiments on a selection of twelve small and medium-sized datasets widely referenced for classification tasks and publicly available. Table 1 shows summary statistics on the datasets[4].

4.1 Latent Space Evaluation

First, we evaluate the quality of the latent space created by ILS. In line with [36], we used different methods, datasets, and metrics described in the following.

Competitors. We compared ILS against two categories of algorithms: autoencoders and dimensionality reduction methods. Since ILS is a hybrid approach of these two categories, we decided to include both in the experiments. The methods tested are PCA [3], UMAP [23], TMAP [5], and VAE [29], described in Section 2. For ILS, we used the Adam optimizer [18] with a learning rate of 1e-3 and a batch size of 4096. For the VAE, we trained it using early stopping of 5 for a maximum of 1000 epochs using the Adam optimizer, a learning rate of 1e-4, and a batch size of 4096. We decided to use three hidden layers with dimensions equal to the number of input features divided by 2. For PCA, UMAP and TRIMAP, we used the standard parameters. We highlight that we did not compare against t-SNE as its main goal is to define a 2d space for visualization purposes rather than a latent space to perform further mining. Also, t-SNE is rarely employed for latent dimensions higher than 3.

[4] ILS code; UCI and pytorch datasets; PCA, UMAP, and TMAP methods links.

Table 2. Space quality metrics. The best scores are in bold.

	Random Triplet Accuracy					Outlier Preservation				
Name	ILS	VAE	PCA	UMAP	TMAP	ILS	VAE	PCA	UMAP	TMAP
credit	.9525	.6731	.9525	.6958	.6803	.0000	.0000	.0000	.0000	.0109
adult	.9560	.7218	.9309	.7318	.6093	.0309	.0946	.0394	.0031	.0111
cover	.9838	.7191	.9740	.7369	.6863	.0013	.0018	.0026	.0643	.1905
clean1	.9862	.7730	.9868	.8069	.8132	.0220	.0063	.0031	.0000	.0145
clean2	.9861	.8371	.9895	.6949	.7761	.0079	.0007	.0052	.0622	.1207
isolet	.9669	.7972	.9572	.7498	.7912	.0021	.0013	.0002	.0153	.0510
madelon	.7738	.5197	.7052	.5977	.6246	.0115	.0011	.0000	.0000	.4038
sonar	.9511	.7928	.9885	.7813	.7180	.0000	.0000	.0000	.0000	.0153
soybean	.9654	.7807	.9479	.7685	.7733	.0306	.0284	.0197	.0349	.0243
anneal	.9927	.7348	.9880	.7441	.7537	.0033	.0000	.0017	.0216	.1464
mnist	.9425	.7375	.9130	.6278	.5993	.0012	.0010	.0011	.0044	.0195
fashion	.9734	.7888	.9598	.7365	.7772	.0020	.0031	.0016	.0074	.0343
wins	9	0	4	0	0	3	5	6	5	0

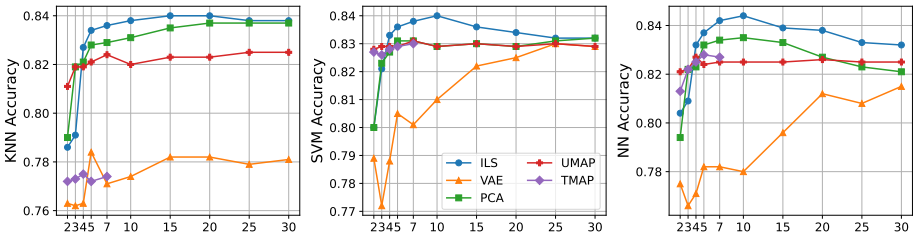

Fig. 2. Accuracy of classifiers on `adult` varying the number of latent dimensions k.

Metrics. We considered two types of evaluation metrics: *space quality metrics* and *accuracy metrics*. Space quality metrics verify different desired proprieties of the space, while accuracy metrics measure the performance of classification models trained on the latent space. To measure the relative positioning of neighborhoods, we sample observations and compute the Random Triplet Accuracy [36], which is the percentage of triplets whose relative distance order is preserved in the high and low-dimensional spaces; the closer to 1, the better. Also, we measure the outliers preservation: we want an outlier in the input space to remain an outlier in the latent space. We used the Local Outlier Algorithm (LOF) [9] to measure which points are labeled outlier or inlier in space. We ran the algorithm in both spaces to check for changes. The percentage of the changes gives the final score. We called this metric *Outlier Preservation*: the lower, the better. The classification quality of the latent space is measured using three classification models. A K-Nearest Neighbours (KNN) [35] classifier, a SVM [35] and a Neural Network (NN). For each method, we partition the embedding into five folds, each time using four folds as the training data and the remaining fold to evaluate accuracy. The metric is denoted as accuracy: the closer to 1, the better.

Results. We trained every algorithm on different latent space dimensions and evaluated the metrics for every dimension. We tested the following latent dimensions $K = \{2, 3, 4, 5, 7, 10, 15, 20, 25, 30\}$ to covert most of the possible

Table 3. Accuracy metrics. The best scores are in bold. Uncertainty is on the third decimal.

	KNN Accuracy					SVM Accuracy					NN Accuracy				
Name	ILS	VAE	PCA	UMAP	TMAP	ILS	VAE	PCA	UMAP	TMAP	ILS	VAE	PCA	UMAP	TMAP
credit	.749	.704	.745	.715	.701	.743	.710	.742	.710	.704	.742	.736	.706	.713	.699
adult	.840	.784	.837	.825	.775	.840	.830	.832	.831	.830	.844	.815	.835	.827	.828
cover	.719	.605	.722	.671	.536	.930	.921	.938	.897	.891	.836	.772	.877	.801	.776
clean1	.878	.610	.877	.770	.830	.843	.793	.840	.840	.833	.905	.588	.880	.685	.799
clean2	.934	.857	.955	.924	.883	.965	.961	.962	.955	.949	.972	.899	.988	.933	.929
isolet	.803	.579	.826	.850	.748	.879	.515	.875	.864	.853	.936	.651	.928	.834	.853
madelon	.722	.537	.795	.614	.581	.847	.542	.884	.605	.676	.753	.521	.870	.579	.668
sonar	.826	.663	.813	.791	.791	.878	.791	.842	.871	.806	.835	.748	.813	.769	.756
soybean	.888	.536	.873	.884	.873	.897	.827	.886	.902	.899	.891	.580	.895	.847	.884
anneal	.963	.769	.958	.917	.910	.985	.938	.977	.953	.948	.987	.769	.978	.889	.907
mnist	.977	.953	.928	.969	.743	.973	.973	.974	.972	.975	.973	.972	.973	.969	.976
fashion	.813	.830	.822	.811	.612	.851	.841	.855	.827	.828	.867	.866	.860	.815	.840
wins	7	1	3	1	0	7	0	3	1	1	7	0	4	0	1

dimensions while not exaggerate on computational times. For ILS, we chose the variance in the similarity loss $\sigma = 1$, since our data are normalized in the range $[-1, 1]$; different normalization may require a different value of σ. In Table 2 we report the best latent space dimension results according to space evaluation metrics. Table 2 (left) shows that ILS is the best to preserve distances, with PCA as the second best. The other approaches largely fail in preserving the original distances. This is probably due to the fact that the preservation of the distance is not explicitly minimized. Table 2 (right), shows the results of the outlier preservation metric. We do not have a clear winner with respect to this score. TMAP is significantly the worst approach[5].

In Figure 2 we report the scores of the classification models with varying latent dimension k for the `adult` dataset. Other datasets have similar behavior. Overall, increasing the latent dimensions leads to better results, although there is a sort of "magic" dimension for every dataset at which the improvement is saturated. This supports the approaches taken by most papers in literature where a fixed latent dimension is used for all the experiments. Still, it is unclear how to find this dimension without trying them all, and ILS is not an exception. Table 3 shows KNN, SVM, and NN accuracy. For KNN, ILS produces a better latent space for most of the datasets. For SVM accuracy, we observe similar results. We notice that VAE does not perform well for tabular data, but it recovers on images. UMAP is generally better than TMAP, while PCA is the second best approach.

4.2 Explanations Evaluation

In this section, we discuss the creation of counterfactual explanations through the interpretable latent space. As a classifier, we use KNN, but the same process can be directly applied to any classifier.

[5] TMAP crashed for $k > 10$ due to the exponential computational cost.

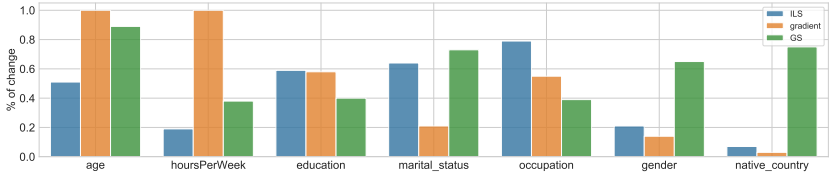

Fig. 3. Histogram of the percentage of changes in the features among the three methods. GD and GSG algorithms focus more on the first two features that are the continuous one for the `adult` dataset.

Table 4. Example of counterfactuals produced by ILS, Gradient, and GSG for `adult`.

		age	hours	education	married	occupation	gender	country
	$\boldsymbol{x_1}$	-0.315	0.000	3	1	0	1	1
$y_1 = 0$	ILS	-0.315	0.000	3	0	1	1	1
$\hat{y}_1 = 0$	GD	0.051	0.463	2	1	1	1	1
	GSG	-0.231	-0.106	3	0	0	1	1
	$\boldsymbol{x_2}$	-0.123	0.000	3	0	3	1	1
$y_3 = 1$	ILS	-0.123	0.000	3	0	1	1	1
$\hat{y}_3 = 0$	GD	-0.168	-0.044	3	0	3	1	1
	GSG	-0.121	0.001	3	0	3	1	1
	$\boldsymbol{x_3}$	-0.315	0.000	1	0	2	1	1
$y_4 = 0$	ILS	-0.315	0.000	1	1	2	1	1
$\hat{y}_4 = 1$	GD	-0.634	-0.363	1	0	2	1	1
	GSG	-0.321	0.006	1	0	2	1	1

Competitors. We compare our proposal against two model-agnostic methods that return counterfactual explanations differently. We selected these algorithms because, similarly to ILS, they are among the few model and data agnostic approaches. As a first comparison method, we search for a counterfactual of a sample x by minimizing the distance between the sample x and the centroid c of the opposite class, following the gradient descent (GD). After every gradient iteration, we modify the instance and check the prediction. We stop iterating as soon as the prediction changes. The other method is called Growing Spheres Generation [20] (GSG). The GSG procedure relies on a generative approach, growing a sphere of synthetic instances around x to find the closest counterfactual x'. Given x, GSG ignores the direction of the closest classification boundary. Indeed, GSG generates candidate counterfactuals randomly in all directions of the feature space until the decision boundary of the classifier is crossed and the closest counterfactual to x is retrieved. We selected these two approaches among the many available ones [7, 34] since they are two popular agnostic approaches to search for counterfactual explanations regardless of model and data type.

Qualitative Evaluation. We report in Tables 4 and Figure 4 the counterfactual explanations returned by ILS, GD and GSG for the `adult` and `mnist` dataset, respectively. We highlight that, for `adult`, the features age and hoursPerWeek are continuous, while the others are discrete. In datasets with mixed continuous and discrete values, we observe that GD and GSG methods tend to focus

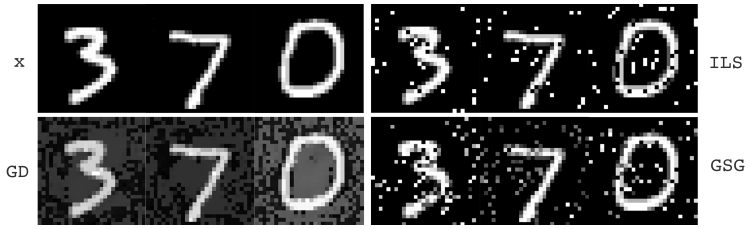

Fig. 4. Example of counterfactuals produced by ILS, GD, and GSG for the `mnist` dataset. The counterfactual classes target are 8, 9, and 8 from left to right.

Table 5. Counterfactual explanations evaluation in terms of distance and plausibility. ILS returns counterfactuals with minimal changing of input features while retaining a good result also in the distance metrics.

	d_{dist}			d_{count}			$impl$			% Success			Run Time		
	ILS	GD	GSG	ILS	GD	GSG	ILS	GD	GSG	ILS	GD	GSG	ILS	GD	GSG
adult	2.74	1.15	0.54	0.43	0.50	0.60	0.14	0.32	0.42	1.00	1.00	0.99	0.01	0.10	1.92
credit	2.94	2.86	1.40	0.11	0.99	0.39	2.62	1.84	1.40	1.00	1.00	0.98	0.03	0.20	1.75
clean1	4.16	4.09	1.16	0.09	1.00	0.19	3.99	4.01	1.16	1.00	1.00	0.72	0.04	0.26	1.33
clean2	4.09	3.32	0.45	0.06	1.00	0.18	3.73	3.31	0.45	1.00	1.00	0.11	0.21	0.51	14.2
madelon	1.96	1.02	-	0.004	1.00	-	1.95	1.02	-	1.00	1.00	0.00	0.08	0.07	-
mnist	4.65	3.11	8.13	0.03	1.00	0.17	4.65	3.11	5.30	1.00	1.00	1.00	0.18	0.24	12.6

more on the continuous features to change the prediction. For example, for x_1 of Table 4, ILS produces a counterfactual by modifying the marital status and the occupation of the person, while GD and GSG also modify the age and the hours per week. Another example that highlights this behavior is given by x_3, where ILS produces a counterfactual by only modifying the marital status while GD and GSG change the first two continuous features again. To further highlight this, we generated counterfactuals for the whole data in the `adult` dataset and checked which features were modified by the method.

In Figure 3 is reported for the percentage of modifications of each input feature. We observe that GSG and GD change the first two features (age and hoursPerWeek) considerably more times than ILS because it is easier in the real space to modify continuous variables than categorical ones to obtain the desired effect. For image datasets, such as the `mnist` examples illustrated in Figure 4, all methods resemble adversarial attacks [4] where only a few pixels are modified, and the counterfactuals found are very far from real samples in the dataset. The counterfactuals found by GD are very confused, and the modifications from the original image look like random background noise. On the other hand, ILS and GSG capture a small set of pixels that modify the attributed class.

Metrics. The quality of the found counterfactuals is evaluated with different metrics. We have chosen datasets with binary classification and a dataset on images for comparison with different data types. For `mnist`, since it is a multi-class dataset, we follow the approach proposed in [20] and search counterfactuals for a selected class. In two different fashions, we decided to measure the proximity

between x and its counterfactual $\bar{x}$. The first one, named dis_{dist}, is the average Euclidean distance between x and the counterfactual $\bar{x}$. The second measure computed, dis_{count}, quantifies the average number of features changed between a counterfactual $\bar{x}$ and x.

$$dis_{dist} = \frac{1}{jXj} \sum_{x \in X} d(x, \bar{x}) \qquad dis_{count} = \frac{1}{jXjm} \sum_{x \in X} \sum_{i=1}^{m} \mathbb{1}_{\bar{x}_i \neq x_i}$$

where $\mathbb{1}$ returns 1 if *cond* is true, 0 otherwise, and m is the number of features. Also, we measured the *implausibility* of the generated counterfactuals in terms of how close a counterfactual $\bar{x}$ is to the reference population X. It is the average distance of $\bar{x}$ from the closest instance in the X. The lower, the better.

$$impl = \frac{1}{jXj} \sum_{x \in X} \min_{\hat{x} \in X} d(\bar{x}, \hat{x})$$

We used the test set as reference population X. Finally, we computed the success rate of the algorithm to produce a counterfactual instance.

Results. The results are presented in Table 5. By modifying the instance in the latent space, our ILS method searches for counterfactuals focusing on fewer features than other methods. The counterfactuals found by ILS are more human understandable since human reasoning often involves modifying only one feature at a time. ILS as GD has a success rate of producing a counterfactual of 100% in contrast to GSG, which is not always successful. In particular, for the dataset `clean2` the success rate of GSG is lower than 10% and for `madelon` GSG completely fails. ILS is the faster method among the three to return counterfactuals. Since ILS can select the right feature to modify to change the prediction, it is faster than GSG, which has to generate many points and call the classifier for each of them to obtain a prediction. All three approaches fail to find plausible counterfactuals for images, but ILS modifies the fewest number of pixels.

5 Conclusion

In this paper, we introduce ILS, a new way of performing classification that foresees the construction of an Interpretable Latent Space for simultaneously classifying and explaining. We have compared our methods against several approaches producing similar types of latent space, demonstrating that our proposal improves with respect to the state-of-the-art because, besides interpretability, we have observed superior performance on different metrics assessing the quality of the latent space and the accuracy of classification models built on it. Also, the transparent nature of the transformation enables the interpretation of the point's position in the latent space in terms of vectors facilitating the explanations of any classifier methods built on it. We have shown how a counterfactual explanation can be produced using linear transformations and its effectiveness compared to state-of-the-art explainers.

Future directions involve studying the k parameter and experimenting with neuroevolution or other types of AutoML. Other future research directions include studying more sophisticated counterfactual explanations that may be enabled by the properties of the vector space, including intentional representations and interactive, visual exploration of the counterfactual explanations by the user beyond the first one. Moreover, by leveraging the fact that the trained ILS model is linear, it is possible to extract the saliency values for a given sample or the whole model from the weights. Additionally, the possibility of using the dataset labels to produce a better latent space should be explored.

Acknowledgment. This work has been partially supported by the European Community Horizon 2020 program under the funding schemes: H2020-INFRAIA-2019-1: Research Infrastructure GA 871042 *SoBigData++*, G.A. 952026 *HumanE-AI Net*, ERC-2018-ADG GA 834756 *XAI: Science and technology for the eXplanation of AI decision making*, G.A. 952215 *TAILOR*.

References

1. Abati, D., et al.: Latent space autoregression for novelty detection. In: CVPR. pp. 481–490. Computer Vision Foundation / IEEE (2019)
2. Abdal, R., et al.: Image2stylegan: How to embed images into the stylegan latent space? In: ICCV. pp. 4431–4440. IEEE (2019)
3. Abdi, H., Williams, L.J.: Principal component analysis. Wiley interdisciplinary reviews: computational statistics **2**(4), 433–459 (2010)
4. Akhtar, N., et al.: Threat of adversarial attacks on deep learning in computer vision: Survey II. CoRR **abs/2108.00401** (2021)
5. Amid, E., Warmuth, M.K.: Trimap: Large-scale dimensionality reduction using triplets. CoRR **abs/1910.00204** (2019)
6. Angiulli, F., et al.: Improving deep unsupervised anomaly detection by exploiting VAE latent space distribution. In: DS. Lecture Notes in Computer Science, vol. 12323, pp. 596–611. Springer (2020)
7. Artelt, A., Hammer, B.: On the computation of counterfactual explanations - A survey. CoRR **abs/1911.07749** (2019)
8. Bodria, F., et al.: Benchmarking and survey of explanation methods for black box models. CoRR **abs/2102.13076** (2021)
9. Breunig, M.M., et al.: LOF: identifying density-based local outliers. In: SIGMOD Conference. ACM (2000)
10. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: KDD. pp. 855–864. ACM (2016)
11. Guidotti, R.: Counterfactual explanations and how to find them: literature review and benchmarking. DAMI pp. 1–55 (2022)
12. Guidotti, R., et al.: Factual and counterfactual explanations for black box decision making. IEEE Intell. Syst. **34**(6), 14–23 (2019)
13. Guidotti, R., et al.: A survey of methods for explaining black box models. ACM Comput. Surv. **51**(5), 93:1–93:42 (2019)
14. Guo, W., Diab, M.T.: Modeling sentences in the latent space. In: ACL (1). pp. 864–872. The Association for Computer Linguistics (2012)
15. Hoff, P.D., et al.: Latent space approaches to social network analysis. Journal of the american Statistical association **97**(460), 1090–1098 (2002)

16. Kim, B., et al.: Examples are not enough, learn to criticize! criticism for interpretability. In: NIPS. pp. 2280–2288 (2016)
17. Kim, J., Cho, S.: Explainable prediction of electric energy demand using a deep autoencoder with interpretable latent space. Expert Syst. Appl. **186**, 115842 (2021)
18. Kingma, D.P., et al.: Adam: A method for stochastic optimization. In: ICLR (2015)
19. Kullback, S., Leibler, R.A.: On information and sufficiency. The annals of mathematical statistics **22**(1), 79–86 (1951)
20. Laugel, T., et al.: Comparison-based inverse classification for interpretability in machine learning. In: IPMU (1). Communications in Computer and Information Science, vol. 853, pp. 100–111. Springer (2018)
21. Lundberg, S.M., et al.: A unified approach to interpreting model predictions. In: NIPS. pp. 4765–4774 (2017)
22. Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(11) (2008)
23. McInnes, L., Healy, J.: UMAP: uniform manifold approximation and projection for dimension reduction. CoRR **abs/1802.03426** (2018)
24. Medrano-Gracia, P., et al.: Atlas-based anatomical modeling and analysis of heart disease. Drug Discovery Today: Disease Models **14**, 33–39 (2014)
25. Mukherjee, S., et al.: Clustergan: Latent space clustering in generative adversarial networks. In: AAAI. pp. 4610–4617. AAAI Press (2019)
26. Ng, A., et al.: Sparse autoencoder. CS294A Lecture notes **72**(2011), 1–19 (2011)
27. Peng, X., et al.: Structured autoencoders for subspace clustering. IEEE Trans. Image Process. **27**(10), 5076–5086 (2018)
28. Pol, A.A., et al.: Anomaly detection with conditional variational autoencoders. CoRR **abs/2010.05531** (2020)
29. Pu, Y., et al.: Variational autoencoder for deep learning of images, labels and captions. Advances in neural information processing systems **29** (2016)
30. Ribeiro, M.T., et al.: "why should I trust you?": Explaining the predictions of any classifier. In: KDD. ACM (2016)
31. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat. Mach. Intell. **1**(5) (2019)
32. Schreyer, M., et al.: Detection of accounting anomalies in the latent space using adversarial autoencoder neural networks. CoRR **abs/1908.00734** (2019)
33. Spinner, T., et al.: Towards an interpretable latent space: an intuitive comparison of autoencoders with variational autoencoders. In: IEEE (2018)
34. Stepin, I., et al.: A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. IEEE Access **9**, 11974–12001 (2021)
35. Tan, P., et al.: Introduction to Data Mining (Second Edition). Pearson (2019)
36. Wang, Y., et al.: Understanding how dimension reduction tools work: An empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. JMLR **22**, 201:1–201:73 (2021)
37. Winant, D., et al.: Latent space exploration using generative kernel pca. In: Artificial Intelligence and Machine Learning, pp. 70–82. Springer (2019)
38. Wu, J., et al.: Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In: NIPS. pp. 82–90 (2016)
39. Yang, B., et al.: Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In: ICML. vol. 70, pp. 3861–3870. PMLR (2017)
40. Yeh, C., et al.: Learning deep latent space for multi-label classification. In: AAAI. AAAI Press (2017)
41. Zhang, L., et al.: LSDT: latent sparse domain transfer learning for visual adaptation. IEEE Trans. Image Process. **25**(3) (2016)