SCUOLA
NORMALE
SUPERIORE

Classe di scienze

Corso di perfezionamento in
Data Science

35° ciclo

Understanding and Exploiting the Latent Space to improve Machine Learning models eXplainability.

Settore Scientifico Disciplinare: INF01-INFORMATICA

Candidato/a
dr.(ssa) Bodria Francesco

Relatore/i (o Relatrice/i o Relatrice e Relatore)	Supervisore interno
Prof.(ssa) Fosca Giannotti	Prof.(ssa) Fosca Giannotti
Prof.(ssa) Dino Pedreschi	
Prof.(ssa) Riccardo Guidotti	

Anno accademico 2022/2023

Understanding and Exploiting the Latent Space of Machine Learning models.

Bodria Francesco

May 2023

Submitted to the Scuola Normale Superiore di Pisa
in xx, in partial fulfillment of the requirements for the
Doctoral Program in Data Science

Abstract

In recent years, Artificial Intelligence (AI) and Machine Learning (ML) systems have dramatically increased their capabilities, achieving human-like or even human-superior performance in specific tasks. This increased performance has gone hand in hand with an increase in the complexity of AI and ML models, compromising their transparency and trustworthiness and making them inscrutable black boxes for decision making. Explainable AI (XAI) is a field that seeks to make the decisions suggested by ML models more transparent to human users, by providing different types of explanations. This thesis explores the possibility of using a reduced feature space called "latent space", produced by a particular kind of ML models, as a means for the explanation process. First, we study the possibility of navigating the latent space as a form of interactive explanation to better understand the rationale behind the model's predictions. Second, we propose an interpretable-by-design approach to make the explanation process completely transparent to the user. Third, we exploit mathematical properties of the latent space of certain ML models (similarity and linearity) to produce explanations that are shown more plausible and accurate than those of existing competitors in the state of the art. In order to validate our approach, we perform extensive benchmarking on different datasets, with respect to both existing metrics and new ones introduced in our work to highlight new XAI problems, beyond current literature.

Keywords: eXplainable Artificial Intelligence, XAI, Machine Learning, Latent Space

Pubblications

1. Bodria Francesco, Fosca Giannotti, Riccardo Guidotti, Francesca Naretto, Dino Pedreschi, and Salvatore Rinzivillo. “Benchmarking and survey of explanation methods for black box models.” Data Mining and Knowledge Discovery (DAMI), Springer, 2023.

2. Naretto Francesca, Bodria Francesco, Fosca Giannotti, and Dino Pedreschi. “Benchmark analysis of black-box local explanation methods”, In proceedings of XAI.it 2022 - Italian Workshop on Explainable Artificial Intelligence 2022.

3. Fosca Giannotti, Naretto Francesca, Bodria Francesco, “Explainable AI for Trustworthy AI”, In proceedings of ACAI 2022 school.

4. Bodria Francesco, Salvatore Rinzivillo, Daniele Fadda, Riccardo Guidotti, Fosca Giannotti, and Dino Pedreschi. “Explaining Black Box with visual exploration of Latent Space.”, In Proceedings of 24^{th} Conference in Visualisation (EuroVis), 2022, Rome.

5. Bodria Francesco, Riccardo Guidotti, Fosca Giannotti, and Dino Pedreschi. “Interpretable Latent Space to Enable Counterfactual Explanations.” In Proceedings of the 25^{th} international conference on Discovery Science (DS), 2022, Montpellier.

6. Bodria Francesco, Riccardo Guidotti, Fosca Giannotti, and Dino Pedreschi. “Transparent Latent Space Counterfactual Explanations for Tabular Data.” In Proceedings of the 9th IEEE International Conference on Data Science and Advanced Analytics (DSAA), Shenzen, 2022.

Acknowledgements

I would like to take this opportunity to express my heartfelt gratitude to everyone who has supported me in the completion of my Ph.D. thesis. First and foremost, I would like to thank my family for their unwavering support, encouragement, and love throughout my academic journey. Their constant encouragement, guidance, and sacrifices made it possible for me to pursue my dreams and achieve my goals. I am also deeply grateful to my friends for their support, encouragement, and for being a constant source of motivation. Their positive energy and unwavering support have helped me to keep going in times of difficulty and have made this journey more enjoyable. I would like to extend my heartfelt gratitude to my supervisors for their guidance, support, and expertise throughout my research. Their invaluable insights, constructive feedback, and suggestions were instrumental in shaping my research, and I am deeply indebted to them. I would also like to thank the staff and faculty of the several institutions involved, for their support and for providing me with access to the resources and facilities necessary to conduct my research. Lastly, I would like to express my gratitude to all the participants who generously gave their time and efforts to participate in my research. Once again, I would like to extend my heartfelt gratitude to everyone who has supported me in the completion of my Ph.D. thesis. Thank you.

This work has been supported by the European Community Horizon 2020 program under the funding scheme ERC-2018-ADG G.A. 834756 *XAI: Science and technology for the eXplanation of AI decision making* (https://xai-project.eu/index.html)

Contents

1 Introduction **11**

2 Background **18**

2.1 eXplainable Artificial Intelligence (XAI) 18

2.1.1 Existing XAI Taxonomy for Explanation Methods 20

2.1.2 Categorization of Explanations 21

2.1.3 Feature Importance 22

2.1.4 Rule-Based 29

2.1.5 Prototypes 30

2.1.6 Counterfactuals 33

2.1.7 Transparent by Design Methods 36

2.1.8 Desiderata of an explanation 41

2.2 Benchmarking of XAI methods 46

2.3 The Latent Space 52

2.3.1 Dimensionality Reduction Techniques 53

2.3.2 Generative models 59

2.3.3 Understanding the Latent space 66

3 Interactive exploration of the latent space **75**

3.1 Meaning of SHAP explanations 78

3.2 Methodology 80

3.2.1 Neighborhood Analysis 80
3.2.2 Establishing the Interactive Connection via SHAP 82
3.3 The Interactive Framework 85
3.4 Discussion and Future Directions 90

4 Interpretable by Design Latent Space 92

4.1 Methodology 94
4.1.1 Interpretable Latent Space Learning 95
4.1.2 Counterfactual Explanations 98
4.2 Experiments 101
4.2.1 Latent Space Evaluations 101
4.2.2 Counterfactuals Evaluations 104
4.3 Discussion and Future Directions 109

5 Latent Space Post-Hoc Explanations 111

5.1 Methodology 114
5.1.1 Constrained Interpretable Latent Space with Black Box Predictions 115
5.1.2 Post-Hoc Explanations 120
5.2 Experiments 125
5.2.1 Prototype Explanation Evaluation 127
5.2.2 Counterfactual Explanation Evaluation 132
5.2.3 Comparison with Rule-based Explainers 141
5.3 Discussion and Future Directions 144

6 Conclusions 147

List of Figures

1.1 The concept of XAI: XAI methods must be able to translate models into understandable and useful explanation dialogues for the end user. 14

1.2 Overview of the thesis structure and relationships between different chapters 17

2.1 Existing taxonomy for the classification of explanation methods. . . . 21

2.2 Examples of explanations divided for different data types and explanations. 23

2.3 Examples of SHAP explanations 25

2.4 Contrastive and counterfactual explanations by CEM and DICE 34

2.5 Explanation from the interpretable by design method EBM 40

2.6 Example of Insertion and Deletion metric computation for an XAI method 43

2.7 Critical difference plot for Nemenyi test for tabular explainer comparison 49

2.8 Critical difference plot for Nemenyi test for image explainer comparison 51

2.9 An example of manifold dimensionality reduction. 53

2.10 Graphical representation of Hughes principle 55

2.11 Standard architecture of an autoencoder generative model 60

2.12 Architecture of GAN model 64

2.13 Examples of images generated by GANs 66

2.14 J-Diagram results of the interpolation of pictures in the latent space of a GAN . . . 67
2.15 Interactive image generation framework example . . . 70

3.1 Distribution analysis of the two clusters analyzed in the Titanic dataset. 81
3.2 Illustration of SHAP vectors . . . 82
3.3 Interactive framework for the titanic dataset. . . . 84
3.4 Interactive Latent Space interface . . . 86
3.5 Effect of the feature modification in the latent space. . . . 88

4.1 Description of the vector model used for the Interpretable Latent Space 95
4.2 Metrics used to evaluate ILS . . . 102
4.3 Accuracy of classifiers on `adult` varying the number of latent dimensions k. . . . 105
4.4 Histogram of the percentage of changes in the features among the three methods . . . 106
4.5 Example of counterfactuals produced by ILS, GD, and GSG for the `mnist` dataset . . . 106

5.1 Representation of a possible adversarial problem in counterfactual explanations . . . 112
5.2 Representation of the original data (left) and the rearrangement of CP-ILS (right). The two points H and G are clearly two adversarial examples that can fool the counterfactual methods to believe that these are the valid explanations for the other data. . . . 118
5.3 Score computed for different latent space dimensions for the `Adult` dataset with SVM as black-box . . . 119
5.4 Representation of the contributions of the input features in the latent space. . . . 121

5.5 Visualization of the latent space created using two dimensions. 124

5.6 Visualization of the cluster in the latent space or `iris` and a synthetic dataset. 128

5.7 Evaluation of *1-KNN Accuracy* and *Sparsity* of prototype explanations while varying the number of prototypes on the explanation set. 131

5.8 LEFT: Correlation matrix between the input features of the `adult` dataset and the label "income". RIGHT: Critical difference plot for Nemenyi test with $\alpha = 0.05$. 135

5.9 Counterfactual metrics in comparison with latent space dimension variation. On the top in blue we have the `adult` dataset with the SVM black-box. On the bottom in orange we have the `compas` dataset when using the XGB black-box. The dashed vertical lines represent the dimension used for the experiments. 140

5.10 Two examples of explanations of LORE and CP-ILS from the `adult` dataset. 144

List of Tables

2.1 black-box training results for benchmarking 48
2.2 Fidelity and faithfulness metric results for tabular explainers benchmarking 48
2.3 Stability metric results for tabular explainers benchmarking 49
2.4 Insertion and deletion metrics results for image explainers benchmarking 50
2.5 Sensitivity metric and runtime results for image explainers benchmarking 51

4.1 Datasets statistics. 101
4.2 Space quality metrics. The best scores are in bold. 103
4.3 Accuracy metrics. The best scores are in bold. Uncertainty is on the third decimal. 104
4.4 Example of counterfactuals produced by ILS, Gradient, and GSG for `adult`. Change to the original instance are highlighted in blue. 105
4.5 Counterfactual explanations evaluation in terms of distance and plausibility. 107

5.1 Description of datasets and performance of the black-boxes 126
5.2 Space quality metrics. Best scores are in bold, second best result are underlined. 127

5.3 Results of the metrics computed on `adult` and `german` datasets and on the four black-boxes. The best results are highlighted in bold, second bests are underlined. 130
5.4 Different counterfactuals found for a sample x of the `adult` dataset. . 134
5.5 Results of the counterfactual metrics computed on the four datasets and on the four black-boxes plus the success rate (SR) of the algorithm for `adult` and `compas`. Best results are highlighted in bold, second best results are underlined, W column highlights the number of wins. 137
5.6 Results of the counterfactual metrics computed on the four datasets and on the four black-boxes plus the success rate (SR) of the algorithm for `diva` and `german`. Best results are highlighted in bold, second best results are underlined, W column highlight the number of wins. 138
5.7 Results of the counterfactual metrics computed on the four datasets and on the four black-boxes for both LORE and CP-ILS. The best results are highlighted in bold. 143

Chapter 1

Introduction

Artificial intelligence is increasingly entering our lives. From the simplest tasks, such as recommending favorite movies, to the most complicated, such as autonomous driving, all areas of knowledge and many of the activities we do on a daily basis are influenced by artificial intelligence decisions. Recent developments in AI systems have demonstrated the capability of AI systems to complete tasks that traditionally require high human intelligence, showing that AI algorithms are potentially useful elements for decision support systems. Sometimes they even surpass the human standard. The most advanced systems can use data to learn and use complex patterns that no human is capable of discovering. The AI systems capable of learning patterns and making decisions characterize a subfield of AI algorithms called machine learning (ML). Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to learn from data and imitate intelligent human behavior. Because of the increasing use of machine learning algorithms in artificial intelligence systems, there is often ambiguity about the two terms. When companies today deploy artificial intelligence programs, they are most likely using machine learning, so the terms are often used interchangeably and sometimes ambiguously. The critical difference between traditional AI software and ML approaches is that in ML models, a human developer has not written codes that instruct the system on how to make decisions. Instead, a machine learning model has built its knowledge by training on a large amount of data. They first learn knowledge from a large data set (training phase), then they can generalize this knowledge to new data and make predictions (inference phase).

However, many ML algorithms cannot be deeply analyzed to understand how and why a decision was made. This is why ML systems have been called black boxes: they are oracles that give us an outcome without telling us why. Unfortunately, this is true for the majority of ML algorithms currently in use. Moreover, without a proper explanation, most people will lose trust in the predictions made by these algorithms. Establishing trust in ML learning algorithms is fundamental: we want systems to work as expected and produce transparent explanations and reasons for the decisions they make. In recent years, trust in ML applications has been put to the test. Many attacks have been applied to show their fragility and weaknesses [1]. In particular, it is possible to change the input data in a humanly invisible way but deeply alter the system reaction [2]. Relying on unexplained models may lead to adopting decisions that we do not fully understand or, even worse, violate ethical principles or legal norms. These risks are particularly relevant in high-stakes decision-making scenarios, such as medicine, justice, finance, recruitment, access to public benefits, and so on [3]; the lack of transparency and trust may explain the relatively low adoption rate of current AI-based decision support systems in the mentioned areas. Moreover, companies that embed black-box ML models in their AI products and applications risk incurring a potential loss of safety [4]. For this reason, in 2018, the European Parliament introduced in the GDPR[1] a set of clauses for automated decision-making in terms of *a right of explanation* for all individuals to obtain "meaningful explanations of the logic involved" when automated decision making takes place. Also, in 2019, the High-Level Expert Group on AI presented

[1]https://ec.europa.eu/justice/smedataprotect/

the ethics guidelines for trustworthy AI[2], where explainability is indicated as one of the fundamental requirements for trustworthiness. Today, the same principle has become a cornerstone of the AI Act, the proposed new EU regulation establishing standardized rules on artificial intelligence[3]. Indeed, there is a widespread and increasing consensus on the urgency of implementing appropriate explanation tools, although it represents a scientific challenge that is still largely open. For this reason, new European projects[4] are arising aiming to produce meaningful explanations for AI/ML systems[5].

Explainable AI (XAI) is an emerging field in machine learning that tries to produce explanations to address how black-box decisions of AI systems are made[6]. We have witnessed in the last years the rise of a plethora of XAI methods [5, 6, 7, 8, 9, 10] both from academia and industries. In general, an eXplainable AI method can be defined as an algorithm with the ability to explain, or present in understandable terms, black-box decisions to a human being [11]. There are several ways for extracting explanations from the black-box model [12, 13], each capturing diverse aspects of a black-box model.

Generally a good XAI method [7] should addresses one or more of the following points[7]:

- Verify the model's behavior and identify whether it is acceptable.
- Help debug the model's unexpected behavior and data collection process.
- Present the model's predictions to the different stakeholders and allow them to comprehend the inner working of the model without compromising the privacy of the original training data.

[2]https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai
[3]https://digital-strategy.ec.europa.eu/en/library/proposal-regulation-laying-down-harmonised-rules-artificial-intelligence
[4]https://cordis.europa.eu/project/id/834756
[5]https://xai-project.eu/
[6]https://www.forbes.com/sites/cognitiveworld/2019/07/23/understanding-explainable-ai
[7]https://www.darpa.mil/program/explainable-artificial-intelligence

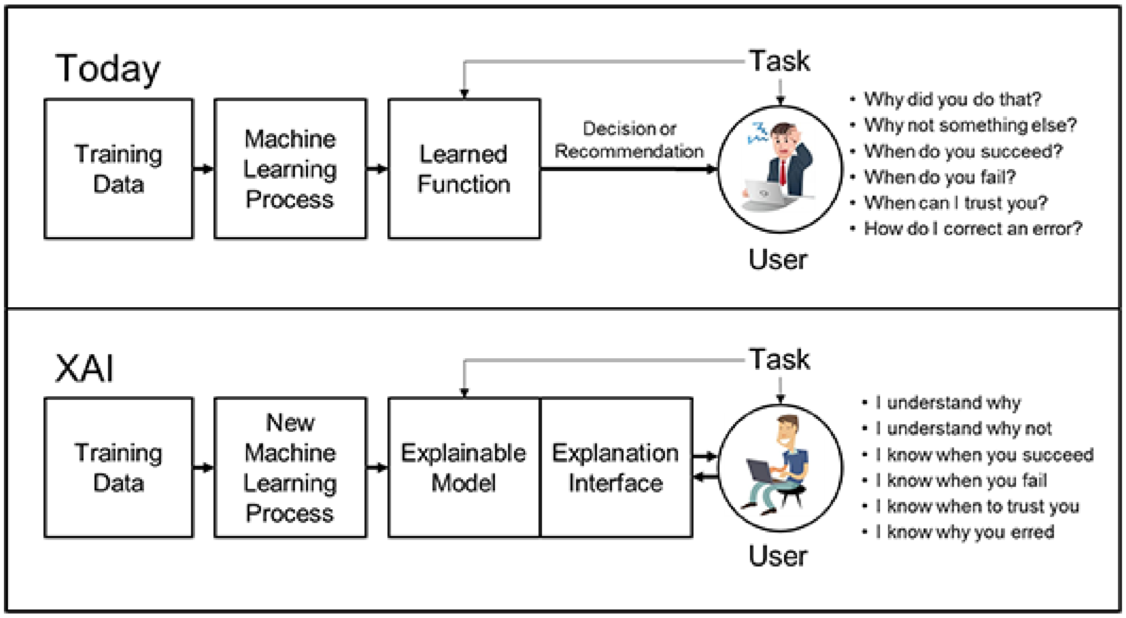


Figure 1.1: The concept of XAI: XAI methods must be able to translate models into understandable and useful explanation dialogues for the end user.

XAI methods must be able to translate model decisions in a human-understandable way (Figure 1.1). However current methods are far away for being perfect [14] and there is much debate on how an explanation should be presented, and several solutions have been proposed in recent years. Generally, good explanations must highlight the right features important to the model's predictions (Accurate), must not change for small variations in the input (Stable), and must represent concepts useful to the end user (Informative).

These criteria are difficult to achieve in high-dimensional datasets. In particular, explanation methods struggle to identify the most essential features. This is due to a large number of features and the fact that distance loses its interpretability in high-dimensionality datasets, making it challenging to produce explanations based on the similarity between data. In addition, methods usually return explanations with a vast number of features considered (e.g., a rule with many conditions). This only confuses the user, who is left with long explanations that become uninterpretable. However, most real-world datasets we encounter in day-to-day life are highly dimensional, often consisting of up to a thousand features. Dimensionality reduction methods are a family of methods capable of reducing the features of a dataset and compressing the dimensionality of the data. These methods map input features into a set of reduced features which are a combination of the original ones. This compression causes the true natural meaning of the features to come out. The meaning of these new features, however, is lost in this compression making them called hidden features or latent features.

One interesting thing about these latent features is that they can be considered as dimensions of a space, called latent space. Researchers found that it is possible to use this space to explore data and even generate new ones. This gave rise to a variety of models capable of producing such latent space. The most famous architecture that can create a latent space is called AutoEncoder. An AutoEncoder is a Machine Learning model composed of two parts: an Encoder and a Decoder. The encoder is responsible for encoding data in the latent space, while the decoder is responsible for decoding back into the original data space. The latent space is learned from the data by the AutoEncoder by minimizing different functions with the final goal that similar points in the input space are also close in the latent space. However, autoencoders are usually composed of very complex ML models making the latent space created opaque.

In this thesis, we explored a new XAI line of research that develops methods that exploit the properties of latent space for the explanation process. In particular, we can use the similarity between the data and create a latent space in which to move and which can be used to retrieve explanations. We explored the possibility of using both an opaque and a transparent latent space to enhance interpretability. To this end, we reviewed the current state of the art of XAI and latent space approaches (chapter 2) and found out which approaches are the main ones in both fields and which can be used for our goals. We studied different approaches to latent space creation and analyzed how they can be used to produce different types of explanations.

The specific research questions of this thesis are the following:

> ***Using explanations to explore and interact with the Latent Space*** *(chapter 3). Can we use the latent space to explore the decision boundary of the black-box? Can XAI methods help enhance such exploration?*
>
> ***Designing a fully Interpretable Latent Space*** *(chapter 4). Is it possible to construct a latent space using interpretable models and still retain its proprieties? Can this new interpretable latent space be used to train ML models and produce explanations?*
>
> ***Using the latent space as a tool for producing post-hoc explanations*** *(chapter 5). Can the latent space proprieties help us in producing better explanations than in the input space? What type of post-hoc explanations can be obtained? How do they compare with other approaches in the literature?*

Analysis of the development and use of XAI and latent space-based methods in the chapter 2 embraces and motivates the other objectives of this thesis. In particular, we analyzed the most common approaches regarding XAI and the latent space but we also tried to benchmark them with respect to several standard metrics. Then, each following chapter explores the three research questions presented here of XAI applied to latent space (Figure 1.2). In each chapter, we present a general overview of the method as well as the result of the experimental part.

Chapter 3 is the first exploration of latent space using existing methods of explanation. In particular, the explanation is provided in the form of an interactive framework to allow the user to explore latent space freely. The following two chapters

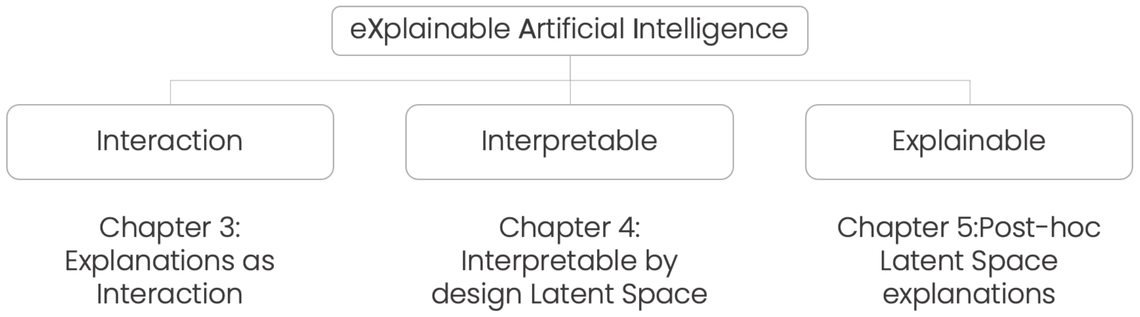

eXplainable Artificial Intelligence
Interaction
Interpretable
Explainable
Chapter 3: Explanations as Interaction
Chapter 4: Interpretable by design Latent Space
Chapter 5:Post-hoc Latent Space explanations

Figure 1.2: Overview of the thesis structure and relationships between different chapters

explore two fundamental aspects of XAI: interpretability and explainability. The former seeks to develop completely transparent methods, while the latter builds methods to explain a given model. In chapter 4, we tried to construct the latent space in a fully transparent way in order to be fully aware of the inner workings of our approach and the explanations produced. Finally, in chapter 5, we developed an XAI method to produce explanations from a given fixed ML model. We obtained different types of explanations and compared them with traditional approaches. Conclusions of the thesis and final remarks are presented in Chapter 6

Chapter 2

Background

This chapter presents the most important works concerning the latent space and XAI besides papers combining these aspects.

2.1 eXplainable Artificial Intelligence (XAI)

Today AI is one of the most important scientific and technological areas, with a tremendous socio-economic impact and a pervasive adoption in many fields of modern society. The impressive performance of AI systems in prediction, recommendation, and decision-making support is generally reached by adopting complex Machine Learning (ML) models that "hide" the logic of their internal processes. As a consequence, such models are often referred to as "black-box models" [5, 15, 16]. Examples of black-box models used within current AI systems include deep learning models like neural networks and ensemble models such as bagging and boosting models. The high performance of such models in terms of accuracy has fostered the adoption of these black-boxes even if their opaqueness may hide potential issues inherited by training on biased or unfair data [17]. COMPAS gives an example of such behavior: an AI system able to predict the risk of recidivism to support judges in such decisions. A few years ago, ProPublica[1] published an article in which they highlighted a racial bias in the COMPAS model employed by the judges in the USA to predict recidivism. The problem with this algorithm was that it predicted a higher recidivism risk for black people since the dataset used for training contained a bias in favor of white people. There have been cases of people incorrectly denied loans, ML-based pollution models stating that highly polluted air was safe to breathe, and generally poor use of limited valuable resources in criminal justice, medicine, energy reliability, finance, and other domains.

Thus there is a substantial risk that relying on opaque models may lead to adopting decisions that we do not fully understand or, even worse, violate ethical principles. Companies are increasingly embedding ML models in their AI products and applications, incurring a potential loss of safety and trust [4]. These risks are particularly relevant in high-stakes decision-making scenarios, such as medicine, finance, and automation. In 2018, the European Parliament introduced in the GDPR[2] a set of clauses for automated decision-making in terms of *a right of explanation* for all individuals to obtain "meaningful explanations of the logic involved" when automated decision making takes place. Also, in 2019, the High-Level Expert Group on AI presented the ethics guidelines for trustworthy AI[3]. Despite divergent opinions among legals regarding these clauses [18, 19, 20], everybody agrees that the need to implement such a principle is urgent and that it is a huge open scientific challenge.

That is where eXplainable AI (XAI) comes in, aiming to make these black-box systems more comprehensible to humans. XAI makes it possible to develop AI systems that are fair and transparent according to certain principles.

- *Build Trust*: If the systems are transparent, it is easier for people to trust and use them.

[1]https://www.propublica.org/datastore/dataset/compas-recidivism-risk-score-data-and-analysis

[2]https://ec.europa.eu/justice/smedataprotect/

[3]https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai

- *Enable Auditing*: Help systems to comply with government regulations.
- *Improve the model*: Looking at the explanations, Data Scientists will be able to fine-tune the models and draw better results.

Leading tech companies like Amazon and IBM often provide XAI tools. They help to describe the AI model pipeline, its expected outcome, and potential biases that can interfere with the model's performance. Today no company that uses artificial intelligence can go without integrating XAI principles into their development process if they want to manage compliance risks. If they fail to provide accounts for their model's decision, they can face million dollar lawsuits which have happened with Facebook and Google many times.

2.1.1 Existing XAI Taxonomy for Explanation Methods

In this section, we synthetically recall the existing taxonomy and classification of XAI methods present in the literature [5, 6, 21, 7, 22, 23]. We summarize the fundamental distinctions adopted to annotate the methods in Figure 2.1.

The first distinction separates explainable by design methods from black-box explanation models:

- ***Intrinsically*** explainable methods are explainable by design methods that return a decision, and the reasons for the decision are directly accessible because the model is transparent.
- ***Post-Hoc*** explanation methods produce explanations for a fixed given black-box.

The second differentiation distinguishes post-hoc explanation methods between global and local methods:

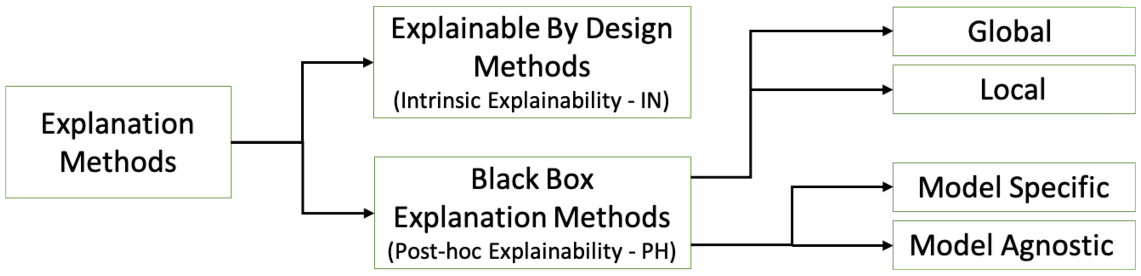

Explanation Methods
Explainable By Design Methods
(Intrinsic Explainability - IN)
Black Box Explanation Methods
(Post-hoc Explainability - PH)
Global
Local
Model Specific
Model Agnostic

Figure 2.1: Existing taxonomy for the classification of explanation methods.

- ***Global*** explanation methods aim at explaining the overall logic of a black-box model. Therefore the explanation returned is a global, complete explanation valid for any instance;
- ***Local*** explainers aim at explaining the reasons for the decision of a black-box model for a specific instance.

The third distinction categorizes the methods into model-agnostic and model-specific:

- ***Model-Agnostic*** explanation methods can be used to interpret any type of black-box model;
- ***Model-Specific*** explanation methods can be used to interpret only a specific type of black-box model.

2.1.2 Categorization of Explanations

An XAI method can produce different types of explanations tied to the data type to explain. Since the type of data we face is very different, the explanations returned are also different. There are three principal data types recognized in the literature: *tabular* data, *images*, and *text*. We can have different types of explanations for each of them, as illustrated in Figure 2.2.

- ***Feature Importance*** methods assign a score for every input feature based on their importance to predict the output. The More the features are responsible for predicting the output, the bigger their score will be.

- ***Rule-Based*** methods are methods that output a set of premises that the record must satisfy in order to meet the rule's consequence.

- ***Prototypes*** are a kind of explanation where the user is provided with a series of examples that characterize a class of the black-box.

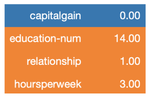
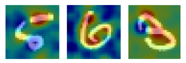
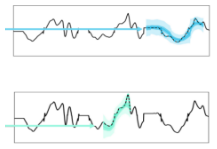
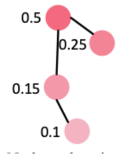
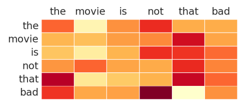
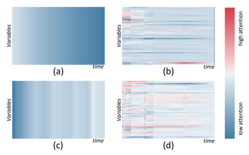
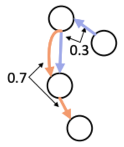
- ***Counterfactuals***, also called contrastive explanations, are a type of explanation in which the user is given either a set of rules to change his prediction or a counter-example of the opposite class.

To be more explicit, in the following sections, we will denote XAI methods as explainers while referring to the ML model as black-boxes.

2.1.3 Feature Importance

For feature importance-based explanations, the explainer assigns an importance value to each feature, which represents how important that feature was for the prediction under analysis. Formally, given a record x, an explainer f models a feature importance explanation as a vector $e = \{e_1, e_2, \ldots, e_m\}$, in which the value $e_i \in e$ is the importance of the i^{th} feature for the decision made by the black-box model $b(x)$. To understand each feature's contribution, the sign and magnitude of each value e_i are considered. If $e_i < 0$, it means that the feature contributes negatively to the outcome y; otherwise, if $e_i > 0$, the feature contributes positively. The magnitude, instead, represents how great the contribution of the feature is to the final prediction y. In particular, the greater the value of $|e_i|$, the greater its contribution. Hence, when $e_i = 0$, it means that the i^{th} feature is showing no contribution to the output decision, and altering the value of that particular feature

Figure 2.2: Examples of explanations divided for different data types and explanations.

TABULAR	IMAGE	TEXT	TIME SERIES	GRAPHS
Feature Importance A vector containing a value for each feature. Each value indicates the importance of the feature for the classification.  capitalgain 0.00; education-num 14.00; relationship 1.00; hoursperweek 3.00	**Saliency Maps** A map that highlights the contribution of each pixel at the prediction. 	**Sentence Highlighting** A map that highlights the contribution of each word to the prediction. the movie is not that bad	**Series Highlighting** A score for every point in the series highlights the contribution to the prediction. 	**Node Highlighting** A score for every node in the graph highlights the contribution of that node to the prediction. 
Rule-Based A set of premises that the record must satisfy in order to meet the rule's consequence. $r = Education \leq College \rightarrow \leq 50k$	**Concept Attribution** Compute attribution to a target "concept" given by the user. For example, how sensitive is the output (a prediction of zebra) to a concept (the presence of stripes)?	**Attention Based** This type of explanation gives a matrix of scores that reveal how words in the sentence are related to each other. 	**Attention Based** This type of explanation gives a matrix of scores that reveal how the points in the series are related to each other. 	**Edge Highlighting** A score for every edge in the graph highlights the contribution of edges to the prediction. 

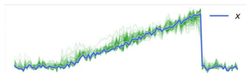
Prototypes

The user is provided with a series of examples that characterize a class of the black box

$p = Age \in [35, 60], Education \in [College, Master] \rightarrow$ "$\geq 50k$"

$p =$ $\rightarrow$ "cat"

$p =$ "... not bad ..." $\rightarrow$ "positive"



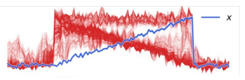
Counterfactuals

The user is provided with a series of examples similar to the input query but with different class prediction

$q = Education \leq College \rightarrow$ "$\leq 50k$"
$c = Education \geq Master \rightarrow$ "$\geq 50k$"

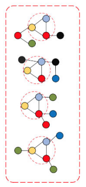
$q = \rightarrow$ "3"
$c = \rightarrow$ "8"

$q =$ The movie is not bad $\rightarrow$ "positive"
$c =$ The movie is that bad $\rightarrow$ "negative"



Graph Prototypes

Identifying which part of the graph has influenced the prediction



should not change the final prediction. An example of a feature based explanation is $e = \{age = 0.8, income = 0.0, education = -0.2\}, y = deny$. In this case, *age* is the most important feature for the decision *deny*, *income* is not affecting the outcome, and *education* has a slightly negative contribution.

Several types of explainers can obtain feature importance scores. The milestone in feature importance explainers is LIME (Local Interpretable Model-agnostic Explanations) [24]. The main idea of LIME is that the explanation may be derived from records generated randomly in the neighborhood of the instance x that has to be explained. By uniformly sampling records in the vicinity of x, LIME is capable of approximating locally the decision boundary of the black-box b and understanding which of the features of x has contributed the most to cross that boundary. However, not all the data are near decision boundaries, and it is possible that in a close neighborhood of x, all the generated data have the same prediction. To overcome this, LIME samples instances both in the vicinity of x and far away from x. The closer the points are to the instance, the greater the possibility of being sampled. After sampling the data, LIME has obtained a perturbed sample of instances $\{z \in \mathbb{R}^d\}$ to feed to the model b and obtain $y_z = b(z)$. These predictions y_z are then used to train a spare linear model g. The feature importance scores consist of the weights of the linear model.

The scores returned by LIME give an indication of which features are more important but do not indicate how exactly they have contributed to the prediction. SHapley Additive exPlanations(SHAP) [25] overcome this by using Shapley values, a concept from cooperative game theory. Each feature is considered as a player, and the payout is the final prediction. Accordingly to the input instance, Shapley values estimate the payout among the input features. The payouts are the feature import scores returned by SHAP. The sum of each score gives the final prediction of the black-box. For this reason, the explanations provided by SHAP are called *additive feature attribution methods* and respect the following definition:

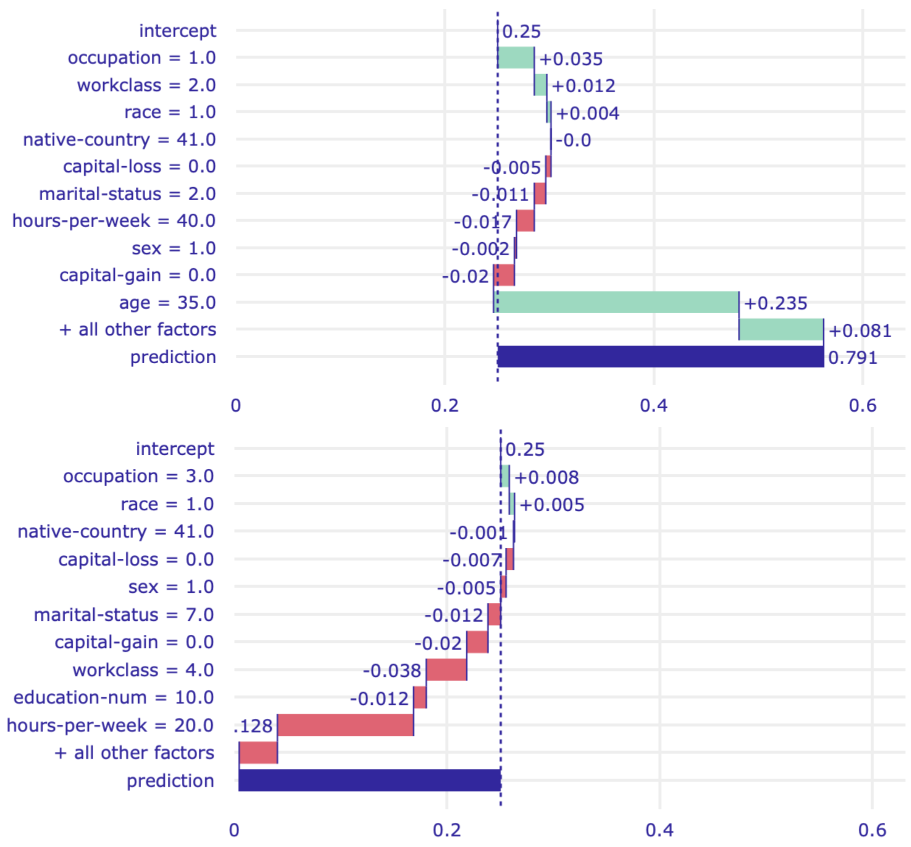


Figure 2.3: Explanations of SHAP for two records. The y-axis is the feature importance, and the x-axis is the positive/negative contribution. The feature importance values sum from the top to the bottom to obtain the final prediction. In the top example, the black-box output of 0.791 is decomposed into contributions of singular features. Age was the one contributing the most. In the bottom one, most features contributed negatively, with hour-per-week being the most influencing.

$b(x) \approx g(x) = \phi_0 + \sum_{i=1}^{M} \phi_i x_i$, where x is the instance to explain, $\phi_i \in \mathbb{R}$ are feature importance scores, M is the number of input features, and ϕ_0 is called expected value and represent the prediction if all the features had zero contribution. SHAP has three properties: *(i) local accuracy*, meaning that $g(x)$ matches $b(x)$; *(ii) missingness*, which allows for features $x_i = 0$ to have no attributed impact on the SHAP values; *(iii) stability*, meaning that if a model changes so that the marginal contribution of a feature value increases (or stays the same), the SHAP value also increases (or stays the same). The construction of the SHAP values allows us to employ them both *locally* (each observation gets its own set of SHAP values) and *globally* (by exploiting collective SHAP values). There are different strategies to compute SHAP's values: *KernelExplainer*, *LinearExplainer*, *TreeExplainer*, *GradientExplainer*, and *DeepExplainer*. In particular, the *KernelExplainer* is an agnostic explainer, while the others are specifically designed for different kinds of ML models.

The last popular approach to obtain feature importance scores is by extracting them from the black-box's gradients. This process can only be applied to differentiable models but has been proven very effective when compared with other feature importance explainers. Indeed, directly extracting the weights from the black-box ensures their validity in representing model decisions. They are the actual values, not an approximation. Integrated Gradient (INTGRAD) [26] is the most famous explainer that applies this methodology. Formally, given the black-box b and the instance to explain x, let x' be the baseline input. The baseline x' represents the null data: for example, for the image domain, the baseline is generally a black or a white image, while for other data can be a vector of zeros or ones. INTGRAD constructs a path from x' to x and computes the gradients of points along the path. For example, with images, the points are taken by overlapping x on x' and gradually modifying the opacity of x. Integrated gradients are obtained by cumulating the gradients of these points. Formally, the integrated gradient along the i^{th} dimension for an input x and baseline x' is defined as follows. Here, $\partial b(x)/\partial x_i$ is the gradient of $b(x)$ along the i^{th} dimension. The equation for computing the scores is:

$$e_i(x) = (x_i - x'_i) \int_{\alpha=0}^{1} \tfrac{\partial b(x' + \alpha(x - x'))}{\partial x_i} \; d\alpha$$

Choosing the proper baseline is more complex than it might seem. For example, a black baseline image could cause the method to lower the importance of black pixels in the source image. This problem is due to the difference between the input and the baseline $(x_i - x'_i)$ present in the integral equation. One possible fix is to average with different baselines [27] or use particular records [28].

These were the most common approaches of feature importance methods. One thing to mention is that feature importance for image data is called saliency maps. A *Saliency Map (SM)* is an image in which a pixel's brightness represents how important the pixel is for the prediction. Formally, a SM is modeled as a matrix S, which dimensions are the sizes of the image we want to explain, and the value s_{ij} is the saliency values of the pixel with indices i and j. The greater the value of s_{ij}, the bigger the saliency of the pixel. To visualize SM, we can use a divergent color map, for example, ranging from red to blue. A positive value (red) means that the pixel ij has contributed positively to the classification, while a negative one (blue) means that it has contributed negatively. There are two approaches to creating SMs. The first one assigns a saliency value to every pixel of the image. The second one segments the image into different pixel groups using a segmentation method, assigning a saliency value for each segmented group. Pixel-based explainers are the most common methods since there are easier to produce. INTGRAD [26] can be easily applied to images. Given the black-box b, the instance to explain x, and let x' be the baseline input (the baseline x' is generally chosen as a black image). INTGRAD constructs a path, varying opacity of the image, from x' to x and computes the gradients of points along the path. Integrated gradients are obtained by cumulating the gradients of these points. ϵ-LRP, Layer-wise Relevance Propagation [29] explains the classifier's decisions by decomposition. ϵ-LRP redistributes the black-box prediction backward to the input using local redistribution rules until it assigns a relevance score to each input pixel. The simple ϵ-LRP rule redistributes relevance from layer $l+1$ to layer l: $R_i = \sum_j \frac{a_i w_{ij}}{\sum_i a_i w_{ij} + \epsilon} R_j$ where a_i and is the activation of the neuron i, w_{ij} is the weight connecting the neurons of i and j of the two layers and a small stabilization term ϵ is added to prevent division by zero. DEEPLIFT [30], computes saliency maps in a backward fashion similarly to ϵ-LRP, but it uses a baseline reference like in INTGRAD. DEEPLIFT uses the slope instead of the gradients, which describes how the output $y = b(x)$ changes as the input x differs from the baseline x'. Like ϵ-LRP, an attribution value r is assigned to each layer i of the black-box going backward from the output y. SHAP has two variants that can be employed for image classification: DEEP-SHAP and GRAD-SHAP. DEEP-SHAP is a high-speed approximation algorithm for SHAP values for deep learning models for images that build on a connection with DEEPLIFT. The implementation differs from the original DEEPLIFT by using a distribution of background samples as a baseline instead of a single value. It uses Shapley equations to linearise non-linear components of the black-box, such as max, softmax, products, divisions, etc. GRAD-SHAP, instead, is based on INTGRAD. As an adaptation to make INTGRAD value approximate SHAP values, GRAD-SHAP reformulates the integral as an expectation and combines that expectation with sampling reference values from the background dataset. GRAD-CAM [31] uses the gradient information that flows into the last convolutional layer of a convolutional neural network to assign saliency values to each neuron for a particular decision. GRAD-CAM++ [32] extends GRAD-CAM solving some related issues about robustness. If multiple objects have slightly different orientations or views, different feature maps may be activated with differing spatial footprints. GRAD-CAM++ fix this problem by taking a weighted average of the pixel gradients. RISE [33] produces saliency map for an image x using a masking mechanism. RISE generates N random mask $M_i \in [0, 1]$ from Gaussian noise. The input image x is element-wise multiplied with these masks M_i, and the result is fed to the black-box. The saliency map is obtained as a linear combination of the masks with the predictions corresponding to the respective masked inputs. Regarding the segmentation-based explainers, most methods are just variations of the already presented, inserting a segmentation algorithm. LIME can also be used for retrieving feature importance. LIME divides the input image into segments called *superpixels*. Then it creates the neighborhood by randomly substituting the super-pixels with a uniform, possibly neutral, color. XRAI [34] is INTGRAD augmented with segmentation. XRAI iteratively segments the image and tests each region's importance using INTGRAD, fusing smaller regions into larger segments based on attribution scores. The segmentation is repeated several times to reduce the dependency on the image segmentation algorithm. One problem with SMs is the confirmation bias, i.e., a user can hardly realize if a SM is a good explanation or only shows what the user wants to see [14]. SMOOTHGRAD [35], and most guided methods that use the target label to alter the SM suffer from this bias. Usually, a saliency map is created directly on the gradient of the model's output signal w.r.t. the input $\partial y/\partial x$. SMOOTHGRAD augments this process by smoothing the gradients using the prediction labels of the black-box.

2.1.4 Rule-Based

After Feature importance, the most widely used explanations are rules. Rules are the most simple and basic form of explanation and are among the most accepted since they are clear to humans. Decision rules give the end-user an explanation of the precise reasons that lead to the final prediction. A decision rule r, also called *factual* or *logic* rule [36], has the form $p \rightarrow y$, in which p is a premise, composed of a Boolean condition on feature values, while y is the consequence of the rule. In particular, p is a conjunction of split conditions of the form $x_i \in [v_i^{(l)}, v_i^{(u)}]$, where x_i is a feature and $v_i^{(l)}, v_i^{(u)}$ are lower and upper bound values in the domain of x_i extended with $\pm\infty$. An instance x *satisfies* r, or r *covers* x, if every Boolean conditions of p evaluate to true for x. If the instance x to explain satisfies p, the rule $p \rightarrow y$ represents a candidate explanation of the decision $g(x) = y$. Moreover, if the interpretable predictor mimics the black-box behavior in the neighborhood of x, we further conclude that the rule is a candidate local explanation of $b(x) = g(x) = y$. We highlight that, in the context of rules, we can also find the so-called *counterfactual rules* [36]. Counterfactual rules have the same structure as decision rules, with the only difference being that the consequence of the rule $\overline{y}$ is different w.r.t. $b(x) = y$. They are important to explain to the end-user what should be changed to obtain a different output. An example of a rule explanation is $r = \{age < 40, income < 30k, education \leq Bachelor\}, y = deny$. In this case, the record $\{age = 18, income = 15k, education = Highschool\}$ satisfies the rule above. A possible counterfactual rule, instead can be: $r = \{income > 40k, education \geq Bachelor\}, y = allow$.

Rules can be obtained in several ways. One approach from the same authors of LIME is ANCHOR [37]. This approach's name comes from the output rules, called *anchors*. The idea is that, for decisions on which the anchor holds, changes in the rest of the instance's feature values do not change the outcome. To obtain the anchors, ANCHOR perturbs the instance x obtaining a set of synthetic records employed to extract anchors with precision above a user-defined threshold. Other approaches like LORE exploit a genetic algorithm for creating the neighborhood of the record to explain, similar to the LIME approach. The rules are then learned by a decision tree trained on the generated neighborhood. Different approaches involve the perturbation of the input [38] or extracting rules directly from a decision tree [39].

2.1.5 Prototypes

Another way to explain a prediction is in terms of similarity to other data. For example, "Your loan was denied because you are very similar to another person who defaulted his loan." According to cognitive science, humans understand categories. In particular, human categorization can be modeled as the use of prototypes: representative examples of the category as a whole [40]. The membership of an item in the category is determined by its similarity to the prototypes of the category. A prototype also called an archetype or artifact, is an object representing a set of similar records. Prototypes serve as examples: the user understands the model's reasoning by looking at records similar to his/hers. It can be *(i)* a record from the training dataset close to the input data x; *(ii)* a centroid of a cluster to which the input x belongs to or *(iii)* even a synthetic record, generating following some ad-hoc process. Depending on the explainer considered, different definitions and requirements to find a prototype are considered.

The first methodology developed for searching for prototypes in the data is a "before the model" methodology, in the sense that it only analyses the distribution of the dataset under analysis. MMD-CRITIC [41] produces prototypes as explanations for a dataset using *Maximum Mean Discrepancy (MMD).* MMD-CRITIC selects prototypes by measuring the difference between the distribution of the instances and the instances in the whole dataset. The set of instances nearer to the data distribution are called prototypes. MMD-CRITIC shows only minority data points that differ substantially from the prototype but belong in the same category. Furthermore, MMD-CRITIC also shows exceptions or criticisms: data that differ substantially from the prototype but still belong to the category. Criticisms represent points that are not well explained by the prototypes. They are selected as the farthest points from the center of data distribution. MMD-CRITIC selects criticisms from parts of the dataset underrepresented by the prototypes, with an additional constraint to ensure the criticisms are diverse. PROTODASH [42] is a variant of MMD-CRITIC. It is an explainer that employs prototypical examples and criticisms to explain the input dataset. Differently, w.r.t. MMD-CRITIC, PROTODASH associates non-negative weights, which indicate the importance of each prototype. In this way, it can reflect even some complicated structures.

Other approaches concern privacy-related prototype explanations. Privacy-Preserving Explanations [43] is the first approach that considers the concept of *privacy in explainability* by producing *privacy protected explanations.* To achieve a good trade-off between privacy and comprehensibility of the explanation, the authors construct the explainer by employing *micro aggregation* to preserve privacy. In this way, the authors obtained a set of clusters with a representative record c_i, where i is the i-th cluster. From each cluster, a *decision trees* is extracted to provide an exhaustive explanation while having good comprehensibility due to the limited depth of the trees. When a new record x arrives, a representative record and its associated shallow tree are selected. In particular, from g, the representative c_i closer to x is selected, depending on the decision of the black-box.

Instead of explaining the data, some methods have applied supervised approaches. Supervised approaches use label or prediction information to find a prototype set that explains the black-box behavior. Hase et Al. created a taxonomy using hierarchically organized prototypes that can be used for classification purposes [44]. Bien and Tibshirani [45] have proposed ProtoSelect, a prototype selection method with the goal of building not only a condensed view of the data but also an interpretable model. ProtoSelect is designed so that each prototype satisfies a set of desired properties of *Sparsity* and *Accuracy*. Future predictions can be made simply by looking at the classification of the most similar prototype.

A prototype is not only a set of data. PROTOPNET [46] figures out that prototypes can be parts of images and then can be used to make the classification process interpretable. A special architecture is needed to produce this particular type of prototypes. The model learns a limited number of prototypical parts from the dataset useful in classifying a new image. The model identifies several parts on the query image that look like some training image prototypical parts. Then, it predicts based on a weighted combination of the similarity scores between parts of the image and the learned prototypes. The performance is comparable to the actual state of the art but with more interpretability. Another variant for building prototypes is to evaluate influence. INFLUENCE FUNCTIONS [47], instead of building prototypes for a class, it tries to find the most responsible data for a given prediction using influence functions. Influence functions is a classic technique from robust statistics to trace a model's prediction through the learning algorithm and back to its training data, thereby identifying training points most responsible for a given prediction. Visualizing the training points most responsible for a prediction could be useful for more in-depth insights into model behavior.

2.1.6 Counterfactuals

Counterfactual thinking [48] is a concept in psychology that involves the human tendency to create possible alternatives to events that have already occurred, something that is contrary to what actually happened. In XAI, these changes can be given in terms of records with similar characteristics as the instance of interest but with an altered outcome. For instance, a record with different features or a text with missing/added words [12, 49]. In this sense, we speak about instance-based counterfactual explanations [50] or counter-exemplars [51]. Another approach is to find counterfactual rules: a set of rules that guide the user to change the outcome of the black-box. There are several methods in the context of counterfactual explanations, even if the majority are only defined from a theoretical point of view.

Wachter et al. [52] formulated that an explanation tells us that $b(x) = y$ was returned because variables of x have values $x_1, x_2, ..., x_n$. Instead, a counterfactual explanation focuses on the opposite prediction. If x_1 and x_2 had values x'_1, x'_2 and all the other variables remained constant, $b(x') = \neg y$ would have been returned, where x' is the record x with the suggested changes. The counterfactual explanation must also be close to the original instance. Following this reasoning, it is possible to search for counterfactual explanations by minimizing a loss function. This searching strategy is called optimization searching since the counterfactual search is done by minimizing a loss [9]. In [52], such loss is composed of a quadratic distance between the desired outcome and the black-box prediction and the distance between the original sample and the counterfactual. The loss function minimized by WACH is defined as

$$\lambda(b(x') - y')^2 + d(x, x')$$

where the first term is the quadratic distance between the desired outcome y and

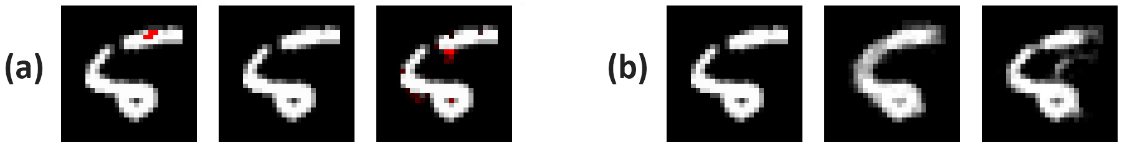


Figure 2.4: Two different counterfactual explanations. **(a)**:Explanation of CEM on `mnist`: the instance to explain on the center, *Pertinent Negative* left, and *Pertinent Positive* right. **(b)**: Explanation of DICE on `mnist`: left to right, the instance to explain, the closest counterfactuals labeled as 6 and 8.

the classifier prediction on x, and the second term is the distance d between x and x'. The parameter λ balances the contribution of the first term against the second term. A low value of λ means that we prefer x' similar to x, while a high value of λ means we aim for predictions close to the desired outcome y. WACH suggested to maximize λ while minimizing the loss. Thus, λ becomes a parameter of the search problem, and a tolerance ϵ is used to constraint the classification of x not too far away from y, i.e., $|b(x) - y| < \epsilon$. Hence, the loss measures how far the outcome of the counterfactual $b(x)$ is from the desired outcome y and how far the counterfactual x' is from the instance of interest x. The distance function d adopted is a crucial characteristic in any counterfactual explainer. Wachter et al. adopt the Manhattan distance weighted with the inverse median absolute deviation (MAD) of each feature, i.e.,

$$d(x, x') = \sum_{i}^{m} \frac{|x_i - x'_i|}{MAD_i}$$

where MADi is the median absolute deviation of the i-th feature. Any other distance, such as the Euclidean distance, can be used. The loss function can be minimized through any suitable optimization algorithm. A drawback of WACH is that setting the initial value of λ a priori is unclear, as well as the value of ϵ.

A more greedy method is called Growing Spheres Generation [53] (GSG). GSG relies on a greedy approach: it grows 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 synthetic counterfactuals randomly in all directions of the feature space until the decision boundary of the classifier is crossed. It starts by generating z instances using a uniform distribution within a given radius r. The radius is halved until, for all the instances z, we have that $b(z) = b(x)$. Then the previous radius is considered, and the most similar instance to z to x is returned as a valid counterfactual x'. We can say that gsg generates candidate counterfactuals in the feature space in a $l2$-spherical layer around x until a valid counterfactual is found. This gives the name to the algorithm. However, greedy approaches can return only one counterfactual per instance to explain and sometimes can be unplausible.

Diverse Counterfactual Explanations (DICE) [54] solves an optimization problem with several constraints to ensure *diversity*, *feasibility*, and *proximity* when returning counterfactuals. Feasibility is critical in the context of counterfactuals since it allows for avoiding unfeasible examples. For example, consider the case of a classifier that determines whether to grant loans. If the classifier denies the loan to an applicant with a low salary, the cause may be low income. However, a counterfactual such as "You have to double your salary" may be unfeasible, and hence it is not a satisfactory explanation. Feasibility is achieved by adding a penalty term in the counterfactual loss that favors counterfactuals with fewer modified features. Besides feasibility, another factor used by DICE is diversity, which provides different ways of changing features to alter the outcome class. Indeed, DICE returns a set of k different counterfactuals for the input x. Diversity is captured by building on determinantal point processes, which have been adopted for solving subset selection problems with diversity constraints [55]. DICE use the following metric based on the determinant of the kernel matrix given the counterfactuals $diversity = det(K)$, where $K_{i,j} = \frac{1}{1+dist(c_i,c_j)}$ and $dist(c_i, c_j)$ denotes a distance metric between the two counterfactual examples. Proximity is achieved as the (negative) vector distance between the original input and counterfactual example features. Counterfactuals are returned by minimizing a loss composed of these three measures weighted with different fixed weights.

Counterfactual reasoning is claimed to deal with cases where the antecedent, i.e., the instance under analysis, is varied to change the prediction outcome: "Would the customer has had the loan accepted if she had an income of 15,000$?" However, in [56] and [57] are illustrated contrastive explanations: counterfactual explanations related to situations where different outcomes are analyzed: "what made the difference between the customer who got the loan accepted and the customer who got the loan refused?" The contrastive explanation shifts the focus to the answer instead of the question. The contrastive explanation method (CEM) described in Dhurandhar et al. [58] formulated that the features which are minimally sufficient to obtain a certain outcome are called pertinent positives, while the features whose absence is necessary for the outcome are named pertinent negatives. Contrastive Explanation Method (CEM) has two components: *Pertinent Positives* and *Pertinent Negatives*. The first can be seen as prototypes and are the minimal and sufficient factors that have to be present to obtain the output y, while the latter are contrastive factors that should be minimally and necessarily absent. CEM is formulated as an optimization problem over the perturbation variable δ. In particular, given x to explain, CEM considers $x' = x + \delta$, where δ is a perturbation applied to x. During the process, there are two values of δ to minimize: δ^p for the pertinent positives and δ^n for the pertinent negatives. CEM solves the optimization problem with a variant that employs an autoencoder to evaluate the closeness of x' to the data manifold.

Other approaches involves the use of a genetic algorithm [59, 60], Shapley values [61], or regression based searches [62].

2.1.7 Transparent by Design Methods

Transparent by Design methods, also called intrinsic methods, require a specific section. In the recent explosion of work on "explainable ML", the main focus of the authors was on post-hoc methods (Section 2.1.1), where a second model is created to explain the first black-box model. This is because it is easier to create a reasonably accurate black-box model and then explain it rather than create a transparent one. The complexity of black-box models will always be an advantage in terms of accuracy over transparent ones. It only matters the quantity and the quality of the data. Often, companies and researchers throw all the data into the model, and something will come up. The prediction explanation only comes in second after a model with acceptable accuracy has been obtained. However, this is problematic since post-hoc explanations are often unreliable and misleading. The problem with post-hoc explanations is that they must be wrong. They cannot have perfect fidelity with respect to the original model. If the explanation were completely faithful to what the original model computes, the explanation would equal the original model. One would not need the original model in the first place, only the explanation. This leads to the danger that any explainer for a black-box model can accurately represent the original model in parts of the feature space. An inaccurate explanation model limits trust in the explanation and, by extension, trust in the black-box it is trying to explain. An explainable model that has a 90% agreement with the original model indeed explains the original model most of the time. However, an explanatory model that is correct 90% of the time is wrong 10% of the time. If a tenth of the explanations is incorrect, one cannot trust the explanations, and thus one cannot trust the original black-box. If we cannot know whether our explanation is correct, we cannot trust either the explanation or the original model. Transparent by design methods come to solve this problem. Due to the intrinsic transparency nature, the prediction decision process is known by construction. However, the accuracy of the prediction would suffer from this design. Usually, transparent-by-design models are weaker in terms of accuracy than black-box models. Also, they are usually difficult to build since they require considerable knowledge of the data used.

In most cases, it is a matter of a trade-off between accuracy and transparency, and the use depends on the case. In high-stack decision-making problems, like healthcare, it is usually preferable to use a transparent method since there is the need to justify the action [3]. However, even in this case, if an algorithm can save lives with a 10% improvement against the transparent model, why would you not use it?

The explanations produced for such types of methods are of the same typology as the one presented before. However, they are not derived from a black-box but are built together with the model. The final model can not only predict new data but, at the same time, can also explain it. The most simple example architecture of a transparent-by-design model is Decision Trees. Decision Trees have been the primary type of interpretable design models for quite a while. Due to their high interpretability, they are the go-to algorithm for real-world business applications. They can be explained as a series of questions of the type of if-else statements that are very human-friendly. Also, the time required for the learning algorithm to make predictions is very low. However, they are unstable, relatively inaccurate, and prone to overfitting. A small change in data can lead to a vastly different decision tree. This can be rectified by replacing a single decision tree with a random forest of decision trees. By doing this, we increase the complexity of the model and, therefore, lower its transparency. Tools like xgboost [63], which relies on Boosting Tree, have implemented some Feature Relevance in years to explain their prediction, but they remain a black-box for all intents.

A ruled-based classifier is a similar approach to Decision trees [64], always based on rule explanation. It provides an output set of rules requiring no further explanations. Nevertheless, the comprehensibility of rule models is also not without caveats. For example, while individual rules may be well understood, the complete rule model may lose its explainability if there are too many rules. Interpretable Decision Sets (IDS) [65] is a rule learning algorithm that provides means for balancing the model size and other facets of interpretability with prediction performance through user-set weights.

Furthermore, ruled-based explainers are not the only methods to make transparent by design models. InterpretML[4] is a package from Microsoft that offers a new interpretability algorithm called Explainable Boosting Machine (EBM), which is based on Generalized Additive Models (GAMS). Generalized additive models were initially invented by Trevor Hastie and Robert Tibshirani in 1986 [66]. Although GAM does not receive sufficient popularity yet as a random forest or gradient boosting in the data science community, it is undoubtedly a powerful yet simple technique. The idea of GAM is that the relationships between the individual predictors and the dependent variable follow smooth functions that can be linear or non-linear. These relationships can be estimated simultaneously and then added up. $y = f_1(x_1) + f_2(x_2) + ...$ This allows computing the exact contribution of each feature to the final prediction y. Although a GAM is easy to interpret, its accuracy is significantly less than more complex models that permit interactions. For this reason, Lou et al. [67] also added interaction terms in the sum of the contributions and called it GA2M: $y = \sum f_i(x_i) + \sum f_{ij}(x_i, x_j)$ As a result, GA2M significantly increases the prediction accuracy but still preserves its nice interpretability. Although the pairwise interaction terms in GA2M increase accuracy, it is highly time-consuming and CPU-hungry. EBM solves the computational problem by learning each smooth function $f(x)$ using bagging and gradient boosting techniques. The functions are modeled as straightforward decision trees, and the resulting adding function of the prediction is a stepwise function. This results in a better computational cost without losing accuracy or interpretability. An example is shown in Figure 2.5

These methods, while fully transparent, have lower performance than very complex black-boxes like Neural Networks. The next step in transparent by design methods is to make Neural Networks transparent. The groundbreaking work on this topic is done by Chen et al. [68]. They won the FICO challenge in 2018 by producing a model capable of performance similar to other black-boxes but fully transparent.

[4]https://interpret.ml/

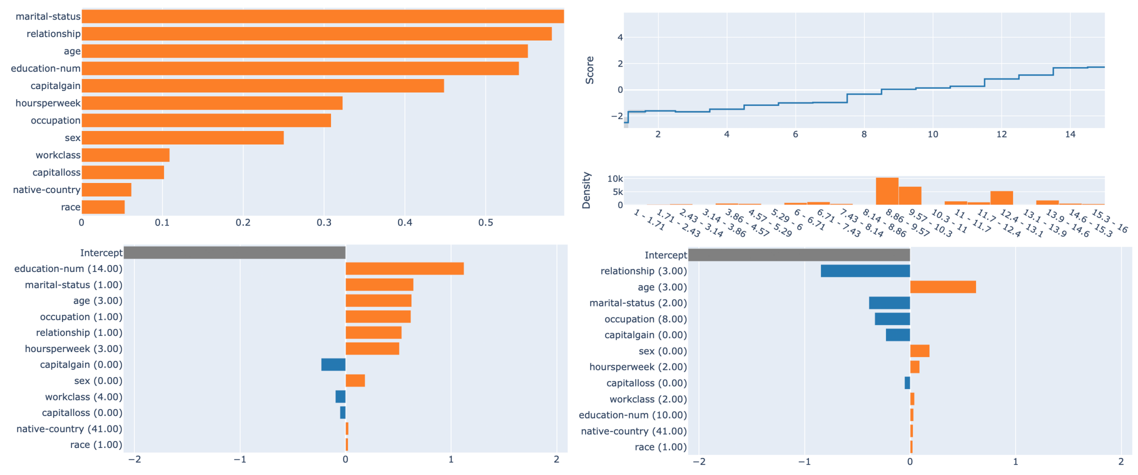


Figure 2.5: *TOP*: Overall global explanation (left), an example of a global explanation (right).
BOTTOM: Local explanations of EBM: left, a record classified as 1 ; right a record classified as 0.

They call their globally interpretable model a two-layer additive risk model. It was designed to resemble traditional subscale models, where the features are partitioned into meaningful subgroups, and the subgroup scores are later combined into a global model. They combined these features by summing the sub-features and applying a sigmoid activation function as in classic Neural Networks. The scores of every sub-feature are computed by minimizing a classification loss. The interpretability in this type of model is in the sub-feature system. Since we have a score for every feature, we can decompose the final prediction in singular feature contribution as EBM did before. The drawback of this approach is that building the subfeature is not done automatically and requires solid domain knowledge to do it efficiently.

More problems arise when moving to image or text data. For years Deep Neural Networks have been predominant in the image domain world. Due to the data's complexity, transparent design methods suffer in performance even more than before. New techniques are arising that promise to enhance the transparency of Deep Neural Networks, but we are far from creating a fully transparent-by-design method.

The most promising approach to transparency is by using Concepts. Most ML models operate on low-level features like edges and lines in a picture that does not correspond to high-level concepts that humans can easily understand. In [14, 69], they pointed out that feature-based explanations applied to state-of-the-art complex black-box models can yield non-sensible explanations. *Concept-based explainability* constructs the explanation based on human-defined concepts rather than representing the inputs based on features and internal model (activation) states. This idea of high-level features is more familiar to humans, that are more likely to accept it. High-level concepts could be used to design a transparent model. During training, each layer of a deep learning model encodes the features of the training images into a set of numerical values and stores them in its parameters. The lower layers of a neural network will generally learn basic features such as corners and edges. The higher layers of the neural network will detect more complex features such as faces, objects, and entire scenes. Ideally, a neural network would represent concepts relevant to the classes of images it is meant to detect. Nevertheless, we do not know that, and deep learning models are prone to learning the most discriminative features, even if they are the wrong ones.

2.1.8 Desiderata of an explanation

There are many concepts and opinions regarding the desiderata for an explanation. First of all, the literature agrees that there is no universal type of explanation that is good for everyone and for all purposes: in some contexts, a more high-level explanation is needed; in others, a more technical one, depending on who is receiving the explanation and why the explanation is required.

The goodness of an explanation should be evaluated by considering its validity and utility. In the literature, these desiderata are evaluated in terms of goodness, usefulness, and satisfaction of explanations. The state-of-the-art provides both qualitative and quantitative methodologies for evaluating explanations.

Regarding the **qualitative evaluation**, it values the actual usability of explanations from the point of view of the end-user, such as if they satisfy human curiosity and trust. A large body of research has identified that one of the critical variables that may influence decisions about automation use is a user's trust in the automation. Users tend to use AI that they trust while rejecting the one they do not. For appropriate use to occur, users' trust must match the true capabilities of the AI system. The appropriateness of trust can be evaluated in terms of its calibration [70], or "the correspondence between a person's trust in the automation and the automation's capabilities". Trust is often seen as a 'single-direction road,' i.e., something that needs to be increased. However, 'trust' in AI technologies must be reduced to boost transparency-based models. In this context, an interesting work of Doshi-Velez [11] proposes a systematization of evaluation criteria into three categories:

1. **Functionally-grounded** metrics aim to evaluate the interpretability by exploiting some formal definitions used as proxies, and they do not require humans for validation.

2. **Application-grounded** evaluation methods require human experts able to validate the specific task and explanation under analysis [71, 72].

3. **Human-grounded** metrics evaluate the explanations through humans who are not experts. The goal is to measure the overall understandability of the explanation in simplified tasks [65, 73].

Quantitative evaluation concerns metrics that try to define how well a method produces valid explanations quantitatively. They are tied down to the explanation type since they are usually used to compare other explainers producing the same explanation.

Feature importance explanations indicate the impact of a particular feature on the final prediction. In particular, they rank the features from the most important to

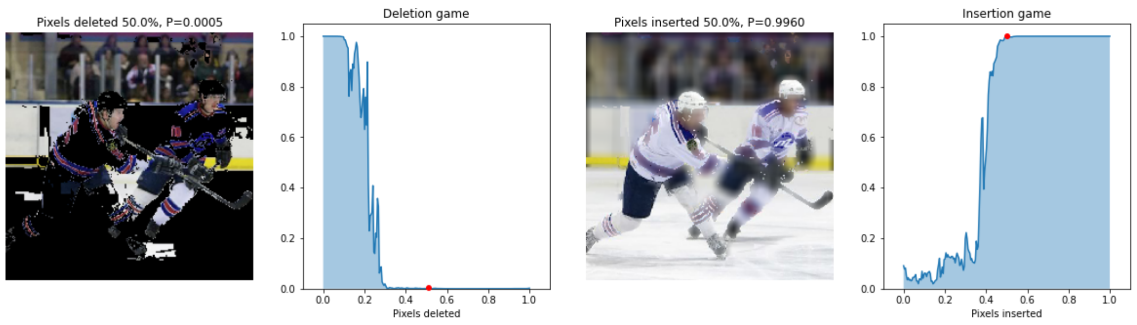


Figure 2.6: Example of Insertion (on the left) and Deletion (on the right) metric computation performed with LIME and a hockey image from the `imagenet` dataset. For the Deletion metric, every pixel is substituted with a black one until we have a fully black image. For the Insertion metric, the image starts blurred, and then pixels are "added" by restoring their opacity. The red dot in the graphs indicates the position of the images (50% deleted/inserted).

the less one, giving them a score. One way to validate feature importance explainers is to check the true importance of features. This metric is called *faithfulness* [74] and consists of removing features from the original sample in the order given by the feature importance explanation and observing the drop in black box accuracy. The intuition behind deletion is that removing the "cause" will force the black-box to change its decision. A complementary approach is to insert features instead of removing them. *monotonicity* evaluates the importance of the features in the prediction by incrementally adding each one in order of increasing importance given by the feature importance. In this case, we expect that the black-box performance increases by adding more and more features, thereby resulting in monotonically increasing model performance. Petsiuk et al. [33] gives a similar approach to deletion and insertion metrics. They measured the modification in accuracy for every feature removal or inserted and obtained an accuracy curve with respect to the number of features. The final score is given by the area under the accuracy drop curve. For deletion, the lower the better, for insertion, the greater the better (Figure 2.6).

Regarding rule-based methods, the comparison is difficult. In this case, the evaluation focuses on the performance of the explainer and how close the explainer f is to the black-box model b. *Fidelity* aims to evaluate this, in particular how good is f at mimicking the black-box decisions. There may be different specializations of fidelity, depending on the type of methods under analysis [36]. Rule-based explanations usually output a decision tree, a surrogate model which tries to mimic the original black-box. Fidelity uses standard accuracy to compare the prediction between the black-box and the surrogate model on the instances used to train the black-box. The more similar the two models are, the close the fidelity is to one.

Prototype and counterfactual explanations are the most challenging type of explanations to evaluate. Nevertheless, they need to guarantee some desirable properties. Bien et al. [45] postulated that a desirable prototype set for a class l would satisfy the following properties: cover as many training points as possible of the same class l, covers as few training points as possible of classes different from l, and must accurately represent the data. In other words, a good set of prototype explanations should cover as better as possible the possible categories in the data (*sparsity*) as well to be accurate to new instances (*accuracy*).

For Counterfactual explanations, the most widely used and shared desirable properties are: validity, minimality, similarity, plausibility, discriminative power, actionability, causality, and diversity. The most rigorous and inspiring works in this direction are Mothilal et al. [54], Verma et al. [75], and Guidotti et al. [9].

- *Validity.* A counterfactual x' is valid iff it actually changes the classification outcome with respect to the original one, i.e., $b(x') = b(x)$.

- *Minimality* (Sparsity). There should not be any other valid counterfactual example x'' such that the number of different attribute value pairs between x and x' is higher than the number of different attribute value pairs between x and x''.

- *Similarity* (Proximity). A counterfactual x' should be similar to x, i.e., given a distance function d in the domain of x, the distance between x and x' should be as small as possible.

- *Plausibility* (Reliability). Given a reference population X, a counterfactual x' is plausible if the feature values in x' are coherent with those in X. This practically means that the feature values of x' should not be higher/smaller than those observable in X, and that x' should not be labeled as an outlier with respect to the instances in X. Plausibility helps increase trust towards the explanation: it would be hard to trust a counterfactual if it suggests a combination of unrealistic features.

- *Actionability* (Feasibility). Given a set A of actionable features, i.e., features that can be mutated, a counterfactual x is actionable iif all the differences between x' and x refers only to actionable features that can be changed. Examples of non-actionable features that cannot be changed in a counterfactual are age, gender, race, etc. Indeed, a counterfactual should never change the non-actionable (immutable) features.

- *Diversity*. Let C be a set of (valid) counterfactuals for the instance x. Diverse counterfactuals should form the counterfactual explanation C, i.e., while every counterfactual $x' \in C$ should be minimal and similar to x, the difference among all the counterfactuals in C should be maximized [54, 76]. For instance, three (similar) counterfactuals saying that a yearly income of 15,000$, of 15,100$, and 14,800$ is going to change the outcome are less useful than three (different) counterfactuals saying that the outcome can be changed (i) with a yearly income of 15,000$, (ii) by owning a car, or (iii) by first paying back the other debts. Indeed, with the second set of diverse counterfactuals, there are more possible actions to change the classification outcome.

Finally, the last propriety we want from an explanation is to not change over subsequent runs or with slightly different inputs. *stability* aims at validating how stable the explanations are. It can be evaluated by exploiting the *Lipschitz constant* [74] as

$$L_x = \max \frac{\|e_x - e_{x'}\|}{\|x - x'\|}, \quad \forall\, x' \in \mathcal{N}_x$$

where x is the explained instance, e_x the explanation and $\mathcal{N}_x$ is a neighborhood of instances x' similar to x. Another way to measure stability is sensitivity. It has been shown that models with high explanation sensitivity are prone to adversarial attacks: Interpretation of Neural Networks is Fragile [77]. Explanation sensitivity [78] measures the extent of explanation change when the input is slightly perturbed. The sensitivity metric measures the maximum sensitivity of an explanation using the Monte Carlo sampling-based approximation. By default, it samples multiple data points from a subspace of an infinite sphere of a predefined radius. Note that the maximum sensitivity is similar to the Lipschitz [79] continuity metric, however, it is more robust and easier to estimate.

2.2 Benchmarking of XAI methods

The metric presented in the previous section can be used to assess the quality of explanations. However, each time a new method is proposed, some of the available metrics are exploited to evaluate the goodness of the explanations extracted, such as in [36, 80]. In addition, some authors also propose new metrics along with their methods of explanation. This thus leads to great difficulty in comparing explanations obtained from different explainers. For this reason, we evaluate the goodness of explanations obtained using the most popular explainers presented before using the same quantitative methodology. To achieve this goal, we compared the explanations obtained from applying different explainers, considering the different metrics in the literature. Given a dataset $\mathcal{D}_{\mathcal{L}}$ with labels $\mathcal{L}$, the methodology followed for comparing the different explanations is as follows:

1. Split the dataset $\mathcal{D}_{\mathcal{L}}$ into train and test, obtaining D_{train} with its labels L_{train} and D_{test} with its labels L_{test};

2. Define and train a black-box model b on the train set D_{train} and L_{train};

3. Test the black-box b on the test set D_{test}, obtaining $T_{pred} = b(D_{test})$;

3. Explain T_{pred}, the local predictions of b, using an explainer E, obtaining a set of explanations $\text{Exps} = E(b, D_{test}, T_{pred})$.

4. Apply the metrics available depending on the type of input data and the kind of explanation provided.

To compare the performance of the metrics, we adapted the Nemenyi test. For each dataset, we record the average ranking of explainers for a given metric and then run the Nemenyi test to see if one method is statistically better than another.

We divided the explainers into two categories based on the data type they handle: tabular explainers and image explainers.

Tabular Datasets: We consider three benchmark datasets for the tabular data: all of them have different characteristics that may affect the performance of the explainers. For all of them, we apply a standard pre-process: we replaced the categorical variables using a TargetEncoder, replaced the missing values using the mean (of median) of the column under analysis, and removed the outliers by visualizing the statistical distribution of the variables. We analyzed `adult`[5]: a binary classification with the task of predicting if a person earns more or less than 50K per year. It has 14 attributes (numerical and categorical) and 48842 records. Then, we considered `german`[6]: a binary classification for predicting the credit risk of a person. It has 20 attributes, mostly categorical, with 1000 records. Lastly, `compass`[7]: a multi-class dataset, in which the goal is to predict the recidivism of a convicted

[5]`adult`: https://archive.ics.uci.edu/ml/datasets/adult
[6]`german`: https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)
[7]`compass`: https://www.kaggle.com/datasets/danofer/compass

	adult			german			compass			mnist	cifar	imagenet
black-box	LG	XGB	CAT	LG	XGB	CAT	LG	XGB	CAT	CNN	CNN	VGG16
F1-score	0.65	0.82	0.80	0.66	0.75	0.79	0.63	0.69	0.68	0.99	0.74	0.76

Table 2.1: We report here the weighted F1 score for the various black-boxes.

Table 2.2: Comparison on fidelity and faithfulness of the explanation methods. We report the mean and the standard deviation over a subset of 50 test set records.

Dataset	**Black-Box**	**Fidelity**					**Faithfulness**		
		LIME	SHAP	DALEX	ANCHOR	LORE	LIME	SHAP	DALEX
adult	LG	0.98 (0.21)	0.61 (0.43)	0.35 (0.03)	**0.99** (0.05)	0.98 (0.03)	0.10 (0.30)	**0.38** (0.37)	0.08 (0.03)
	XGB	**0.98** (0.03)	0.88 (0.02)	0.64 (0.07)	**0.98** (0.03)	**0.98** (0.04)	0.03 (0.32)	**0.36** (0.49)	0.27 (0.31)
	CAT	0.96 (0.32)	0.78 (0.51)	0.70 (0.15)	**0.99** (0.21)	0.98 (0.43)	0.10 (0.32)	**0.44** (0.37)	0.11 (0.30)
german	LG	**0.98** (0.06)	0.91 (0.23)	0.57 (0.21)	0.73 (0.09)	**0.98** (0.12)	**0.23** (0.60)	0.19 (0.63)	0.20 (0.03)
	XGB	**0.99** (0.10)	0.82 (0.02)	0.65 (0.03)	0.80 (0.03)	0.98 (0.21)	0.16 (0.26)	**0.44** (0.21)	0.31 (0.09)
	CAT	**0.98** (0.05)	0.67 (0.12)	0.63 (0.09)	0.62 (0.31)	**0.98** (0.35)	0.34 (0.33)	**0.43** (0.32)	0.33 (0.12)
compass	LG	**0.95** (0.31)	0.83 (0.41)	0.23 (0.03)	0.53 (0.46)	0.82 (0.03)	0.12 (0.56)	**0.41** (0.54)	0.11 (0.08)
	XGB	**0.97** (0.21)	0.43 (0.33)	0.45 (0.23)	0.67 (0.42)	0.87 (0.03)	0.19 (0.44)	**0.56** (0.38)	0.13 (0.13)
	CAT	**0.98** (0.27)	0.54 (0.10)	0.55 (0.30)	0.22 (0.92)	0.81 (0.02)	0.22 (0.42)	**0.57** (0.32)	0.18 (0.07)

person, with 3 classes of risk recidivism. It has 21800 records and 10 variables, all of them categorical except *age*.

Tabular Black-Boxes and Explainers: For comparing the tabular explainers, we define and train 3 ML models, for each dataset: a *Logistic Regression* (LG), then *XGBoost*[8] (XGB), and *Catboost*[9] (CAT). The performances of the black-box models are reported in Table 2.1[10]. For validating the explanations on tabular data, we refer to seven explanation methods already presented in Section 2.1.8. For feature importance, we considered LIME with 5000 synthetic samples to generate for each record to explain, SHAP, and DALEX with the *break down* method. The results obtained from the applications of these metrics are reported in Table 2.2 for fidelity and faithfulness, while in Table 2.3, we report the stability. The monotonicity is not reported since, for every method, it was *False*, showing that no method complies with this requirement.

In Figure 2.7, we report an overall ranking evaluation of the explanation methods

[8]https://xgboost.readthedocs.io/en/stable/
[9]https://catboost.ai/
[10]The dataset was split into train and test with ratio 80% – 20%

Table 2.3: Comparison on the stability metric. We report the mean and the standard deviation over a subset of 50 test records.

Dataset	Black-Box	Stability				
		LIME	SHAP	DALEX	ANCHOR	LORE
adult	LG	24.37 (2.74)	1.52 (4.49)	5.40 (0.10)	**22.36** (8.37)	21.76 (11.80)
	XGB	10.16 (6.48)	2.17 (2.18)	6.00 (0.06)	26.53 (13.08)	**30.01** (20.52)
	CAT	0.35 (0.43)	0.03 (0.01)	4.3 (0.04)	6.51 (4.40)	**27.80** (70.05)
german	LG	18.8 (0.73)	19.01 (23.4)	12.54 (0.05)	101.0 (62.7)	**622.1** (256.7)
	XGB	26.08 (14.5)	38.43 (30.6)	5.12 (0.10)	121.4 (98.4)	**725.8** (337.2)
	CAT	2.49 (9.91)	15.92 (10.71)	3.54 (0.9)	123.7 (76.86)	**756.7** (348.2)
compass	LG	0.51 (0.21)	0.54 (0.10)	11.42 (19.24)	112 (23.52)	**321.3** (261.4)
	XGB	0.676 (0.30)	13.67 (21.64)	6.00 (0.06)	97.20 (18.04)	**229.1** (39.61)
	CAT	2.49 (9.91)	14.22 (10.01)	4.33 (0.04)	100.7 (60.60)	**526.9** (341.5)

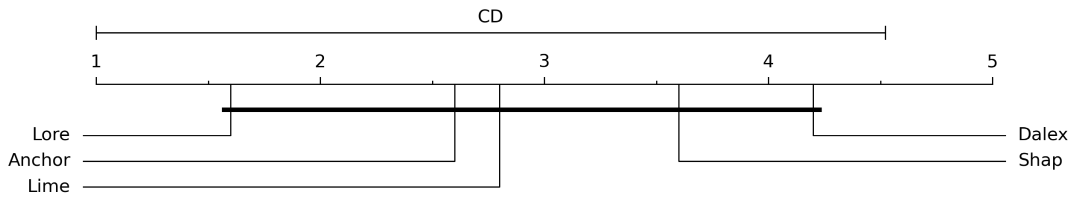


Figure 2.7: Critical difference plot for Nemenyi test ($\alpha = 0.05$). We compare the tabular explanations in terms of fidelity and stability computable for all the explanation kinds.

in terms of *fidelity* and *stability*. From this plot, we can clearly see that LORE and ANCHOR, which are the rule-based methods, perform better than the feature importance ones. This result is particularly interesting because feature importance methods are more studied than logical explanations, even though the latter is more similar to human thinking [81]. Our experiments show that rule-based methods have high fidelity, correctly replicating the black-box behavior. This fact is also highlighted by the stability results, which are extremely good for LORE, followed by ANCHOR. Regarding the feature importance methods, LIME also has excellent fidelity, but unfortunately, this method suffers in terms of stability due to its random generation of the neighborhood. SHAP and DALEX, instead, do not exhibit good fidelity but are better in terms of stability w.r.t. LIME. Finally, in Table 2.2, we present faithfulness. SHAP achieves the best results, being the metrics with values between -1 and 1. However, we remark that none of the methods reached optimality.

Table 2.4: Insertion (left) and deletion (right) metrics expressed as AUC of accuracy vs. percentage of removed/inserted pixels. The reported value represents the mean of the scores obtained on a subset of 100 instances of the dataset, and the value on the parenthesis is the standard deviation. Best results are highlighted in bold, and second best results are underlined.

	Insertion			**Deletion**		
	`mnist`	`cifar`	`imagenet`	`mnist`	`cifar`	`imagenet`
LIME	0.807 (0.14)	0.41 (0.21)	0.34 (0.25)	0.388 (0.21)	0.221 (0.19)	0.051 (0.05)
DEEP-SHAP	**0.981** (0.01)	0.32 (0.28)	0.25 (0.22)	0.182 (0.18)	0.187 (0.32)	0.098 (0.09)
GRAD-SHAP	<u>0.980</u> (0.01)	0.46 (0.24)	0.35 (0.24)	0.188 (0.19)	0.153 (0.24)	0.056 (0.07)
ϵ-LRP	0.976 (0.02)	0.56 (0.20)	0.28 (0.19)	**0.120** (0.01)	0.127 (0.11)	**0.014** (0.02)
INTGRAD	0.975 (0.03)	**0.64** (0.22)	0.37 (0.23)	<u>0.128</u> (0.01)	**0.118** (0.07)	<u>0.019</u> (0.04)
DEEPLIFT	0.976 (0.02)	0.57 (0.20)	0.28 (0.19)	**0.120** (0.01)	0.127 (0.11)	**0.014** (0.02)
SMOOTHGRAD	0.959 (0.03)	0.55 (0.23)	0.34 (0.26)	0.135 (0.04)	0.153 (0.13)	0.033 (0.05)
XRAI	0.956 (0.04)	0.58 (0.21)	<u>0.40</u> (0.26)	0.151 (0.04)	0.144 (0.07)	0.086 (0.11)
GRAD-CAM	0.941 (0.04)	0.57 (0.20)	0.21 (0.19)	0.297 (0.20)	0.153 (0.12)	0.139 (0.12)
GRAD-CAM++	0.941 (0.04)	0.52 (0.22)	0.32 (0.26)	0.252 (0.13)	0.283 (0.24)	0.081 (0.10)
RISE	0.978 (0.03)	<u>0.61</u> (0.21)	**0.50** (0.26)	**0.120** (0.01)	<u>0.124</u> (0.07)	0.044 (0.05)

Nevertheless, SHAP turns out to be the best in this context, followed by DALEX and LIME.

Image Datasets: For the experiments on images, we considered three datasets. The handwritten number classification dataset `mnist`[11]. It has ten classes, from 0 to 9, and the images are low resolution (28x28) and greyscale. Then, `cifar`[12]: low resolution (32x32) color images dataset with ten classes, ranging from dogs to airplanes. Lastly, `imagenet`[13]: composed of high resolution color images (224x224), with a 1000 classes. We chose these datasets because they are the most utilized, and we have different classes with various image dimensions.

Image Black-Boxes and Explainers: On these three datasets, we trained the models most used in literature to evaluate the explanation methods: for `mnist` and `cifar` we trained a convolutional neural network with two convolutions and two linear layers, while for `imagenet` we decided to use the VGG16 network [82]. The performances of the black-box models are reported in Table 2.1. For the LIME

[11]http://yann.lecun.com/exdb/mnist/
[12]http://image-net.org/
[13]https://www.cs.toronto.edu/ kriz/cifar.html

Table 2.5: Sensitivity metric and runtime results, the lower, the better. The best results are highlighted in bold, second best results are underlined. The reported value represents the mean of the scores obtained on a subset of 100 instances of the dataset, and the value on the parenthesis is the standard deviation. Runtime is expressed in seconds, and uncertainty is on the last decimal.

	Sensitivity			Runtime		
	mnist	cifar	imagenet	mnist	cifar	imagenet
LIME	2.509 (1.261)	1.529 (2.176)	2.090 (0.612)	1.9	10	50
DEEP-SHAP	0.198 (0.071)	1.649 (1.054)	0.089 (0.189)	4.4	5.2	8.4
GRAD-SHAP	0.615 (0.099)	1.986 (0.931)	0.153 (0.357)	3.1	4.2	6.5
ϵ-LRP	0.394 (0.113)	2.311 (0.752)	0.207 (0.806)	1.5	1.3	2.1
INTGRAD	0.262 (0.121)	1.851 (1.063)	0.131 (0.738)	**0.03**	**0.06**	5.01
DEEPLIFT	0.293 (0.132)	2.272 (1.039)	<u>0.055</u> (0.010)	2.2	1.3	3.2
SMOOTHGRAD	9.498 (5.847)	1.367 (0.506)	1.829 (0.350)	<u>0.04</u>	<u>0.07</u>	0.8
XRAI	2.256 (0.512)	1.072 (0.621)	0.310 (0.225)	1.1	1.5	18
GRAD-CAM	0.605 (1.519)	0.877 (1.110)	0.093 (0.592)	0.1	0.15	**0.25**
GRAD-CAM++	<u>0.132</u> (0.165)	**0.339** (0.537)	**0.047** (0.292)	0.1	0.15	**0.25**
RISE	**0.117** (0.041)	<u>0.501</u> (1.310)	0.501 (0.461)	0.5	2.3	21.4

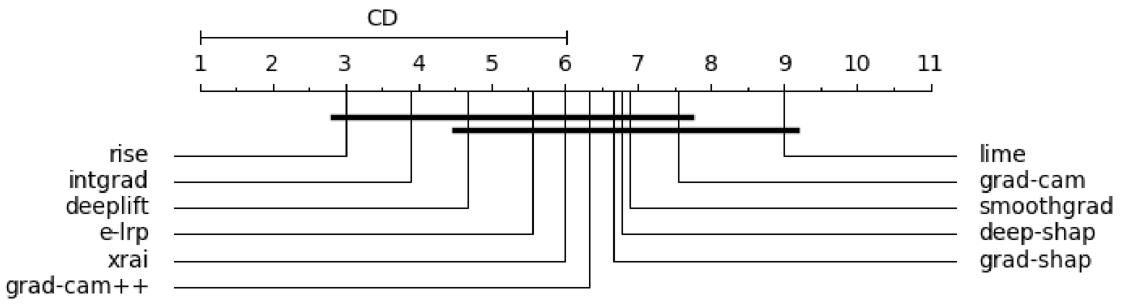


Figure 2.8: Critical difference plot for Nemenyi test with $\alpha = 0.05$.

segmentation, we used the quickshift algorithm [83] with a neighborhood size of 2000. In INTGRAD, XRAI, and DEEPLIFT, we used a black image as the background. For DEEP-SHAP and GRAD-SHAP, 100 images are taken randomly from the training set and used to approximate the Shapley values. In GRAD-CAM and GRAD-CAM++, the last convolutional layer was selected from which to calculate the gradients. For the masking of RISE, we used 2000 masks generated randomly. Deletion/Insertion results are reported in Table 2.4 and the Sensitivity results in Table 2.5.

For image data, the best method, in general, is RISE. However, as highlighted in Figure 2.8, none of the methods has statistical significance to be considered better than the rest. All the methods are very noisy and unstable, as pointed out from the stability and the high standard deviation among all the methods in the deletion/insertion metrics. LIME and XRAI suffers from stability issues due to the randomness of the segmentation pre-processing. LIME is also the worst method when measuring accuracy. Guided methods like SMOOTHGRAD are even worst than random methods when computing the stability of the explanations. We support the findings of [14] in which they pointed out that guided methods are not good explainers. SMOOTHGRAD is okay in high-resolution images, but this is caused by the fact that the guided perturbation plays an inferior role than the gradient computation. In general, gradient approaches like INTGRAD and DEEPLIFT are the best approaches for accuracy, especially when dealing with high-resolution images. The computation is fast and stable, even if we compute the second-order gradients like in GRAD-CAM++. INTGRAD and DEEPLIFT are more precise than GRAD-CAM and GRAD-CAM++ since the saliency maps produced by these last two methods are coarse and unrefined. SHAP based methods work only on low-resolution images due to the approximation factor. The higher the resolution, the more images are needed as background to better approximate the Shapley values. However, in doing this, the memory used and the runtime increase exponentially. RISE is the best compromise and can reach a high level of accuracy and stability, even if it is based on random masking.

2.3 The Latent Space

Real-world data are often redundant and large in size. This poses challenges not only for computational efficiency but also for representation modeling. Consider, for example, the Swiss roll in figure 2.9. The data is in three dimensions, but only two dimensions are sufficient to represent the same object when we unroll it. This is called dimensionality reduction by manifold learning. The basic assumption is that high-dimensionality data often have lower dimensionality embedding that is sufficient to represent the content of the original data.

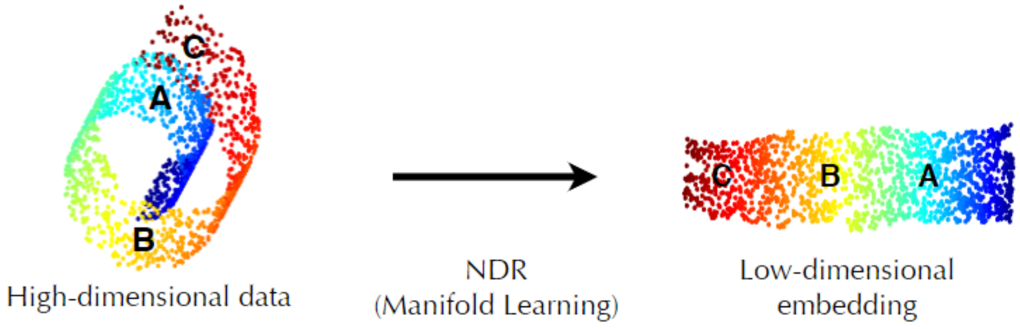


Figure 2.9: An example of manifold dimensionality reduction. (Image courtesy of golden.com/wiki)

Now, if we extend this concept to the data representation problem, we realize that there is a lower dimensional space sufficient to describe the contents of our dataset. We call such a space "latent space". It is a low-dimensional manifold of high-dimensional data in which we expect all instances of the dataset to be in close proximity. Two main categories of algorithms are capable of obtaining such space: dimensionality reduction methods and generative models. Dimensionality reduction methods create features with simple combinations of original features, while generative models incorporate non-linear relationships. Both types of models seek to incorporate data information in a latent space as reduced as possible.

2.3.1 Dimensionality Reduction Techniques

Dimensionality reduction is defined as the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. [84] Basically, it transforms our data from a higher-dimensional space into a lower-dimensional space while preserving as much information as possible.

Dimensionality reduction techniques were born for visualization. Data visualization boosts the ability to process information in an easy and faster way to compare and make conclusions out of it. However, as humans, the higher the dimension, the more difficult it is to analyze the data and derive a conclusion since we can see three dimensions at a time maximum. For N dimensional data, we need N dimensions, which we can only visualize two or three at a time, and in our big-data era, with data with a load of features, it is not feasible. It might take us days or months to perform any meaningful analysis, which requires much time. So it is common to reduce data dimension to 2 or 3 dimensions using techniques called dimensionality reduction.

Reducing the dimension of the data is beneficial not only for visualization but also for learning. In machine learning, it is believed that the greater the number of features, the better our prediction, but this is not always true. If we keep on increasing the number of features after a certain point, the performance of our machine learning algorithm tends to decrease. This phenomenon is called *Curse of Dimensionality*, and it describes the explosive nature of increasing data dimensions and its resulting exponential increase in computational efforts required for its processing and analysis. This term was first introduced by Richard E. Bellman [85] to explain the increase in the volume of Euclidean space associated with adding extra dimensions in the area of dynamic programming. Applied to machine learning, this means that an increase in the dimensions can, in theory, add more information to the data, thereby improving the quality of data but practically increasing the noise and redundancy during its analysis. As the dimensionality increases, the number of data points required for good performance of any machine learning algorithm increases exponentially. The reason is that we would need more data points for any given combination of features for any machine learning model to be valid. For exam-

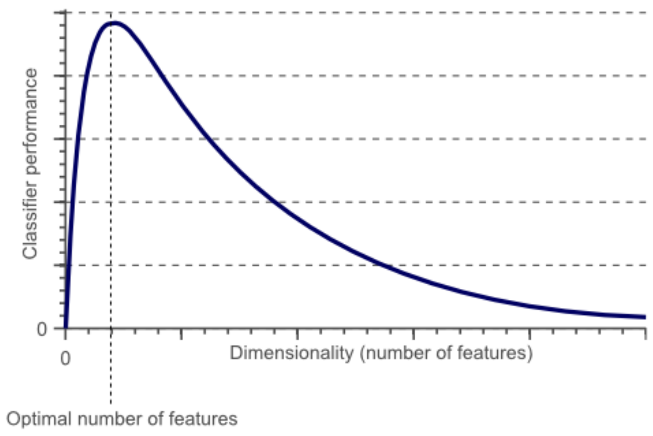


Figure 2.10: Graphical representation of Hughes principle

ple, suppose a model needs at least 10 data points for each combination of feature values to perform well. If we assume that we have one binary feature, then for its 2 unique values (0 and 1), we would need $2^1 \cdot 10 = 20$ data points. For 2 binary features, we would have 2^2 unique values and need $2^2 \cdot 10 = 40$ data points. Thus, for k-number of binary features, we would need $2^k \cdot 10$ data points. Hughes (1968) [86], in his study, concluded that with a fixed number of training samples, the predictive power of any classifier first increases as the number of dimensions increases, but after a specific value of dimensions, the performance deteriorates. Thus, the curse of dimensionality is also known as the Hughes phenomenon.

Reducing the number of features in the training data helps reduce this effect and offers other benefits as well:

- Minimize the space required to store the data.
- Fewer dimensions will take low time complexity in training a model.
- As dimensions decrease, the possibility of overfitting the model also decreases.
- It will remove all the correlated features in our data.

Many techniques were developed to transform the data from a high-dimensional space to a low one.

The Principal Component Analysis (PCA) [87] is one of the dimensionality reduction techniques widely used in Machine Learning. The main idea behind PCA is to figure out patterns and correlations among various features in the data set. On finding a strong correlation between different variables, a final decision is made about reducing the dimensions of the data in such a way that the significant data is retained. The new variables obtained from the initial set are called principal components. Principal components are calculated so that the newly obtained variables are highly significant and independent. Specifically, the principal components are the eigenvectors of the covariance matrix of the data. By calculating the eigenvectors of the covariance matrix, we can identify the directions of maximum variance of the data and, thus, the directions of maximum information. There could be more than one dimension where the information is spread. By ordering the principal components by their eigenvalues is possible to obtain a ranking from the most important dimensions to the less ones. Only components that account for about 90% variance of the data are kept.

After PCA, LDA is the most commonly used dimensionality reduction technique in machine learning, statistics, and pattern recognition. The goal of this algorithm is always to project a dataset to a lower dimensional space, but with a separability not only on the features of the data but also on the categories in order to avoid overfitting and to reduce the computational power. Both PCA and LDA are linear reduction techniques, but unlike PCA, LDA focuses on maximizing the separability of groups. LDA uses features to create new dimensions and tries to project the data onto new axes to maximize the separation of the data and the categories or groups. This is why LDA is a supervised learning algorithm since it uses target values to find the new axes. PCA tries to find the components that maximize the variance, while on the other hand, LDA tries to find the one that maximizes the separability of the categories and minimizes the variance among categories.

Another way to aggregate data is by similarity. The method of t-distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction method used to visualize data in 2D and 3D maps. t-SNE finds non-linear connections between the data based on their similarity. The more similar are two instances, the more probable it is to be close in the reduced space. The first stage of the algorithm is calculating the Euclidean distances of each point from all of the other points. Then, it takes these distances and transforms them into conditional probabilities representing the similarity between every two points. The conditional probability $p_{i|j}$ of point x_j to be next to point x_i is represented by a Gaussian centered at x_i with a standard deviation of $sigma_i$. $p_{i|j}$ is therefore the probability that i would pick j as its neighbor: the more similar i is to j, the higher the probability. After computing the joint probability $p_{ij} = \frac{p_{j|i}+pi|j}{2n}$, t-SNE creates a dataset of random points in a low-dimensional space and calculates a joint probability distribution for them as well. Now we use the Kullback-Leiber (KL) divergence to make the joint probability distribution of the data points in the low dimension as similar as possible to the one from the original dataset. This is done by using gradient descent. The cost function that the gradient descent tries to minimize is the KL divergence of the joint probability distribution between the high and low-dimensional space. From this optimization, we get the values of the points in the low-dimension dataset and use them for our visualization. This loss is difficult to optimize and requires significant use of computational resources. Another problem of t-SNE is the restriction of the reduced dimensions. Due to the exponential computational costs of calculating KL divergence, dimensions larger than three are prohibitive from a running time perspective. For this reason, t-SNE is always used as a visualization technique rather than a dimensionality reduction one.

A very similar algorithm to t-SNE is UMAP. Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualization similarly to t-SNE but also for general non-linear dimension reduction.

The improvement of UMAP over t-SNE is to preserve more of the global structure of the data with superior run time performance. Furthermore, UMAP has no computational restrictions on embedding dimensions, making it viable as a general-purpose dimension reduction technique for machine learning. UMAP uses local manifold approximations to construct a topological representation of the high dimensional data. A similar process can be used to construct an equivalent topological representation given some low-dimensional data representation. UMAP then optimizes the layout of the data representation in the low dimensional space to minimize the cross-entropy between the two topological representations. It does that by building a graph of the data in the original space, and it tries to preserve its proprieties in the reduced space. Due to this topological preservation, UMAP can generate better reduced space than t-SNE. However, UMAP is not perfect, and it comes with its vulnerabilities. One of the core assumptions of UMAP is that there exists a manifold structure in the data. Because of this, UMAP can tend to find various structures within the noise of a dataset, similar to the way the human mind finds structured constellations among the stars. As more data are sampled, the number of structures evident from noise will tend to decrease, and UMAP becomes more robust. However, care must be taken with small sample sizes of noisy data or data with only large-scale manifold structures.

t-SNE considers the distance between data pairs, but this does not preserve the global structure of the data. TRIMAP is a dimensionality reduction method that uses triplet constraints to form a low-dimensional embedding of a set of points. The triplet constraints are of the form "point i is closer to point j than point k". By using triplets, TRIMAP provides a significantly better global view of the data than the other dimensionality reduction methods, such as t-SNE and UMAP. The global structure includes relative distances of the clusters, multiple scales in the data, and the existence of possible outliers. The triplets are sampled from the high-dimensional representation of the points, and a weighting scheme is used to reflect the importance of each triplet. TRIMAP defines a global score to quantify the quality of an embedding in reflecting the global structure of the data. One major problem of this approach is that it highly depends on initialization.

While the non-linearity of dimensionality reduction methods can improve data representation, at the same time, reduced dimensions lose their meaning. In particular, the dimensions of the UMAP, t-SNE, and TRIMAP latent spaces have no specific meaning, unlike PCA, where the dimensions are the directions of greatest variance in the source data. For several use cases, the interpretability of the reduced dimension is of critical importance.

2.3.2 Generative models

When speaking about learning, there are two types of approaches on how a model learns: discriminative models and generative models. The first approach tries to learn only the unique features that help distinguish between different classes in training data. This class belongs mostly to machine learning classification algorithms. They try to estimate $P(y|x)$ that is the probability of a class y given an instance x.

On the other hand, generative models try to learn the distribution of the data rather than identifying unique features that can help distinguish different classes. They learn a latent space in which they place data: the more similar two data are, the more likely they are to be close in latent space. In this way, such models actually understand the data distribution and offer a significant advantage over discriminative models: data generation. By determining the distribution from which the training samples were generated, new data can be generated by sampling from this distributions[14].

However, they have their weaknesses. Compared with a discriminative model,

[14]Estimating a distribution mean to estimate distribution parameters (like mean and standard deviation)

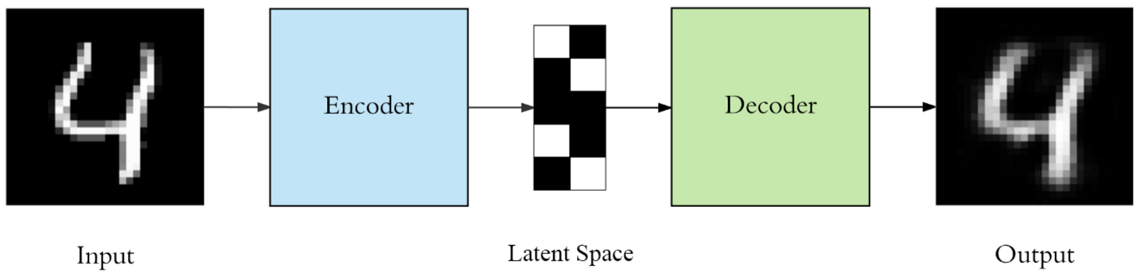


Figure 2.11: Standard architecture of an autoencoder generative model

quantitatively evaluating the model performance of generated data is challenging, so they are difficult to train. Discriminative models can be quickly evaluated using standard metrics such as accuracy and F1 score on the labels of the dataset. Generative models, instead, generate new data that has to be similar to the one in the training dataset. They usually minimize different types of losses to ensure different proprieties of the generated data.

The world mainly focuses on discriminative modeling because it fits well with real-world problems. Rarely do problems need to know the distribution from which samples were generated to generate artificial samples, but instead, they need to classify new samples given the training data, thus a classification task. Nevertheless, recent advances in image and text generation have revived generative modeling, which is now growing and finding many new applications.

Autoencoders Autoencoders are the most common type of generative models. They are composed of two parts: An *encoder* and a *decoder* (Figure 2.11). The encoder takes as input the raw data (x) and outputs a vector (z) in a reduced space called the latent space. The decoder takes the vector z in the latent space as input and tries reconstructing the data in the original space. The encoder and the decoder are usually machine learning models, particularly neural networks. Standard neural networks called feed-forward are usually used, while convolutional layers are preferred when using complex data, such as images. The dimension of the vector

z is usually lower than x since we want z to capture only the meaningful factors of variations that can describe the input data. It needs to be trained to work as with all machine learning models. The goodness of autoencoders lies in their ability to reconstruct data from their latent representations to their original features. A perfect autoencoder should have the output of its decoder identical to the input of the encoder. The loss usually employed is the metric that computes the difference between the original and reconstructed data. Data is then generated by randomly taking data in the latent space and decoding it with the decoder.

When building an autoencoder, it is important that it does not store all the information. Instead, it should be constrained to prioritize which information should be kept and which should be discarded. This constraint is introduced in the following ways:

- number of layers: It is possible to keep as many layers in the encoder and decoder as you require. Usually, the number of nodes decreases as it increases the number of layers in the encoder and vice-versa for the decoder.

- number of nodes in the latent layer: It is always better to have less number of nodes in this layer than the input size. The smaller size of the latent layer leads to better compression.

- loss: The mean squared error or binary cross entropy are usually chosen for the loss function.

To improve the latent representation, several autoencoder variations exist. The most famous one is called Variational AutoEncoder (VAE). A Variational Autoencoder [88] can be defined as an autoencoder whose training is regularized to ensure that the latent space has suitable properties that enable the generative process. Like a standard autoencoder, a variational autoencoder is an architecture composed of both an encoder and a decoder. It is trained to minimize the reconstruction error between the encoded-decoded data and the initial data. However, in order to introduce some regularization of the latent space, there is a slight modification of the encoding-decoding process: instead of making the encoder output an encoding vector of size n, we output two vectors of size n: a vector of means and another vector of standard deviations. VAE encodes the inputs as a mixture of Gaussian distributions. To decode back from the latent space, it is possible to sample from these distributions and then use the decoder. This modification adds a stochastic factor to the model that helps the sparsity of generated data. The loss function that is minimized when training a VAE is composed of a "reconstruction term" and a "regularization term" (on the latent layer). The regularization term is expressed as the Kullback-Leibler divergence between the returned distribution and a standard Gaussian. With this regularization term, we prevent the model from encoding data far apart in the latent space and encourage returning distributions to "overlap" as much as possible. Naturally, as for any regularization term, this comes at the price of a higher reconstruction error on the training data. Training a VAE is a trade-off between the reconstruction error and the KL divergence. The admirable propriety of VAE is that the introduction of regularization tends to create a "gradient" over the information encoded in the latent space. For example, a point of the latent space that would be halfway between the means of two encoded distributions coming from different training data should be decoded in something that is somewhere between the data that gave the first distribution and the data that gave the second distribution as the autoencoder may sample it in both cases.

Several other variations exist. β-VAE [89] is an augmenting of the original VAE framework with a single hyper-parameter β that modulates the learning constraints applied to the model. These constraints limit the capacity of the latent information channel and control the emphasis on learning statistically independent latent factors. β-VAE with $\beta = 1$ corresponds to the original VAE framework. With $\beta > 1$, the model is pushed to learn a more efficient latent representation of the data. Adversarial Autoencoders are a modification of the original architecture of autoencoders. The encoder's output is used not only to feed the decoder but also as input to a discriminative model. The input of the discriminative model is the concatenation of the label and the latent space of the autoencoder. Training is performed by minimizing the autoencoder loss and a cross-entropy term between the data label and the discriminative model's output. The idea behind adversarial autoencoders is that it is possible to train the encoder to produce a latent space that looks like a distribution of choice. This is achieved with a different neural network model that learns to distinguish the output of a network from a real-world example of the target. However, instead of learning to tell noise from real cat pictures, the discriminator learns to tell the latent space from the chosen distribution and gives feedback to the encoder about how unfair the distribution of the latent space is. The encoder learns to distribute the code as desired from this feedback in the form of a gradient.

Generative Adversarial Networks. is the other generative model that has recently gained popularity. A Generative Adversarial Network (GAN) [90] observes the given dataset and generates new samples that fit the underlying distribution in the given data samples with the help of two models, which are adversaries of each other. Hence the name Generative Adversarial Networks. A generative adversarial network (GAN) consists of two components:

- The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator.

- The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input (Figure 2.12). Through back-

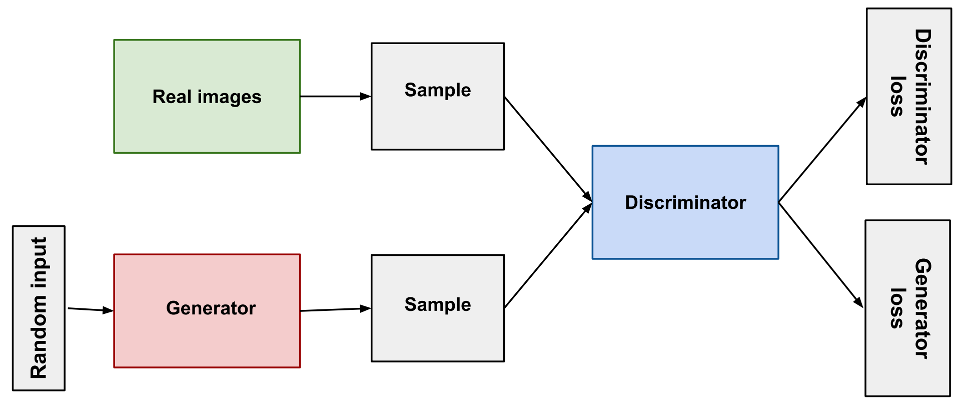


Figure 2.12: Architecture of GAN model

propagation, the discriminator's classification provides a signal that the generator uses to update its weights.

Two steps compose the training: firstly, the discriminator is trained, and then later, the generator. The following steps compose the Discriminator training:

1. The discriminator classifies both real data and fake data from the generator.
2. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
3. The discriminator updates its weights through back-propagation from the discriminator loss through the discriminator network.

After it is time for the Generator Training Procedure:

1. Sample random noise.
2. Produce generator output from sampled random noise.
3. Get discriminator "Real" or "Fake" classification for generator output.
4. Compute loss from discriminator classification.

5. Back propagates through both the discriminator and generator to obtain gradients.

6. Use gradients to change only the generator weights.

As the generator improves with training, the discriminator performance gets worse because the discriminator cannot easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. In effect, the discriminator flips a coin to make its prediction. This progression poses a problem for the convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its quality may collapse. Convergence is often a fleeting rather than a stable state for a GAN. As a result, the generator model maps the latent input vector (z) to an output similar to the training dataset's data. The primary objective of GANs was to generate new samples from the given dataset. Moreover, since the invention of GANs, they have grown to accomplish this task with better results and added features.

In recent years several GAN variation has been introduced. Progressive GAN was introduced in 2017 [91], where the authors showed that the quality of the generated images could be increased significantly by training the generator to output low-resolution images at the beginning and increase the resolution as the training went on. Regular GANs adopt the sigmoid cross-entropy loss function. This loss function will cause the problem of vanishing gradients for the samples on the right side of the decision boundary but still far from the actual data. LSGAN [92] will use Least Square loss instead of binary cross-entropy loss for both the generator and the discriminator. Wasserstein GAN [93] is another variation on Discriminator loss; this time, the loss used is the Wasserstein one. Using this loss, the discriminator does not classify instances. For each instance, it outputs a number. This number

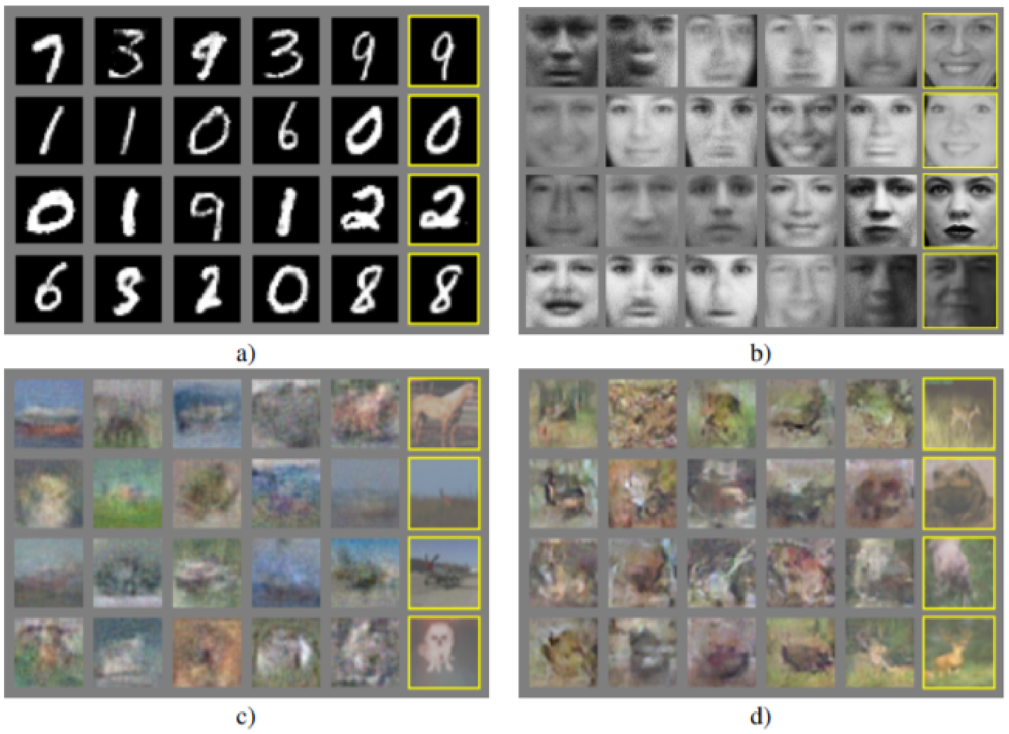


Figure 2.13: Examples of images generated by GANs

does not have to be less than one or greater than zero, so we cannot use 0.5 as a threshold to decide whether an instance is real or fake. Discriminator training makes the output bigger for real instances than for fake ones. Several other variation are described here [94, 95, 96, 97, 98, 99]

2.3.3 Understanding the Latent space

Generative models, as the name says, have the goal of generating multiple samples starting from the dataset one provides. However, since auto-encoders learn the natural feature of the dataset in the latent space, this space could be meaningful for understanding the data. Several works have demonstrated the performance capabilities of the latent space in doing clustering [100], and sometimes these clusters are even better than doing it in the input space [101]. Nevertheless, performance is not all; researchers have recently focused on latent space explorations and the

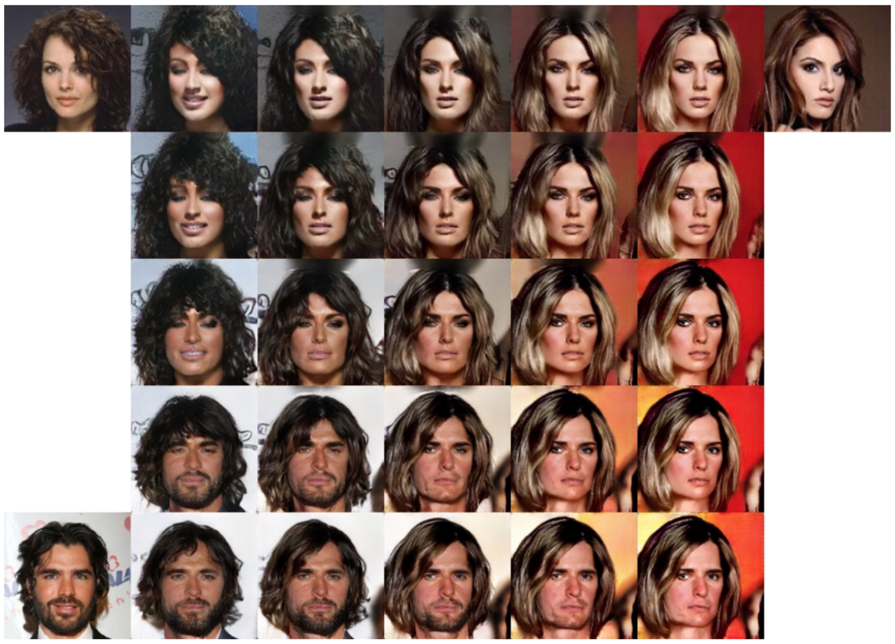


Figure 2.14: J-Diagram. The three corner images are inputs to the system, with the top left being the ”source” (A) and the other two being ”analogy targets” (B and C). Adjacent to each figure is the reconstruction resulting from running images through both the encoder and decoder of the model. The bottom right image shows the result of applying the analogy operation (B + C) − A. All other images are interpolations.

possible implications. Latent space features are easy to visualize because it is easier to sample data from it and use the decoder to represent the data in the original input space. In image recognition tasks, some dimensions of the latent space can be attributed to the position of the object in the image [102]. Interpolation is used to traverse between two known locations in latent space. Research on generative models often uses interpolation to demonstrate that a generative model has not simply memorized the training examples [103]. In creative applications, interpolations can provide smooth transitions between two decoded images [104]. For example, it is possible to move from different pictures of faces, Figure 2.14, or from different sketches [105] or fonts [106] or even molecules [107].

These examples show us the great potential of latent space reasoning, but since an autoencoder reduces many variables to just a few variables, many aspects of system behavior can be deduced. For example, it is possible to take the average of all the latent vectors for user-specified features, woman pictures, for example, and the average for all the user-specified non-woman pictures. We then compute the difference between these two average vectors and obtain the so-called gender vector. To change the gender of an image, add the gender vector to the corresponding latent vector; the amount of gender vector added controls how much we want to change the gender. This technique was introduced by [108], and vectors like the gender vector are sometimes called attribute vectors. Moreover, this reasoning is expandable to every possible feature one can think of.

However, autoencoders are not perfect, and while it is great that such profound principles as gender can be inferred, sometimes such models infer other things that are wrong or undesirable. For example, [104] the addition of a smile vector in some face models will make faces smile more and appear more feminine. Why? Because in the training data, more women than men were smiling. So these models may not just learn in-depth facts about the world but also internalize prejudices or erroneous beliefs. Once such a bias is known, it is often possible to make corrections. However, finding those biases requires careful auditing of the models, and it still needs to be made clear how these audits can be exhaustive. More broadly, why do attribute vectors work, when they work, and when they fail? At the moment, the answers to these questions need to be better understood.

Another way of understanding the latent space is by interaction [109]. Interaction can be a powerful tool for producing explanations of machine learning models. By allowing users to interact with the model and explore its behavior, interaction can provide a more intuitive and accessible way to understand the underlying concepts and patterns. One way interaction can be used to produce explanations is through interactive visualizations. These visualizations allow users to explore the relationships between different features and characteristics of the data, and to see how the model responds to changes in these features. For example, users can manipulate sliders or other controls to adjust the input data and observe how the model output changes in response. Another way interaction can be used is through interactive natural language processing (NLP) interfaces [110]. These interfaces allow users to interact with the model through natural language queries, providing a more intuitive and accessible way to understand the model's behavior. For example, users can ask the model to explain why it made a certain prediction or to provide more details on the factors that influenced its decision. In addition to these techniques, interaction can also be used to facilitate collaboration between humans and machine learning models [111]. By allowing users to provide feedback and input into the model's behavior, interaction can help to improve the accuracy and interpretability of the model and to ensure that it is aligned with the needs and goals of its users.

Interactive generative adversarial networks, or iGANs, [112] are another example of latent space exploration. The final result is utilizing these variations of GAN in an interface to generate images of different things, like shoes, t-shirts, and landscapes 2.15. Conventionally, such an interface would require the programmer to write a program containing a great deal of knowledge about the image. Instead of doing this, Zhu et al. train a GAN model using 5050 thousand images downloaded from the internet. They then use that generative model to build an interface that lets a user roughly sketch the shape of the object, and the model will reproduce the object most similar to the sketch.

Rather than using variational autoencoders, these interfaces are based on GANs. However, the underlying idea is still to find a low-dimensional latent space that can represent (say) all landscape images and map that latent space to a corresponding image. Again, we can think of points in the latent space as a compact way of describing landscape images. Roughly speaking, the way the iGANs work is as follows. Whatever the current image is, it corresponds to some point in the latent space. The interface works by finding a point in the latent space near the current image so the image is not changed too much but also comes close to satisfying the

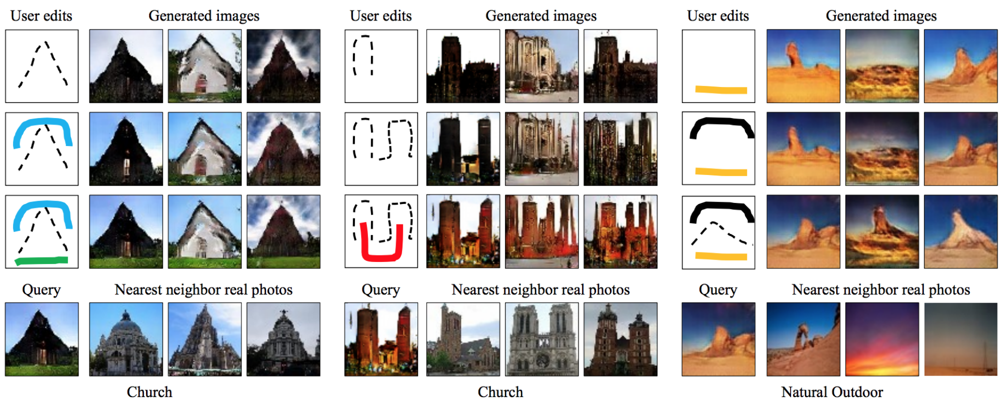


Figure 2.15: Interactive image generation framework example. The user uses the brush tools to generate an image from scratch (top row) and then adds more scribbles to refine the result (second and third rows). In the last row, we show the most similar real images to the generated images. (dashed line for the sketch tool and color scribble for the color brush)

imposed constraints. This is done by optimizing an objective function that combines the distance to each imposed constraint, and the distance moved from the current point.

The compressed nature of latent space helps not only to understand the data but also to produce explanations. The latent space created by generative models is a compressed representation of data based on Euclidean distance: this implies that the distance between data can be found quickly and easily. The distance between data with many features translates into the latent space in the distance between a few features. In addition, very different features can be compared. The distance between categorical (fixed number of possibilities) and continuous features is difficult to evaluate in the data domain. However, everything is converted in the latent space where distances can be computed using the Euclidean distance. One of the most common explanations retrieved from the latent space are counterfactuals. We can search for counterfactuals in the latent space and retrieve data using a decoder. Integrating the latent space into the counterfactual search can be done in two ways: by adding an additional term in the loss used for the search or by performing the search directly in latent space [113, 58]. The data are usually encoded in a latent space, then it is performed the search for a point that is considered a counterfactual, decode it back into the original feature space using the decoder, and check if it has a different prediction. This process is facilitated because the latent space is a Euclidean space, and every point can be decoded back easily into the original space using the decoder. Most of the works using this procedure focus on images. Data encoding in the latent space can be computed in different ways. In [114], a median absolute deviation is used as a distance measure, while in [115], they minimized the fisher information.

Other methods tried to interpret the latent space features to generate counterfactual data. One of them is counterfactual Recourse Using Disentangled Subspaces (CRUDs) by Downs et al. [116]. CRUDs is a probabilistic model that uses conditional subspace variational autoencoders (CSVAEs) that extracts latent features relevant to a prediction. CSVAE partitions the latent space into two parts: learning representations that predict the labels and learning the remaining latent representations required to generate data. In CRUDs, counterfactuals that target desirable outcomes are generated using CSVAEs in four major steps: (1) disentangling latent features relevant for classification from those that are not, (2) generating counterfactuals by changing only relevant latent features, filtering counterfactuals given constraints, and (4) summarise counterfactuals for interpretability. The result indicates that CRUDS counterfactuals preserve the true dependencies between the covariates better than other approaches. Guidotti et al. used a decision tree to help generate rules in the latent space. Adversarial Black box Explainer generating Latent Exemplars (ABELE) by Guidotti et al. [51] is a local model-agnostic explainer for image data that uses Aversarial AutoEncoderes (AAEs) that aim at generating new counterfactuals that are highly similar to the training data. ABELE generates counterfactuals in four steps: (1) by generating a neighborhood in the latent feature space using the AAEs, (2) by learning a decision tree on the generated latent neighborhood by providing local decision and counterfactuals rules, (3) by selecting and decoding exemplars and counter-examples satisfying these rules, and (4) by extracting the saliency maps out of them. Guidotti et al. found that ABELE outperforms current state-of-the-art algorithms, such as LIME, in terms of coherence, stability, and fidelity.

Several authors take the concept of crossing the decision boundary in the latent space for counterfactual search. Joshi et al. [117] presented a novel approach to characterizing and explaining black-box supervised models via examples. An unsupervised implicit generative model is used to approximate the data manifold and subsequently used to guide the generation of increasingly confounding examples given a starting point. These examples are used to probe the target black-box in several ways. In particular, they demonstrate the utility of manifold guided examples in automatically detecting bias in black-box learning with respect to a potentially protected attribute as well as for model comparison. The proposed method also allows the visualization of training progression and provides insights complementary to notions of calibration of the black-box model. Limitations of the proposed method include reliance on the implicit generator as a proxy of the data manifold. However, the authors specified that their approach is model agnostic both in terms of architectures and training mechanisms for the generative model. They used images as they are easy to visualize even in high dimensions. Samangouei et al. [118] introduced ExplainGAN to interpret black box classifiers by visualizing boundary-crossing transformations. These transformations are designed to be interpretable by humans and provide a high-level conceptual intuition underlying a classifier's decisions. This visualization style can overcome limitations of attribution and example-by-nearest-neighbor methods by making spatially localized changes along with visual examples. While not explicitly trained to act as a saliency map, ExplainGAN's maps are very competitive at demonstrating saliency. They also introduced a new metric, Substitutability, that evaluates how much label-capturing information is retained when performing boundary-crossing image transformations.

Another critical feature of latent space is to group data based on similarity. By putting similar data together, it is possible to identify the data closely resembling different instances. These data are called prototypes, and they are a type of explanation introduced in Section 2.1.5. The most common way to search latent space prototypes is by clustering. Clustering is proven to be more effective on the latent space. Barbakh et al. [119] introduce a set of latent clustering algorithms whose performance function is such that the algorithms overcome one of the weaknesses: its sensitivity to initial conditions which leads it to converge to a local optimum rather than the global optimum. Their online learning algorithms illustrate their convergence to optimal solutions, which standard methods fail to find. Moreover, prototype explanation can be extracted from clustering. After learning the latent space, the prototypes are selected as cluster centers of the clustering algorithm trained on such space. Kleinerman et al. [120] demonstrated that this approach is particularly efficient on health records. They propose a novel deep learning-based patient treatment selection approach using latent-space prototyping. Machine-assisted treatment selection commonly follows one of two paradigms: a fully personalized paradigm which ignores any possible clustering of patients, or a sub-grouping paradigm which ignores personal differences within the identified groups. While both paradigms have shown promising results, each suffers from important limitations. The use of latent space prototyping allows for balancing between the two paradigms and allows better results. In an extensive evaluation, using both synthetic and Major Depressive Disorder (MDD) real-world clinical data describing 4754 patients from clinical trials for depression treatment, they show that their approach favorably compares with state-of-the-art approaches.

A pervasive use of the latent space is for anomaly detection. When placing data in latent space, data clusters are clearer, and exceptions can be better identified. Liu et al. showed that it is possible to use the latent space of a generative model to build prototypes and then use them to detect deviations from them to catch anomalies [121]. Anomaly detection in automated quality inspection is essential for guaranteeing industrial products' surface quality. They proposed a semi-supervised anomaly detection method Dual Prototype Auto-Encoder (DPAE), based on the latent space paradigm. At the training stage, DPAE is optimized to minimize the reconstruction loss between the input of the encoder and the output of the decoder. At the same time, DPAE also minimizes the distance between the latent vectors and the standard prototypes, which are iteratively updated to represent the center of the latent vectors of defect-free samples. Given a test sample, the model can automatically generate two latent vectors: one for the sample and one for the prototype. As a result, the distance between the two vectors of the standard sample tends to be closer, resulting in small errors for standard samples and large errors for anomalies, which can be adopted as a criterion to detect the anomalies. Some approaches have shown that it is possible to use energy-based optimization methods [122] to detect anomalies and generate data. Due to the latent space's low dimensionality and the top-down network's expressiveness, a simple energy-based method in latent space can capture regularities in the data effectively. A similar concept is used in Abati et al. [123], where this time, the prototypes found in the latent space are used to detect novelties instead of anomalies. They designed a general framework where a deep autoencoder with a parametric density estimator learns the probability distribution underlying its latent representations through an autoregressive procedure. They show that a maximum likelihood objective, optimized in conjunction with the reconstruction of standard samples, effectively acts as a regularizer for the task at hand by minimizing the differential entropy of the distribution spanned by latent vectors. Unlike prior works, this proposal does not make any assumption about the nature of the novelties, making our work readily applicable to diverse contexts.

Other uses of the latent space prototypes are for face identification [124], guided generation of sentences [125], and for other explanations generations like SMs [51].

Chapter 3

Interactive exploration of the latent space

In this first latent space analysis, we used an interactive visualization to unveil the unknown meaning of the latent space dimensions. In particular, we interactively explored the latent space of a variation autoencoder guided by SHAP scores. We developed a model agnostic approach for explaining the outcome of a black box model based on two steps:

1. A variational autoencoder maps multi-dimensional input features to a bi-dimensional space.

2. A widely used explanation technique, i.e., SHAP [25], is exploited to measure the relevance of each input attribute to the position in the latent space.

The combination of the two techniques enables the effective visualization and exploration of the contribution of each original feature to the latent space. Modifications on a single input feature generate a spatial offset on the latent space visualization, allowing cognitive-friendly navigation of the joint modification of multiple attributes.

We chose a specific family of variational autoencoder architectures, called Conditional Variational Autoencoder (CVAE), which exploits the outcome of a classification model to generate a latent space where data are grouped by the similarity of their attributes and the prediction label assigned by the black box. We exploit this mapping to encode the classification provided by the black box to be explained in the latent space positions. The underlying idea of autoencoders is to create a more informative latent representation of data that holds more valuable properties than real input data. By enhancing the latent space embeddings with prediction labels, we can observe how the modification of the features affects the position in the latent space and deduce which features need to be modified in order to alter the black box predictions. Creating such adequate latent space is a challenging task that is fulfilled by autoencoders forcing themselves to use a latent dimension lower than the input to model the same amount of information. We can efficiently map multi-dimensional points into a display using a low-dimensional space (i.e., two dimensions). The positions of instances in the latent space can be used to reason about the internal mechanism adopted during the classification. The low dimensionality forces the autoencoder to maximize the contribution of relevant features, discarding those that are marginal for the classification task. For this reason, we learn the mapping using a training set labeled by the black box.

However, since Neural Networks are considered black box models [5], also autoencoders can be automatically considered black boxes. The reasons why an instance is mapped to a specific position remain unknown since the autoencoder model does not reveal linear and non-linear relationships with the data properties. Therefore, the latent space cannot be considered an interpretable space, and the relationship between the latent feature and the real one is unknown. To overcome this, we exploit and re-adapt existing eXplainable Artificial Intelligence (XAI) techniques to unveil the meaning of the features in the latent space and make them understandable. Our idea is that, since the latent space algorithm is a Neural Network, we can apply XAI techniques to explain the latent space in terms of the input features. Our contribution focuses on building an explanation for the latent space mapping. The explanation method is based on the SHAP (SHapley Additive exPlanations) [25] technique, which returns the contribution of each input feature to the final classification. We apply the SHAP method to the mapping model, yielding a sum of linear contribution for both the x and y components. If we consider a single input feature, its contribution is approximated by SHAP as a vector in latent space: an increase or decrease in the value of the input feature produces an increase or decrease in its length without changing direction. Since we are considering multiple input attributes, the final position in the latent space is the vector sum of the single feature contributions. The order of the importance of the features is a result of the SHAP method, going from the most relevant to the less relevant (more details in Section 3.2).

A recent study [126] described interactivity as an added value when speaking about explainability. We used a popular interactive framework called DASH to visualize the explanations with interaction. We developed an interactive framework that allows users to interact with the input features and understand how modifying them influences the latent space position. The single contributions are presented as linear contributions that we can map directly on the latent space, giving the user a visual overview of the outcome. We leverage interactivity to let the user vary the values of the input space (i.e., the original attributes of the instance) to explore the resulting mapping in the latent space display. In this way, the user can explore how variation in the input may influence the mapping of the autoencoder and, hence, the relevance of features. The variation of a feature f influences the final mapping of the point, since it modifies the magnitude of the vector corresponding to f. Our framework makes it possible to observe how changes to real features affect/or do not affect changes in the latent features and guide the data into different prediction regions.

The rest of the chapter is organized as follows: Section 3.1 recalls the notions needed to understand the proposed methodology, which is illustrated in Section 3.2.

Section 3.3 presents the interactive framework. Finally, Section 3.4 concludes the paper by discussing known limitations and proposing future research directions.

3.1 Meaning of SHAP explanations

This section will explain what SHAP scores are to understand our implementation fully. Our contribution leverages the widely used XAI method named SHAP (SHapley Additive exPlanations) [25]. The goal of SHAP is to explain the prediction for any instance x as a sum of contributions from its individual feature values. SHAP employs game theory to obtain single-feature contributions. It uses *Shapley values* to define how to distribute the output of a model among all the input features. SHAP is a game-theoretic approach, based on *Shapley values*, to explain the output of any machine learning model. The explanation is provided as a sum of relevance from the input feature values. Note that the "feature value" x_i here refers to the numerical or categorical value of a feature i for the instance x. Each value is considered as a contribution to a cooperative game whose payout is the final prediction. Accordingly to the input instance, Shapley values estimate the payout among the input features.

We can clarify the intuition behind SHAP with the following example. A group F of people is playing a basketball game where the goal is to make as many baskets as possible, and we want to compute the contribution of player j. SHAP removes player j from group F and repeats the game. The difference between the scores obtained with and without player j represents how much player j has contributed to the final scores in this scenario. SHAP computes this difference for every possible combination of other players, removing a different player every time. First, it removes player j and the player j', then j and j'', then j, j', and j'', etc.. The average of these scores is the Shapley value of player j representing the contribution of j to the initial score.

If we consider that player j is an input feature, F is the set of all features, and the basket game is a machine learning model f, then we can move to a machine learning setting. Here, a Shapley value represents for each feature j the change in the model prediction when conditioning on that feature. Formally, consider S as a possible subsets of F without the feature j, i.e., $S \subseteq F \setminus \{j\}$. The prediction of the model is obtained with that feature present $f_{S \cup \{i\}}$ and with the feature withheld f_S. Then the two predictions are subtracted $f_{S \cup \{i\}} - f_S$. Since the effect of withholding a feature depends on what other features are present during the prediction, the above differences are computed for all possible subsets S. The Shapley values ϕ_j are then computed as a weighted average of all possible differences.

$$\phi_j = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{j\}} - f_S \right] \tag{3.1}$$

The computation of the Shapley values is exponential with the number of features. Kernel SHAP implements an approximated estimation of the Shapley values by using a weighted linear regression [25]. The connection between linear regression and Shapley values is that Eq. 3.1 is a difference of means. Since the mean is also the best least-squares point estimate for a dataset, it is natural to search for a weighting kernel that causes linear least squares regression to approximate the Shapley values. This variation is implemented into the so-called Kernel SHAP method. The regression coefficients estimate the Shapley values with a weighted linear regression model and an appropriate weighting kernel. The only requirement of Kernel SHAP is a black box (the autoencoder in our case) and a set of points to train the weighted linear regression. The more points, the more accurate the approximation of the Shapely values.

The other challenge to compute Shapley values is the correct estimation of suppressing one of the features. Since many machine learning models are based on fixed-length input, suppressing a single feature is impossible. To overcome this problem, Kernel SHAP substitutes the value of a feature with the mean value observed over the whole dataset. If all the features are replaced with their corresponding means, it is possible to identify a mean input instance with the corresponding mean prediction value. This value is also called the *expected value* and is the model's prediction when all the feature values are set with the mean value.

3.2 Methodology

This section describes how we performed the exploration in the latent space. We will describe the cluster analysis, how we applied SHAP, and the interactive framework. Our objective is to analyze a given data by encoding it in the latent space using an autoencoder. Firstly, we can perform a cluster analysis of the latent space. Then, we use SHAP to unveil why a point has been encoded in a particular part of the space. Guided by the SHAP score, we can move to different portions of the latent space with different predictions. The scores returned by SHAP allow us to understand which real features have influenced the position in the latent space most. Therefore, we can use these scores to explain the role played by the real features w.r.t the machine learning model. We organize this section in two parts: the first focuses on describing the latent space structure, while the second focuses on how the explanation is created.

3.2.1 Neighborhood Analysis

Let $\mathcal{D}\langle X, Y\rangle$ be a dataset where $X = \{x_1, \dots, x_n\}$ represents the features space, while $Y = \{y_1, \dots, y_n\}$, represents the label space. Let b a black box model trained on $\mathcal{D}$ which return the prediction labels $\hat{Y} = \{\hat{y}_1, \dots, \hat{y}_n\}$. Without loss in generality and for simplifying the presentation, we may consider a binary classification task and use the probability returned by the classifier as labels. However, we remark that this process can be expanded to multi-class problems. Given a record $x \in X$, represented

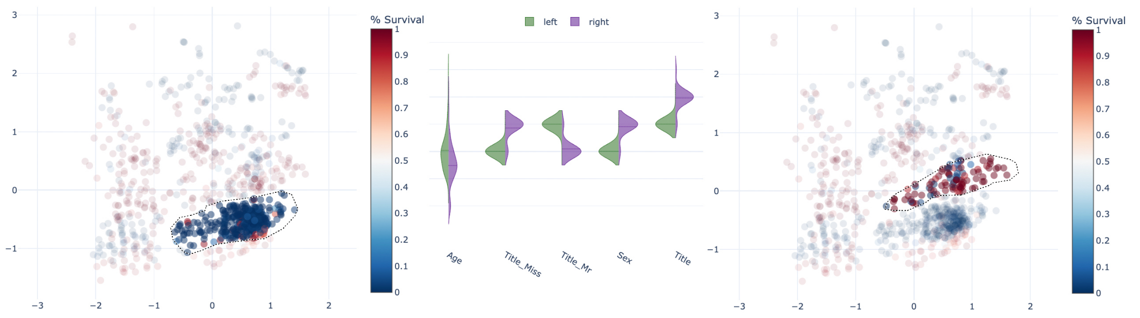


Figure 3.1: Here, we have the distribution analysis of the two clusters analyzed in the Titanic dataset. The violet distribution is the distribution of the points belonging to the cluster highlighted in the right figure, while the dark green is the distribution of the left one. The features are sorted in descending order from the most separated distributions on the right to the least separated one on the left. The feature "Sex" and "Title" are the ones that characterize the clusters: all the points in the blue cluster are men with a low title.

as a vector $x_i \in \mathbb{R}^m$, the corresponding classification of the black box $\hat{y} = b(x)$ and an autoencoder A, we indicate with the notation $z = A(x)$ the encoding of the instance x with A. Thus, we indicate with z the latent representation of x according to A. As a model, we adopted a particular type of autoencoder called Conditional Variational Autoencoder (CVAE). A Conditional Variational Autoencoder is a Variational Auto Encoder with extra input to both the encoder and the decoder. In addition to traditional data, a CVAE concatenates to the encoder and the decoder the output probability value $\hat{y}$ of the black-box b. This additional information allows the latent space to represent the data not only by their features but also by their predictions. Two data are mapped to the same point in latent space if they are similar to the features and black-box prediction. We indicate with the notation $z = A([x, \hat{y}])$ the CVAE encoding of the instance x.

Typically, the representation of $\mathcal{D}$ in the latent space has a clustering-based structure [100], i.e., similar points are grouped in close areas in the latent space. In addition, CVAEs exploit the information given by $\hat{y}$ to place instances into the space accordingly with their labels. Then, with our proposal, we can observe the neighbors in the latent space, identifying common feature characteristics among the instance

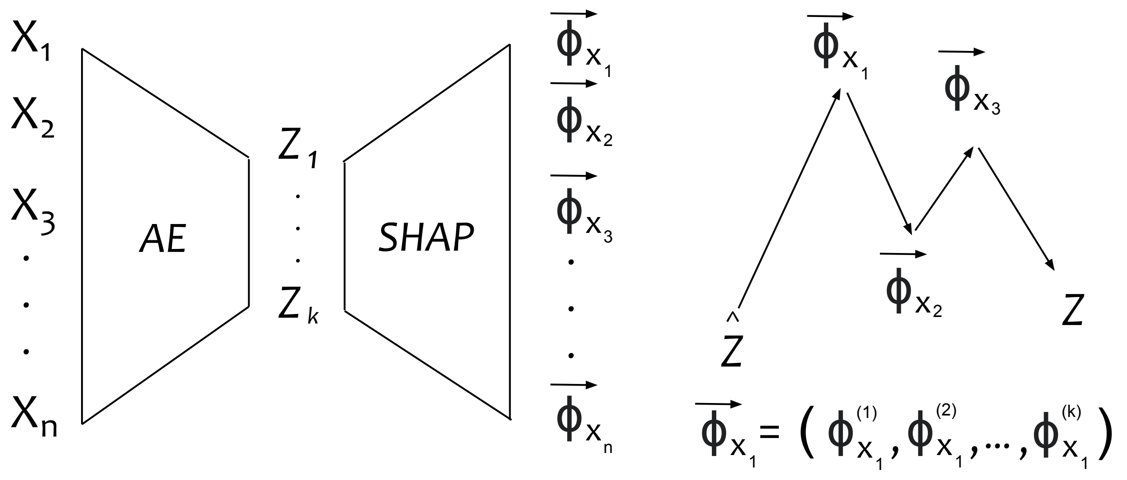


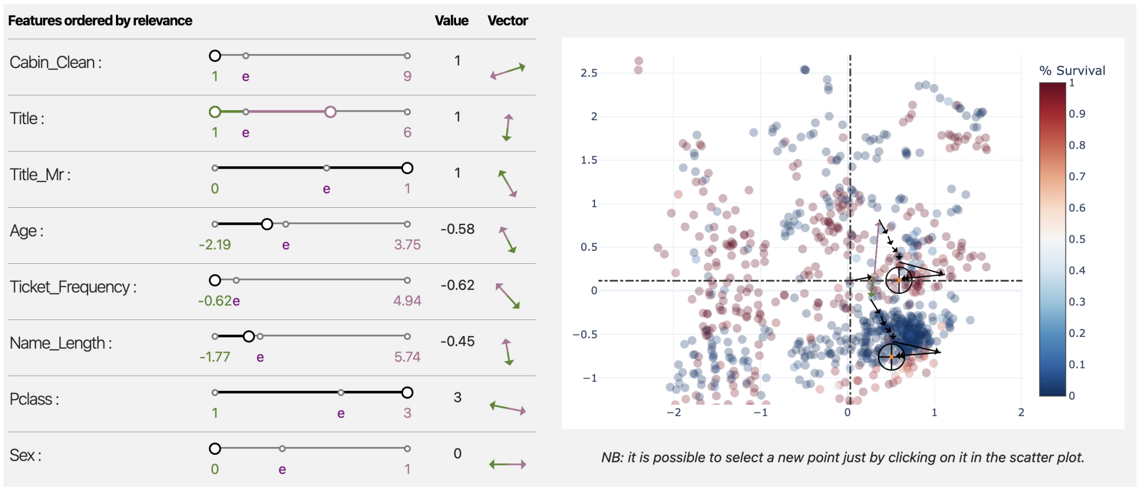
Figure 3.2: Illustration of SHAP vectors. For every feature in the input space x, SHAP returns a vector of dimension equal to the latent space. The sum of these vectors from the expected value gives the final position in the latent space.

in the same group and catching the correlation with the target variable $\hat{y}$. With our framework, it is possible to define and select different neighborhoods in the latent space and compare the distributions of the real features of the points in the two groups. In Figure 3.1 there is an The features with the most different distributions among the two selected sets are the ones that characterize the belonging to the groups. In particular, if we select two clusters with points with different labels, it is possible to identify the discrimination features. For example, we can select a cluster with many points with target label L and another one with a lot of $\neg L$ and notice that the points in the L cluster are characterized by the value of the feature "Male" set to *true*.

3.2.2 Establishing the Interactive Connection via SHAP

Given an instance $x \in X$ and a CVAE autoencoder A trained on $[x, \hat{y}]$, our goal is to find the explanation e unveiling why x has been encoded in the latent representation $z = A(x)$. We highlight that we assume that the autoencoder A is well-trained as it is meaningless to explain the behavior of a model not adequately learned. The proposed explanation is built from the Kernel SHAP method to estimate the Shapley values of z. We rely on the *kernel version* of SHAP since it is the one that works with NNs [25]. Kernel SHAP adopts a specially-weighted local linear regression to estimate the SHAP values for any model. Our exploration approach is based on the approximation of the SHAP's contributions as coefficients of a linear equation. The sum of these coefficients gives the final position in the latent space. This approximation has two advantages: on one hand, it allows efficient computation of the Shapley values; on the other hand, after an initial computation for all the boundary values of the attributes, the position on the latent space can be approximated with linear interpolation, making the visualization and interaction more responsive.

We can interpret these values in terms of comparison with the *expected prediction*. As described before, the expected prediction $\hat{z}$ is the latent representation, corresponding to substituting every feature value with its mean in the dataset $\mathcal{D}$. Thus we can project this feature in the latent space at its representation $\hat{z}$ and use that location as a reference position. Given the Shapley value of the instance x, we can add the contribution of each value starting from the reference position $\hat{z}$. Thus, starting from the expected prediction $\hat{z}$, we can sum to it the contribution of every input feature in the dimension k and obtain the position in the latent space, i.e., the latent representation $z = \hat{z} + \sum_m \vec{\phi}(x_m)$. As illustrated in Figure 3.2, the output returned by SHAP can be considered as a vector $\vec{\phi}_{x_i}$ which components are the contributions of a feature in a specific dimension of the latent space. Therefore, if we sum these vectors starting from the expected value position $\hat{z}$, the output will be the original position z in the latent space. For example, a typical output would be: the feature "Title" has contributed a value of 1 to the first dimension of the latent space and 1.5 to the second one, so it is relevant for both latent space dimensions. Instead, the feature "age" has contributed 1.5 to the first dimension and 0.1 to the second one, so it is relevant only for the first dimension of the latent space and does not affect the other one.



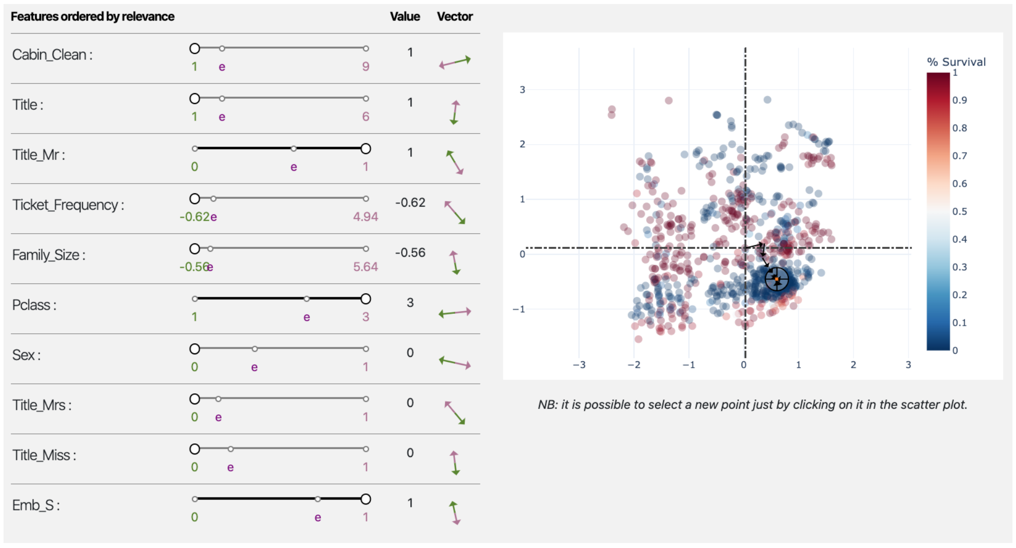
NB: it is possible to select a new point just by clicking on it in the scatter plot.

Figure 3.3: Interactive framework for the titanic dataset. We have several sliders of the top ten most informative features on the left, ordered by relevance. The violet letter e on the bottom of every slider is the expected value for that particular feature. The two columns on the right of the sliders are in order: the feature's value and the shap score's vector representation. On the right, we have a graph of the latent space learned. The points are labeled red and blue according to the prediction label. The point taken into analysis is selected using a black viewfinder. The expected position is the origin of the two gray axes, and the black vectors represent the contributions of each feature, the sum of which lead from the expected value to the actual point. By modifying the Title feature from 1 to 4, we can see how the position of the point is changed in the latent space following the shap vector. By moving above the expected value of the feature, the effect in the latent space is given by the shap vector highlighted in pink.

An explanation for a classified instance is built around exploring the Shapley values contributions. First, the contributions of each feature are sorted by order of magnitude. The user can then change the value of one of the input features to observe how the position of z in the latent space changes. Those features with higher relevance will produce a broader impact on the latent space since their corresponding magnitude is larger. We leverage an exploration strategy based on a low-dimensional latent space to allow an efficient and comprehensible visualization of the induced space, i.e., two or three dimensions. When the user explores a new instance, the corresponding latent position is determined, and the vectors representing the contributions of the feature are computed and mapped on the visual space as vector arrows (Figure 3.3). The variation of one of the attributes affects changing the magnitude of the vector arrows displayed on the visualization. We clarify this aspect by highlighting that, since SHAP values can be approximated with linear regression, they also have some linear properties concerning the expected value. For example, consider a feature like *age* ranging from 0 to 100 with expected value 40, and consider that the value 80 has a SHAP score of 0.6. If we substitute the value of the feature to 40, then the SHAP score would be 0 since the contribution of *age* equal to the expected value is null. However, if we substitute the value with 20, the SHAP score will be negative due to the linear propriety. Since the position in the latent space is given by the SHAP values, we can use them as a guide to increase or decrease feature values in the real space to move points to different positions in the latent space. SHAP vector directions are reported on the right of the sliders in Figure 3.3, allowing the user to see how the change of an input feature will result in a modification of the position in the latent space. This allows the user to explore the latent space guided by the direction of the SHAP scores and analyze the prediction in different space portions.

3.3 The Interactive Framework

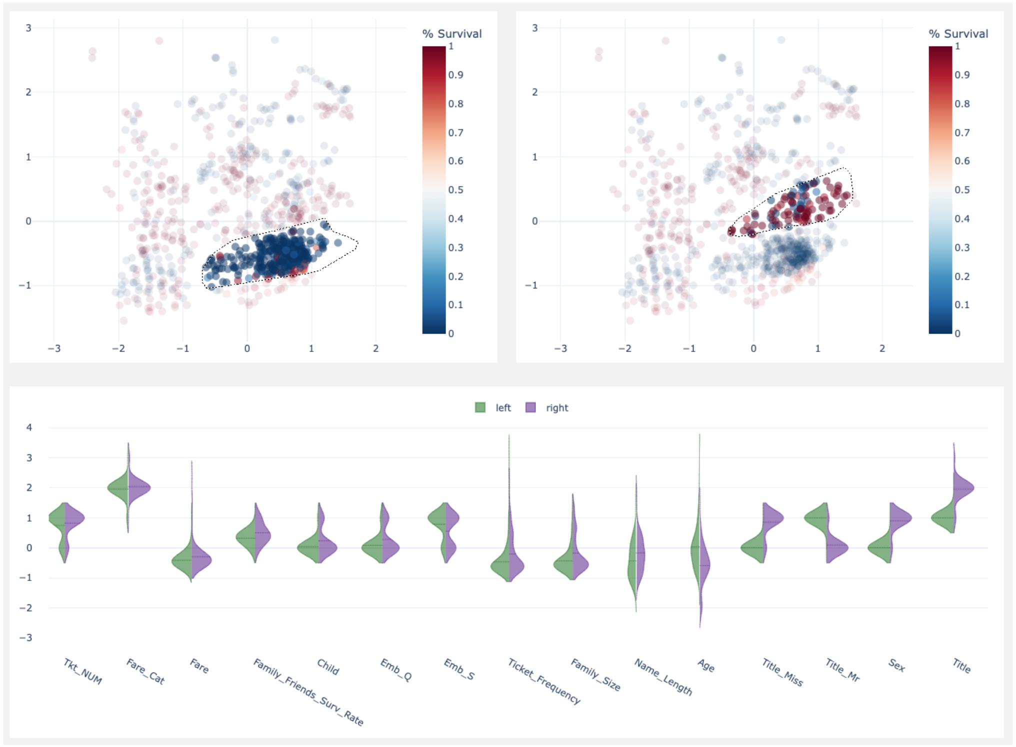
Finally, this section demonstrates how the interactive framework can be used as an explanation tool. Moving into the latent space guided by SHAP scores can provide insights to identify homogeneous groups of points, outliers, or close points with different predictions despite their similar latent characteristics. In particular, this last case can help identify misclassifications or malicious use of the black box, for example, through adversarial instances or isolated data. Since it is interesting to analyze different latent positions by playing with the input features, we realized an interactive framework, acting as a front-end for our latent space explanation proposal, that allows the user to analyze the distribution of latent points and interact with their SHAP values (Figure 3.4).



Features ordered by relevance	Value
Cabin_Clean :	1
Title :	1
Title_Mr :	1
Ticket_Frequency :	-0.62
Family_Size :	-0.56
Pclass :	3
Sex :	0
Title_Mrs :	0
Title_Miss :	0
Emb_S :	1

NB: it is possible to select a new point just by clicking on it in the scatter plot.

(a) Latent Space Exploration: on the left, we have ten sliders of the top 10 most important features given by SHAP in order of importance from top to bottom. On the right, the latent space created by the autoencoder is represented with the SHAP vector explanation for a given point.



(b) Clustering Analysis: the user can select two different clusters and analyze the distributions in the histogram plot at the bottom.

Figure 3.4: How the interface is presented to the user. On the top, there is the latent space exploration guided by SHAP scores, while on the bottom, there is the clustering analysis

The central component of our solutions is the visualization of the latent space. The two dimensional visualization of the latent space is built as a scatter plot, where latent instances are represented as points. The projection of the expected value is highlighted as the intersection of two dotted lines. Contributions of single values are visualized as black arrows oriented accordingly to the coefficients returned by the SHAP explanation method. The destination location of z is highlighted with a viewfinder shape. The latent space visualization is linked with a set of sliders where the user can specify values for each attribute. The user selects a point in the latent space, and the sliders update with the new feature values. The higher the SHAP score, the more important the feature, so we sorted the sliders in descending order of importance. Each slider highlights the minimum and maximum values for that feature along with its expected value position represented as a small violet e. In addition, we also provide directional vectors for each feature to help the user to predict the direction and verse of the translation in the latent space. Every feature is encoded as a vector in the latent space, and the user can move the sliders to change the magnitude of the corresponding vector arrow. In Figure 3.5, there is a small example of the effect of the modification. If we substitute the value of the feature with its expected value, then the contribution in the latent space is zero. Suppose we modify to values lower than the expected value. In that case, the effect will be the one highlighted by the green vector, whose magnitude increases linearly the more distance we move from the expected value. Conversely, if we set a value higher than the expected value, the effect will be highlighted by the pink vector. Due to the propriety of the SHAP values, these vectors are always one opposite of the other.

When the user changes one of the values on a slider, a new instance is generated, the black box classifies it, and the SHAP values are computed. The vectors are then updated on the latent space visualization. Since the number of features of the dataset may be large, we decided to visualize only the sliders of the top 10

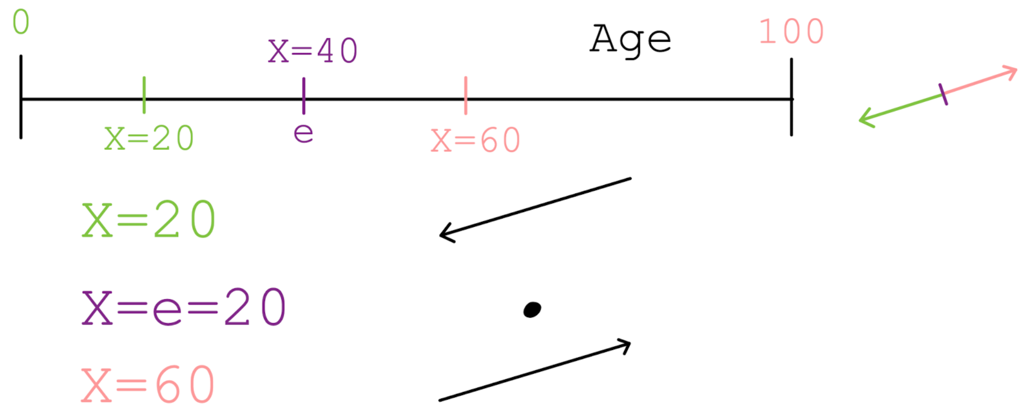


Figure 3.5: effect of the feature modification in the latent space. Here, the feature Age is represented with an expected value of 40. If the feature has this value, then the contribution in the latent space is null. If we set a lower value, the green vector guides the effect, and a higher value follows the pink one.

most representative features selected in decreasing order of magnitude (from top to bottom) of the SHAP scores. The two displays are mutually linked: when a slider is changed, the latent space visualization is updated; when a point in the scatter plot is clicked, the corresponding feature values are loaded in the set of sliders. Due to visualization purposes, we decided to fix the latent space dimension with $k = 2$, but we remind that our approach remains valid for higher dimensions.

The interface is divided into two parts. On the top, the sliders and the latent space visualization allow the exploration of latent space by changing the value of the input feature. On the bottom, the user can perform a neighborhood analysis by selecting groups of points from two instances of the latent space visualization (Figure 3.1). The two groups of points are then analyzed to determine which features distinguish one group from the other. The differences are visualized as a set of violin plots, comparing the distribution of observed values in the two groups. The violin plots are sorted accordingly with features that maximize the distance of the mean of distributions.

We tested our framework with a publicly available tabular dataset, i.e., the

Titanic dataset, which contains the data of real Titanic passengers.[1] Each passenger is labeled as a *survivor* (red) or *not-survivor* (blue). Every categorical feature is transformed into a one-hot dimension vector equal to the number of possible categorical values. The continuous variables are normalized in the range [-1,1]. The dataset contains 1310 passengers described by attributes such as age, passenger class, sex, fee, etc. As a classification model, we trained an xgboost model [127]. A web application that implements our framework is accessible at the following link: https://kdd.isti.cnr.it/LSE/. The source code is also available on GitHub: the link is available from the web application URL.

The latent space created by the autoencoder is illustrated on the right of Figure 3.3. Most of the data labeled as *non-survivor* are concentrated in the lower right part of the latent space. The survived people, instead, are more sparse. The most critical features reported by our method are *Title* and *Cabin*. However, if we look at the contributions of the SHAP score, only the feature Title can move from the bottom blue cluster to the top red one while Cabin is more responsible for the x-axis movements. These two features are the most informative ones of our dataset; in addition, the number of features with a score bigger than zero is 19, 30% of the 60 original features. This sparsity means that many of our features do not represent our data at all.

We selected two clusters in the interface to analyze the difference in distributions. We selected the one on the right with many not survived people and the one closer at the top. We can see, as expected, that the feature "Title" is the most representative of the two clusters (Figure 3.1). In the blue cluster, the people have lower titles than those in the red. However, the people in the blue cluster are all men. We conclude that in the dataset, there is a correlation between the features "Sex" and "Title". Male people with a lower title are the ones with meager survival chances. In Figure 3.3, we took a point in the blue cluster on the bottom and tried to modify

[1] https://www.kaggle.com/c/titanic/data

its most relevant features to move it to the top cluster. In particular, we can see that the feature "Title" contributes to the cluster's direction and has a lower value than the expected one. By modifying this feature to a value greater than the expected value (i.e., on the right of the e placeholder), we discovered that it is possible to move the point from the original cluster to the one in the center, which is mainly labeled as *survivor*. The more we increase the value, the more the point will be in the top part of the space. Interestingly the feature "Sex" has a low SHAP value meaning that our model is only looking at the feature "Title" for distinguishing points.

3.4 Discussion and Future Directions

In this chapter, we explored the possibility of using an explanation methodology that exploits the combination of latent spaces learned by a conditional autoencoder with a feature relevance method, namely SHAP. Interactive learning is a novel framework of explanations where, in each step, the user interacts directly by modifying and correcting the explanation. This interactive feedback has been proven to be essential for better comprehension of the black-box decision [128], and here it is no exception. We have extracted beneficial information about the data under analysis by utilizing an interactive exploration of the latent space to evaluate the position of single instances or groups of points. This interactive framework is perfect for exploring the latent space and extracting information about the black-box predictions.

However, it is not free of problems. SHAP scores are expensive to compute and they are still an approximation of the actual Shapley values. A faster method similar to SHAP in leveraging a similar procedure for identifying the importance of the features is DALEX [129]. DALEX employs a different strategy to approximate Shapley values, using a greedy search to find the best feature to remove. In our implementation, we used SHAP because it is one of the most well-known

XAI procedures. However, we plan to experiment our proposal with DALEX as future work. Another difficulty encountered was the conditional autoencoder training. While autoencoders are the most popular and established generative models, they lack training consistency. Different runs could produce very different results. In general, evaluating the goodness of the created latent space is difficult, particularly if the generated latent space is suitable for explanation exploration. Autoencoder training problems are well known in the actual state-of-the-art and, for now, without any solutions. The conditional variation is better for representing explanations, but most of the stability problems remain. About the interaction framework, it lacks smooth animations. A further development could be to introduce real-time visual-based animation to bind the changes on the sliders with the updates on the visualization. This would help the user better understand the movements in the latent space even more.

Although this interactive approach may seem attractive at first glance, we are trying to explain a black-box using an auto-encoder, which is itself a black-box. How can we be sure of its behavior? We will never know whether the problem encountered comes from the black-box or the autoencoder: we will never know who is right and wrong. To solve this problem, in the next chapter, we will examine the possibility of transparently creating the latent space so that we know precisely how the latent space is created and better analyze the black-box.

Chapter 4

Interpretable by Design Latent Space

In this chapter, we tried to answer the question: is it possible to create the latent space transparently and retain its similarity and classification properties? Can this new interpretable latent space be used to train ML models and produce explanations?

We found in the last chapter that the latent space has properties that enable researchers to understand the data better and produce better models. However, it is usually created using black-box models such as autoencoders. This makes it difficult to use as an explanation technique since one must apply a post-hoc technique on it as well as on the black-box, doubling the effort. 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 [3] (Section 2.1.7). In particular, we propose a new approach to perform classification and explanation, 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. Then, the latent space is used to obtain classification and retrieve

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, $x'$ - counterfactual explanation
Train($X, Y, f, K$):
1  $\mathcal{M} \leftarrow LearnBestLatentSpace(X, f, K)$;                //find best latent space
2  $Z \leftarrow \mathcal{M}(X)$;                          //turn training data into latent space
3  $b \leftarrow f(Z, Y)$;                               //train classifier on the latent space
4  return $b, \mathcal{M}$;
Predict and Explain($x, b, Z, \mathcal{M}$):
5  $z \leftarrow \mathcal{M}(x)$;                                    //get latent representation
6  $\hat{y} \leftarrow b(z)$;                                                 //apply prediction
7  $x' \leftarrow GetCounterfactual(z, b, Z, \mathcal{M})$               //get counterfactual explanation
8  return $\hat{y}, x'$;
```

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, it returns a *counterfactual example*, i.e., another instance x' with minimal changes to the features of x that is classified differently (see Section 2.1.6).

Thus, the main contribution of this chapter 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 [87, 130]. Second, we assess qualitatively and quantitatively the counterfactuals provided by our method compared to others.

The rest of the chapter is organized as follows. Section 4.1 illustrates the proposed methodology. Then, Section 4.2 reports the experimental results comparing ILS against state-of-the-art methods. Discussion and future research directions are discussed in Section 4.3.

4.1 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 in the latent space can be explained exactly in terms of input characteristics. By using an interpretable mapping instead of an opaque Ml model, we can know exactly how the data were mapped in the latent space and be able to trace their position in the input space without the help of a decoder. We claim that the latent space created by ILS is interpretable since the linear mapping of the input features can be represented in the latent space in the form of vectors (Figure 4.1 (left)). Transparency can be exploited to obtain the counterfactual explanation of a classifier trained in such a space. Using a transparent mapping, it is possible to translate the explanation made in the latent space into the input space in an effortless way. Like most classification methods, ILS has a *train phase* and a *predict phase*. The latter is indeed a *predict & explain phase*.

The whole procedure, illustrated in Algorithm 1, starts by learning the latent space model $\mathcal{M}$ from the training set X. With $\mathcal{M}$, we indicate a model able to map the input dataset $X \in \mathbb{R}^n$ into a latent version $Z \in \mathbb{R}^k$. A set of latent dimensions K is tried and the best k dimension is selected among this set (line 1). Details about the latent space learning are discussed and formalized in Section 4.1.1. Then, $\mathcal{M}$ 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 & 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 5-6). Then, the prediction $\hat{y}$ is explained by exploiting the interpretability of the learned space, returning a counterfactual instance x' (Line 7). Details on how the counterfactual is constructed are given in

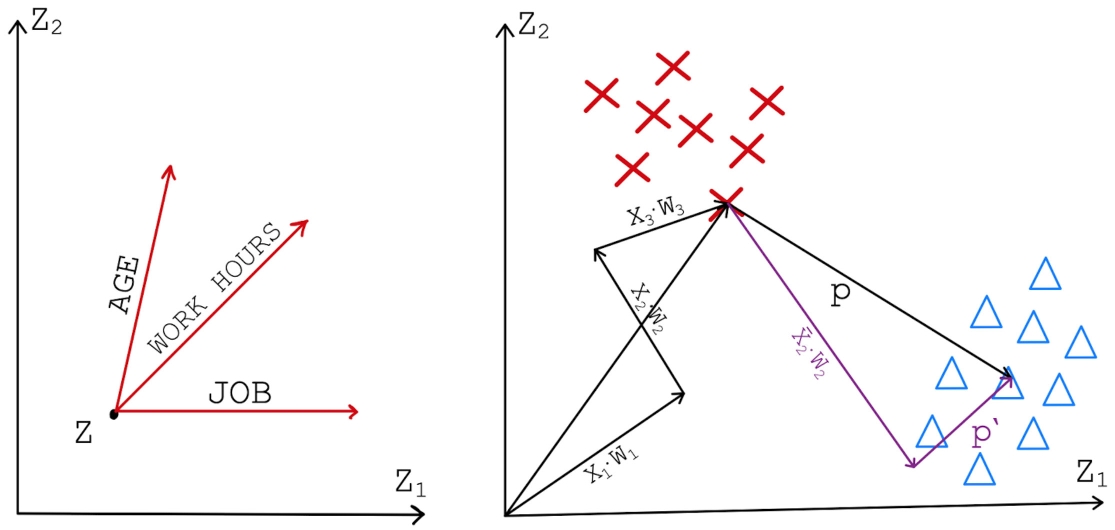


Figure 4.1: Scheme of the vector model used for creating explanations. Left: representation of the input features in the latent space. Right: we illustrate a step in the ILS algorithm to modify the input features based on the position of the latent space, as explained in Section 4.1.2. The best update found by ILS is $(\bar{X}_2)$, p is the projector to the different class center, p' is the new projector for the next step

Section 4.1.2.

4.1.1 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 learn a latent space by combining a similarity loss analogous to the one utilized in t-SNE [131] with the mapping reasoning of PCA: the data are mapped into the space based on the similarity between them. 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 [25] 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 $\mathcal{M}$ that transforms the input space X of dimension n into a latent space of dimension k, i.e., $Z = \mathcal{M}(X)$ such that

Algorithm 2: *LearnLatentSpace*(X, k)

```
Input  : $X$ - training data, $k$ - latent space dimension
Output: $\mathcal{M}$ - trained transformation model
$i \leftarrow 0$;                                        //init. iteration index
$L_i \leftarrow \infty$;                                 //initialize loss
$\mathcal{M} \leftarrow init()$;                         //initialize model weights
$S_X \leftarrow PairwiseSimilarity(X)$;                  //original similarity matrix
while $L_{i-1} > L_i$ do                                 //until the loss decreases
    $Z \leftarrow \mathcal{M}(X)$                        //get latent representation
    $S_Z \leftarrow PairwiseSimilarity(Z)$;              //latent similarity matrix
    $L_i \leftarrow KLD(S_X, S_Z)$;                      //compute Kullback{Leibler Divergence loss
    $\mathcal{M} \leftarrow update(\mathcal{M}, L_i)$;   //Update the model using backpropagation
    $i \leftarrow i + 1$
return $\mathcal{M}$;
```

$z_j = w_0x_0 + w_1x_1 + \cdots + w_ix_i + \cdots + w_nx_n$, where w are the weights of the model $\mathcal{M}$, x is an instance belonging to the input space $\mathbb{R}^n$, z its transformation to the latent space $\mathbb{R}^k$, and k is the number of latent dimensions. In the literature, the k parameter is challenging to select and is usually provided heuristically. Hence we have two objectives: finding the "best" weights w for the linear model $\mathcal{M}$ and the best latent space dimension k. The former is achieved by Algorithm 2 (*LearnLatentSpace*), while the latter from Algorithm 3 (*LearnBestLatentSpace*).

LearnLatentSpace (Algorithm 2) starts by initializing the model $\mathcal{M}$ with random weights (line 3). Thus, we use gradient optimization [132] to minimize an unsupervised loss that encourages similarity between near points. We adopt the similarity probability loss introduced in [131] 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.

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

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

Algorithm 3: $LearnBestLatentSpace(X, K, f)$

```
Input  : $X$ - training data, $K$ - list of latent space dimensions, $f$ - classifier training function
Output: $\mathcal{M}$ - Trained model
1  $\mathcal{M} \leftarrow \emptyset$;                                  //empty best transformation model
2  $s \leftarrow 0$;                                                      //init.  best model score
3  for $k \in K$ do
4      $\mathcal{M}' \leftarrow LearnLatentSpace(X, k)$;              //learn latent space with $k$ dimensions
5      $Z \leftarrow \mathcal{M}'(X)$;                                     //get latent representation
6      $b \leftarrow f(Z, Y)$                                              //train classifier on Z
7      $s' \leftarrow evaluate(b(Z), Y)$;                              //evaluate classification performance
8      if $s < s'$ then
9          $\mathcal{M} \leftarrow \mathcal{M}'$;              //take best transformer w.r.t classifier performance
10 return $\mathcal{M}$;
```

where x_i and x_j are the two points, and σ is the variance of the Gaussian. Algorithm 2 computes the similarity probability of any two points x_i and x_j in the input space (Line 4) and in the latent space (Line 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 [133] (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 $\mathcal{M}$ until convergence (line 9)[1].

The other algorithm *LearnBestLatentSpace* described by Algorithm 3 is designed

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

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 $\mathcal{M}'$, mapping the input space to a k-dimensional latent space (line 4). Next, X 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 $\mathcal{M}'$ (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.

4.1.2 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 $x = \{x_1, x_2, \dots, x_n\}$ 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 $p = c - z$. The goal is to find the best feature x_i such that its new value $\bar{x}'_i$

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: $x'$ - counterfactual explanation
1  $z \leftarrow M(x)$;                                        // get latent representation
2  $C \leftarrow Clustering(z'|z' \in Z_{\neq})$;              // centroids with different prediction
3  $i \leftarrow \arg\min_{i} d(C_i, z)$                       // find the centroid
4  $c \leftarrow C_i$
5  $\bar{x} \leftarrow x$;                                     // init.  counterfactual
6  while $b(M(x)) = b(M(\bar{x}))$ 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         $x'_i \leftarrow Equation2(\bar{x}, i, p, M)$       //calculate update for feature i
11         $u_i \leftarrow x'_i$;                              //store update
12     $i \leftarrow \arg\min_{i \in [1,n]} \{d_{euclidean}(u_i, c)\}$   // find best update
13     $\bar{x}_i \leftarrow x'_i$                             // apply the best update
14 return $\bar{x}$;
```

moves the candidate counterfactual $\bar{x}$ towards the desired prediction. In particular, each input feature x_i is responsible for a direction of movement in the latent space as shown in the example in Figure 4.1 (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 $b(\mathcal{M}(x)) \neq b(\mathcal{M}(\bar{x}))$.

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 4.1 (right)). More formally, this translates 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 $\mathcal{M}$ with dimension k.

$$0 = p' \cdot (W_i x_i') = x_i' * \left(\sum_{j=1}^{k} p_j' w_{ji} \right) = x_i' * \sum_{j=1}^{k} \left(c_j - z_j' \right) w_{ji} \tag{4.1}$$

$$= \sum_{j=1}^{k} \left(c_j - \sum_{l=1}^{n} x_l' w_{jl} \right) w_{ji} \tag{4.2}$$

$$= \sum_{j=1}^{k} \left(c_j - \sum_{l \neq i}^{n} x_l w_{jl} - x_i' w_{ji} \right) w_{ji} \tag{4.3}$$

$$= \sum_{j=1}^{k} \left(c_j w_{ji} - \left(\sum_{l \neq i} x_k w_{jl} \right) w_{ji} - x_i' w_{ji}^2 - b_j w_{ji} \right) \tag{4.4}$$

$$\rightarrow x_i' = \frac{\sum_{j=1}^{k} c_j w_{ji} - \sum_{j=1}^{k} \left(\sum_{l \neq i}^{n} x_l w_{jl} \right) w_{ji}}{\sum_j w_{ji}^2} \tag{4.5}$$

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 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 changes at every iteration, and it is necessary for some refinement of the modification done before.

4.2 Experiments

We conducted two types of experiments. The first type of experiment aims 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 4.1 shows summary statistics on the datasets[2].

Table 4.1: 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

4.2.1 Latent Space Evaluations

First, we evaluate the quality of the latent space created by ILS. In line with [134], 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 [87], UMAP [135], TMAP [136], and VAE [130], described in Section 2.3. For ILS, we used the Adam optimizer [132] 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

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

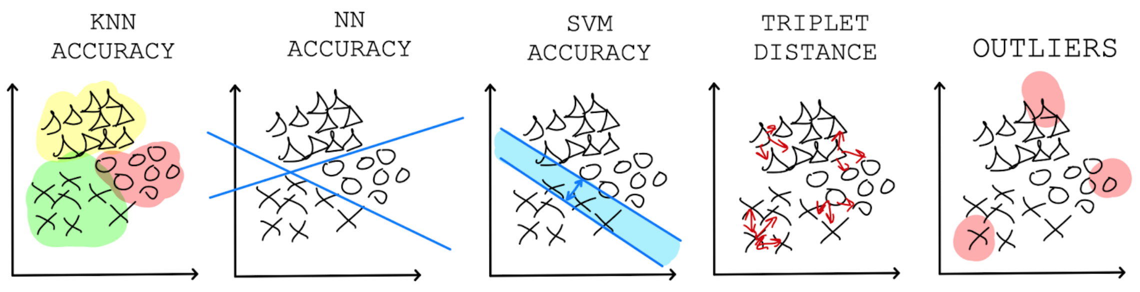


Figure 4.2: Metrics used to evaluate ILS

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.

Metrics. We considered two types of evaluation metrics: *space quality metrics* and *accuracy metrics* (Figure 4.2). 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 [134], 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) [137] 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) [138] classifier, a SVM [138] 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

Table 4.2: 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

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 dimensions while not exaggerating 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 4.2, we report the best latent space dimension results according to space evaluation metrics. Table 4.2 (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 4.2 (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[3].

In Figure 4.3, we report the scores of the classification models with varying latent dimension k for the `adult` dataset. Other datasets have similar behavior. Overall,

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

Table 4.3: 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

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, finding this dimension without trying them all is unclear, and ILS is not an exception. Table 4.3 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.

4.2.2 Counterfactuals Evaluations

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.

Competitors. We compare our proposal against two model-agnostic methods

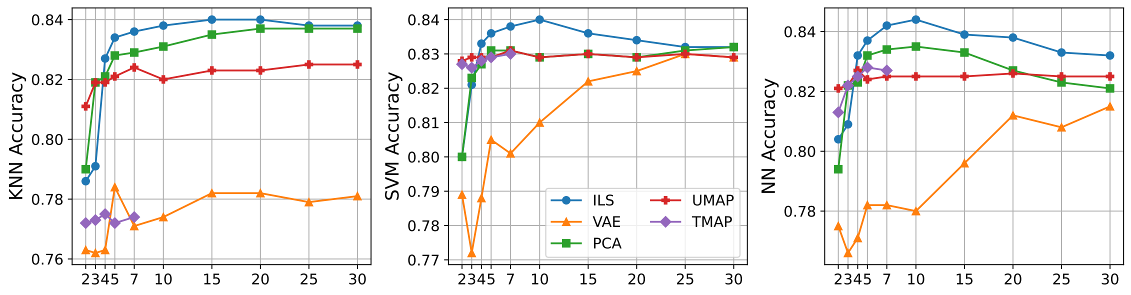


Figure 4.3: Accuracy of classifiers on `adult` varying the number of latent dimensions k.

Table 4.4: Example of counterfactuals produced by ILS, Gradient, and GSG for `adult`. Change to the original instance are highlighted in blue.

		age	hours	education	married	occupation	gender	country
	$\boldsymbol{x_1}$	**25**	**50**	**High School**	**Married**	**Factory Worker**	**Male**	**US**
$y_1 = 0$	ILS	25	50	High School	Not Married	Services	Male	US
$\hat{y}_1 = 0$	GD	45	80	Dropped	Married	Services	Male	US
	GSG	36	40	High School	Not Married	Factory Worker	Male	US
	$\boldsymbol{x_2}$	**32**	**50**	**College**	**Not Married**	**Services**	**Male**	**US**
$y_3 = 1$	ILS	32	50	College	Not Married	Prof Specialty	Male	US
$\hat{y}_3 = 0$	GD	29	48	College	Not Married	Services	Male	US
	GSG	31	51	College	Not Married	Services	Male	US
	$\boldsymbol{x_3}$	**31**	**50**	**Community College**	**Not Married**	**Gov**	**Male**	**US**
$y_4 = 0$	ILS	31	50	Community College	Married	Gov	Male	US
$\hat{y}_4 = 1$	GD	20	38	Community College	Not Married	Gov	Male	US
	GSG	30	51	Community College	Not Married	Gov	Male	US

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 [53] (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.

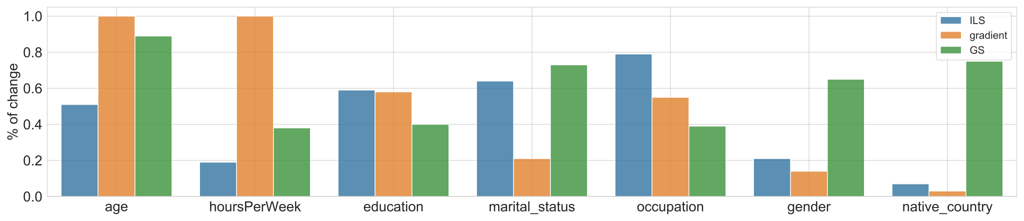


Figure 4.4: Histogram of the percentage of changes in the features among the three methods. GD and GSG algorithms focus more on the first two features, which are the continuous ones for the `adult` dataset.

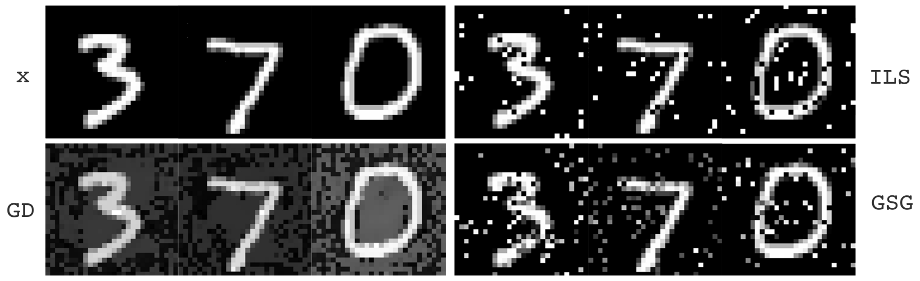


Figure 4.5: Example of counterfactuals produced by ILS, GD, and GSG for the `mnist` dataset. The counterfactual classes target are 8, 9, and 8 from left to right.

We selected these two approaches among the many available ones [139, 57] since they are two popular agnostic approaches to search for counterfactual explanations regardless of model and data type.

Qualitative Evaluation. We report in Tables 4.4 and Figure 4.5 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 more on the continuous features to change the prediction. For example, for x_1 of Table 4.4, 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 gen-

Table 4.5: Counterfactual explanations metrics evaluation. 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

erated counterfactuals for the whole data in the `adult` dataset and checked which features were modified by the method.

In Figure 4.4 is reported 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 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 4.5, all methods resemble adversarial attacks [2] 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 confusing, 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 [53] and search counterfactuals for a selected class. We decided to measure the proximity between x and its counterfactual $\bar{x}$ in two different fashions. 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.

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

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.

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

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 4.5. 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 humanely understandable since human reasoning often involves modifying only one feature at a time. In contrast, other methods can search for counter exemplars more plausible and more similar to the original example by altering more features. 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.

4.3 Discussion and Future Directions

In this chapter, we explored the possibility of creating the latent space transparently. We introduced ILS, a method that foresees the construction of an Interpretable Latent Space for simultaneously classifying and explaining. Through the use of an interpretable model, it is possible to create a latent space that maps the data by similarity. However, this space is difficult to evaluate, due to the lack of metrics in literature. We proposed a methodology to evaluate ILS against several approaches producing similar types of latent space, demonstrating that our proposal improves with respect to the state-of-the-art. 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. Moreover, the transparent nature of the transformation allows the position of points in latent space to be interpreted in terms of vectors, enabling counterfactual explanations of the classification methods built on it. We have shown how a counterfactual explanation can be produced using ILS and compared it to state-of-the-art explainers. ILS is able to produce counterexemplars that change fewer features than their competitors, allowing simpler explanations for humans. However, the counterexemplars created could be better both in terms of proximity and plausibility. This problem may be related to the search in latent space. A linear transformation is great for keeping the transformation interpretable, but there is no indication of the direction in which to move for finding counterfactuals. Our methodology uses cluster centroids of opposite classes as direction, however, this approach is not accurate since not all the clusters in the latent space are pure. Moreover, since the black-box is learned after the latent space is created, it is not possible to bind the space with its predictions. Another effect of the transparent nature of the latent space transformation is that it does not allow the latent space much freedom. Usually, when building a ML model, there is a trade-off between performance and interpretability [140, 141, 142], and ILS is no exception. The linear transformation is great for finding explanations, but it is limited in terms of expressiveness. This is quite clear in the counterfactuals created for the dataset `mnist`. The counterfactuals are confusing and quite far from a convincing explanation.

In the next chapter, we will explore the possibility of using black-box predictions to constrain the creation of latent space and create a clear direction in which to move to find explanations.

Chapter 5

Latent Space Post-Hoc Explanations

In this chapter we will try to answer our third research question: Can the latent space proprieties help us in producing better explanations than in the input space? What type of post-hoc explanations can be obtained? How do they compare with other approaches in the literature? As seen in Chapter 2 there are several ways for extracting explanations from black-box models [50]. One way to explain a prediction is in terms of similarity to other data. For example, "Your loan was denied because you are very similar to another person who defaulted on your loan." In particular, human categorization can be modeled as the use of prototypes: representative examples of the category as a whole [40]. The membership of an item in the category is determined by its similarity to the prototypes of the category. Following this idea, it is possible to extract a set of samples that best divide the data into different sets, each with its own characteristics. By observing the similarity of a data item to this dataset, it is possible to assign it to one of the prototype categories, allowing the user to better categorize the model decision.

A different intuitive way is instead to provide the changes to alter the outcome of the black-box. Counterfactual or *contrastative* thinking [48] is a concept in psy-

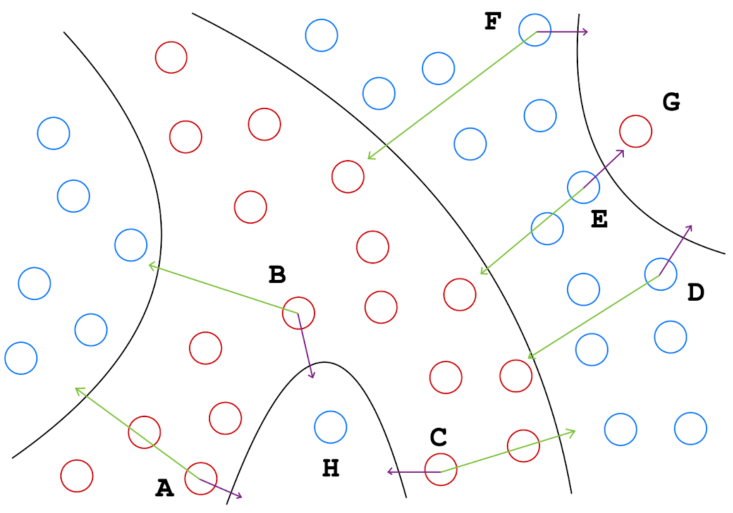


Figure 5.1: Representation of a possible adversarial problem in counterfactual explanations. Points G and H can be selected as counterfactual explanations even if they are clearly adversarial examples.

chology that involves the human tendency to create possible alternatives to events that have already occurred; something that is contrary to what actually happened. In XAI these changes can be given in terms of records with similar characteristics as the instance of interest but with an altered outcome. For instance, a record with different features or a text with missing/added words [12, 49]. In this sense, we speak about instance-based counterfactual explanations [50] or counter-exemplars [51]. In the following, we adopt the short name counterfactual explanations to refer to this type of explanation.

Counterfactual explanations are among the most widely used tools in XAI due to their simplicity and understandability and they can be classified under several aspects [9]. However, a counterfactual is usually defined as the minimal changes that alter a model's prediction. This definition overlaps with adversarial attacks: a type of attack that is made on ML models in which slightly altered data are proposed to the model that is nevertheless classified in a totally different way. Adversarial attacks are used to demonstrate the fragility of a black-box, whereas a counterfactual should highlight the model's decisions and not try to break it. In literature, the difference between a counterfactual and an adversarial sample is not clear. In this chapter, we defined *Robustness*: the property of a counterfactual explanation to be free from adversarial attacks. This metric can be put together with the other metrics presented in Section 2.1.8 used to evaluate the goodness of a counterfactual explanation. The presence of an adversarial sample in the data can alter counterfactual explanations in dangerous ways. In Figure 5.1 this problem is highlighted in which two adversarial samples can be selected as an explanation for an entire set of data points.

Our objective is to exploit a *latent space representation* of the data to search for Counterfactual and Prototype explanations while respecting the criteria of *Robustness* as well the others presented in Section 2.1.8 (*Proximity*, *Plausibility*,*Diversity*,*Accuracy* and *Sparsity*). By placing data in the latent space based on similarity, it is easier to measure distances and better categorize data into groups, allowing us to find better prototype explanations. Beside that latent space models are perfect for creating counterfactual explanations since they can generate data similar to the original one [143, 144]. Having both prototype and counterfactual explanations is a powerful way to gain a deeper understanding of the behavior of machine learning models. By examining both types of explanations together, we can better understand the strengths and limitations of the model, and gain insights into its inner reasoning. Overall, the combination of prototype and counterfactual explanations provides a powerful tool for understanding and improving the behavior of machine learning models.

However, the creation of latent space representations is unsupervised, meaning that there is no indication of how to move in the space to alter the prediction of an instance. Another problem is that the mapping between the input features and the latent space is done by neural networks (NNs), which introduces several nonlinear transformations making the relationships between latent space dimensions and real features incomprehensible for humans [145]. This makes it difficult to recover the counterfactuals found in the encoded latent space.

In this chapter, we present a method for searching Counterfactual and Prototypical explanations via Interpretable Latent Space (CP-ILS). CP-ILS produces local prototypes and counterfactual explanations able to overcome the problems highlighted above by exploiting a transparent latent space representation. Indeed, CP-ILS builds a transparent latent space representation by using a linear approach and by taking advantage directly of the predictions of a given black-box model. CP-ILS uses the real space and the predictions of a black-box to construct a latent space. Prototype explanations are achieved by looking at similarities in this custom designed space, while counterfactual explanations are obtained in the form of diverse instances following the prediction direction of the data. The counterfactual explanations are decoded back into the real space using the linear nature of the transformation, making the whole process transparent. Our experiments demonstrate that it is possible to use the interpretable latent space representations of CP-ILS for generating explanations for tabular data obtaining better results than current explanation methods taken individually. The results show that CP-ILS outperforms several state-of-the-art counterfactual explanation methods w.r.t. metrics quantifying *Proximity*, *Robustness*, *Plausibility*, and *Diversity* and prototype methods regarding *Accuracy* and *Sparsity*. Also, thanks to its linear latent space approach, CP-ILS enables the actionability of the counterfactual explanations. The user can select which features of the input wants to modify, and CP-ILS will produce a counterfactual with only that subset of features changed, if possible. We focus our analysis on tabular data. However, we highlight that our proposal remains valid for every data type.

5.1 Methodology

This section describes the proposed Counterfactual and Prototypical explanations via Interpretable Latent Space (CP-ILS), and it is divided into two parts. The first part reports the methodology used to learn the latent space representations, and the second part focuses on how to exploit the latent space to produce the counterfactual and prototype explanations.

5.1.1 Constrained Interpretable Latent Space with Black Box Predictions

CP-ILS is based on a linear transformation which enables the transparent mapping between the input features and the latent ones. We opted for this direction because linear models have been proven in recent years to be the best methodology to produce explanations [25, 146]. Indeed, through linear transformations, it is possible to isolate the contribution of each feature to the prediction. The idea is to create/learn a latent space by combining a similarity loss analogous to the one utilized in Chapter 4 in which the data are mapped into the space based on the similarity between them. However, this time the latent space is also created using black-box predictions, increasing the data information.

The pseudo-code is illustrated in Algorithm 6. Formally, CP-ILS learns a linear model $\mathcal{M}$, which maps the input features X into a latent one Z by using a linear transformation: $Z = \mathcal{M}(x) = W \cdot x = w_0x_0 + w_1x_1 + \cdots + w_ix_i + \cdots + w_nx_n$, where W are the weights of the transformation, x is an instance belonging to the input space $\mathbb{R}^n$, z its transformation to the latent space $\in \mathbb{R}^k$, and k is the number of latent dimensions. Hence, the first objective is to find the "best" weights W for the linear model $\mathcal{M}$, given a specific dimension k.

CP-ILS maps the data X into the latent space Z based on data similarity. Also, we want this mapping to be transparent to not lose the link between the latent and the real features and be able to decode easily from the latent space back into the real one without the use of any extra ML model. If we relax the hypothesis of linearity, then to decode back in the input space there is the need for a decoder.

Algorithm 5: $CP{-}ILS(X, x, b)$

```
Input  : $X$ - data, $x$ - instance to explain, $b$ - black-box
Output: $C$ - set of counterfactual explanations,
        $P$ - set of prototype explanations
1  $i \leftarrow 0$;                                      // init. iteration index
2  $n \leftarrow length(x)$;                              // compute features length
3  $k \leftarrow 2$;                                      // init. latent space dimensions
4  $\mathcal{S}_i \rightarrow \infty$;                    // init. latent space score
5  $\hat{y} \leftarrow b(X)$;                             // obtain the black-box predictions
6  while $\mathcal{S}_{i-1} < \mathcal{S}_i \wedge k < n$ do
7      $\mathcal{M} \leftarrow$ LatentLearning$(X, b, k)$;        // learn the model
8      $Z \leftarrow \mathcal{M}([X, \hat{y}])$;                 // compute latent space for X
9      $\mathcal{S}_i \leftarrow$ Score$(Z)$;                     // compute the new score
10     $k \leftarrow k + 1$;
11     $i \leftarrow i + 1$;
12 $P \leftarrow$ PrototypeSearch$(x, X, \mathcal{M}, b, n)$;   // compute the prototype explanation
13 $C \leftarrow$ CounterfactualSearch$(x, \mathcal{M}, b)$;    // compute the counterfactual explanation
14 return $(P, C)$;
```

The weights W of the transformation need to be learned from the data X based on feature similarity. The concept of similarity between two samples has been consistently seen in terms of feature similarity. However, we think that the prediction similarity between samples should be also taken into account. The prediction label $\hat{y} = b(x)$ adds more information into the latent space representation and contributes to identify the relevant features to alter the prediction. As illustrated in Figure 5.2, creating a latent space using black-box predictions allows more correct examples to be found. The red circle H at the bottom of the original space (left) cannot be a good counterfactual example for points A, B, and C because it will probably be an adversarial point. On the other hand, in the latent space (right), the data are rearranged according to the similarity of both features and prediction outcome, allowing for more diverse and less contradictory counterfactuals.

However, using only the prediction as similarity data near the decision boundary will be mapped close to points that are very far away. Instead, by using the probability of the prediction, the model can effectively distinguish between these types of

Algorithm 6: *LatentLearning*(X, b, k)

```
Input  : $X$ - training data, $k$ - latent dimensions, $b$ - black-box
Output: $M$ - trained transformation model
1  $i \leftarrow 0$;                                      // init. iteration index
2  $L_i \leftarrow \infty$;                                // initialize loss
3  $\mathcal{M} \leftarrow$ init();                        // initialize model weights
4  $\hat{y} \leftarrow b(X)$;                              // obtain the black-box predictions
5  $S_X \leftarrow PairwiseDistance(X)$;                   // original sim. matrix
6  while $L_{i-1} > L_i$ do                                // until the loss decreases
7      $Z \leftarrow \mathcal{M}([X, \hat{y}])$;           // get latent representation
8      $S_Z \leftarrow PairwiseDistance(Z)$;               // latent sim. matrix
9      $L_i \leftarrow KLD(S_X, S_Z)$;                     // compute KL-Divergence loss
10     $\mathcal{M} \leftarrow update(\mathcal{M}, L_i)$;  // update using backprop.
11     $i \leftarrow i + 1$;
12 return $\mathcal{M}$;
```

data in the latent space. Two points close in the latent space will have similar features as well as similar prediction probability given by the black box. Therefore, CP-ILS uses gradient optimization [132] to minimize a loss that encourages both feature and prediction similarity. In particular, as loss function L, CP-ILS adopts a modified similarity probability loss introduced in Section 4.1.1. Similar to Section 3.2, CP-ILS concatenates to the data x the predictions probabilities $\hat{y}$ given by the black-box b, i.e., $[x, \hat{y}]$. Thus, we have $Z = W \cdot [x, \hat{y}] = w_0x_0 + w_1x_1 + \cdots + w_ix_i + \cdots + w_nx_n + w_{\hat{y}}\hat{y}$. More in detail, the similarity between two records x_j and 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. More formally:

$$PairwiseDistance(x_i, x_j) = \frac{\exp\left(-d(x_i, x_j)/2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-d(x_i, x_k)/2\sigma_i^2\right)} \tag{5.1}$$

where x_i and x_j are two records for which we compute the similarity, σ is the variance of the Gaussian, and d the distance between x_i and x_j. The more two samples are similar, the higher this value.

There are several ways to compute the distance d between two records. CP-ILS

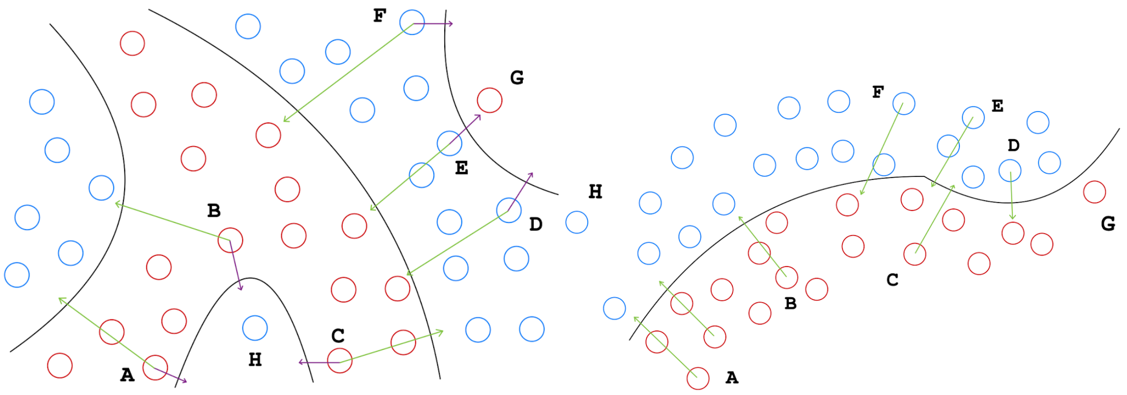


Figure 5.2: Representation of the original data (left) and the rearrangement of CP-ILS (right). The two points H and G are clearly two adversarial examples that can fool the counterfactual methods to believe that these are valid explanations for the other data. By reorganizing data by similarity and prediction the resulting counterfactuals are less adversarial, more plausible, and also more diverse than if found in the original input space.

computes the distance between the input features and sums the distance between the black-box prediction. However, the Euclidean distance does not work correctly with categorical values. To solve this problem that is typical of many counterfactual explainers [9], CP-ILS computes the distance between the categorical and continuous input features using a mixed distance [59] composed of two additive terms: the normalized Hamming distance [147] for categorical features, and the cosine distance [148] for continuous features. The three distances are computed separately and then added together (Equation 5.2).

$$\begin{aligned} d(x_i, x_j) = & \left(\frac{h}{m}\right) \cdot Hamming(x_i, x_j) + \\ & \left(\frac{m-h}{m}\right) \cdot Cosine(x_i, x_j) + \\ & \|\hat{y}_i - \hat{y}_j\|_2) \end{aligned} \tag{5.2}$$

where h is the number of categorical features, and m is the total number of features. From a computational point of view, the issue with using the similarity loss defined in Equation 5.1 is that it requires computing the similarity between every pair of

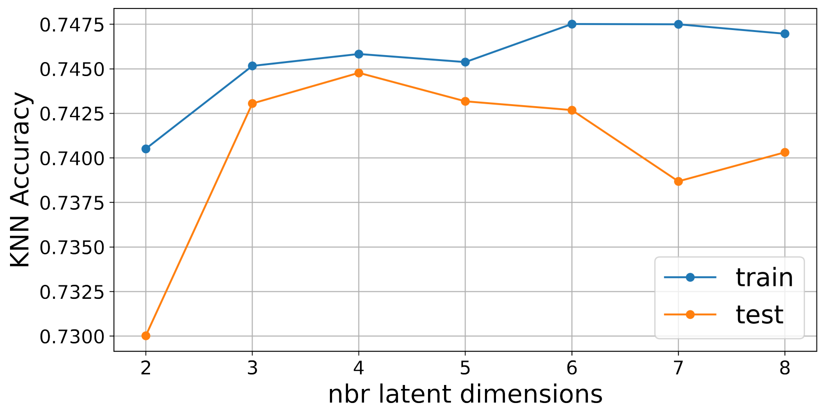


Figure 5.3: Score computed for different latent space dimensions for the `Adult` dataset with SVM as black-box. As a scoring function, we used the accuracy of a KNN model. The metric computed indicates four as the best dimension.

instances in the real space and in the latent space. However, since the fit is performed by gradient optimization, it is possible to divide the data into fixed-sized batches and compute the similarity matrices separately for any small batch of samples. 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 records in Z changes after every iteration. Finally, the loss function minimized by CP-ILS is the Kullback–Leibler divergence [133] between the matrices of distances: $L(S_X, S_Z) = KL(S_X \| S_Z)$, 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 until convergence[1].

When dealing with latent space representations, an essential parameter to select is the number of latent space dimensions k. The size k of the latent space is often chosen unwisely in the literature, usually based on the performance achieved in terms of reconstruction of the input [149]. Increasing the size of the latent space allows the space to store more information. This causes the creation of a particular dimension for which only noise is added for subsequent dimensions, resulting in a kind of

[1]For the convergence problem, we used the Early Stopping technique.

Algorithm 7: *PrototypeSearch*$(x, X, \mathcal{M}, b, n)$

```
Input  : $x$ - Data to explain, $X$ - reference data, $\mathcal{M}$ - latent space model, $b$ -
         black-box, $n$ - nbr of prototype to produce
Output: $P$ - Prototype explanation
1 $\hat{Y} \leftarrow b(X)$;                          // compute data prediction in the latent space
2 $Z \leftarrow \mathcal{M}(X, \hat{Y})$;             // compute latent space position
3 $centers \leftarrow \text{KMeans}(Z)$;              // obtain the centers
4 $\hat{y} \leftarrow b(x)$;                          // compute data label in the latent space
5 $z \leftarrow \mathcal{M}(x, \hat{y})$;             // compute latent space position of the data to explain
6 $P \leftarrow \underset{n}{\text{argmin}}[d(z, c)]$;  // find the $n$ most similar prototypes
7 return $P$;
```

overfitting behavior. To avoid this effect, CP-ILS starts with the lowest possible dimension of latent space, i.e., $k = 2$, and progressively increases the dimensions. To identify possible overfitting behaviors, we tested several metrics, and we decided to use the accuracy of a KNN [150] classifier trained on top of the created latent space Z. The accuracy is computed on a validation set. In Figure 5.3 there is represented an example of such behavior. As the dimensions increase, the KNN score computed in the validation set performs better and better until we reach the fourth dimension, where it starts decreasing. CP-ILS selects the last latent space dimension before this accuracy drop.

After selecting the best latent space, CP-ILS begins the procedure to search for counterfactuals in the latent space and decode them back into the real space using the linear propriety of the learned mapping.

5.1.2 Post-Hoc Explanations

In this section, we describe how is possible to use latent space representations learned by CP-ILS in the section before to search for Prototype and Counterfactual explanations of the black-box b.

Formally, CP-ILS explanation is composed of the pair (P, C) where P is the set of Prototypes most similar to the sample, and C is the set of possible Counterfactuals.

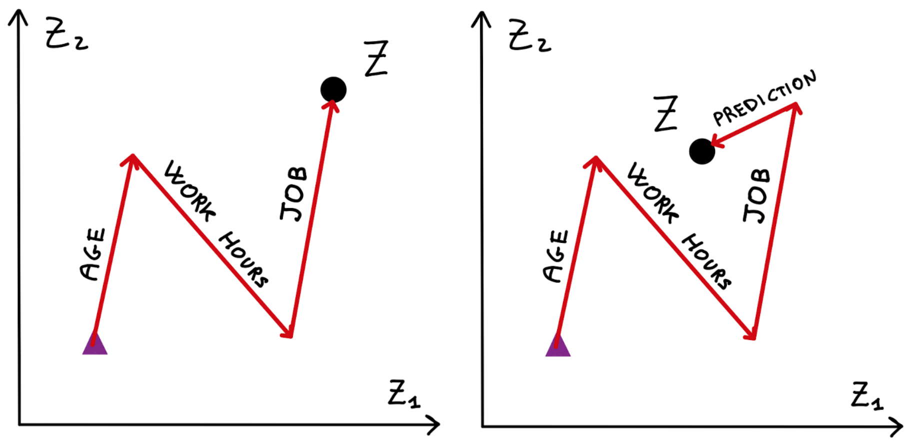


Figure 5.4: Representation of the contributions of the input features in the latent space. By concatenating the prediction of the black to the input we obtain a prediction direction along which we have similar points but different predictions.

The procedure to find the set of prototypes P is illustrated in Algorithm 7. After we learned the weights W of the model $\mathcal{M}$ we fix their values to fix the latent space. Then CP-ILS applies KMeans clustering [151] on a reference population X mapped into the latent space, to obtain a set of possible Prototypes which are the cluster centers. Given an instance x, CP-ILS maps it into its latent space position z with the procedure explained in the previous section. Then CP-ILS selects, from the set of possible Prototypes the n most similar using the Euclidean distance d. We selected as reference population X the training set, so the set P is the set of the training data closest to the centers found.

After retrieving the prototype set, CP-ILS switches to Counterfactuals explanations. For counterfactual explanations, we need to firstly understand how the input features are mapped into the space. In particular, each input feature x_i is responsible for a direction of movement in the latent space as shown in the example in Figure 5.4 (left). The dot product of the model $\mathcal{M}$ can be decomposed into a sum of contributions whose sum gives the latent position z. Indeed, the contribution of an input feature x_i to a latent space with k dimensions can be seen as a vector of dimensions equal to the dimensions of the latent space. Every feature in the input

Algorithm 8: *CounterfactualSearch*$(x, \mathcal{M}, b)$

```
Input  : $x$ - sample to explain, $\mathcal{M}$ - latent space model, $b$ - black-box, $s$ - latent
         space step
Output: $C$ - set of counterfactual explanations
1  $\hat{y} \leftarrow b(x)$;                                         // compute sample prediction
2  $z \leftarrow \mathcal{M}(x)$;                                     // compute latent space position
3  $C \leftarrow \emptyset$;
4  for $idx \in combination(n)$ do
5      while $b(x + \Delta) = \hat{y}$ do                             // until same prediction
6          $v \leftarrow z + s * \|w_y\|$;                            // set the destination
7          $\Delta \leftarrow$ LagrangeMultipliers$(idx, v)$;         // compute optimization
8          $s \leftarrow s + \epsilon$;                               // compute new step
9      $C_i \leftarrow x + \Delta$;
10 return $C$;
```

space contributed to a vector of k dimensions, which, when all added together, sum up to the latent position $z = W \cdot [x, \hat{y}] = w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_n \cdot x_n + w_{\hat{y}} \cdot \hat{y}$, where the weight w_i is the latent direction of the respective feature x_i. Since CP-ILS concatenates the prediction $\hat{y}$ of the black-box b to the input we obtain in the latent space representation an additional direction $w_{\hat{y}}$ which we call the *prediction direction*. Alongside this direction we have samples with similar features but with different predictions, which is crucial for finding counterfactuals.

The procedure to find a counterfactual is illustrated in Algorithm 8. Firstly, CP-ILS encodes x into its latent space position $z = W \cdot [x, \hat{y}]$. The next step is to move from the encoded sample, toward the prediction direction until the black-box change prediction. As illustrated in the left of Figure 5.5, the prediction direction is the violet arrow, while the effect of the real features on the latent space is represented by the black vectors. CP-ILS finds the minimum changes Δ to the input features in order to move toward the prediction direction. CP-ILS does that by taking advantage of the linearity of the model $\mathcal{M}$. Given an input data x, we need to find the minimum changes Δ to the input features to move in a new position v which is located along the w_y direction. This is an optimization problem, and since the mapping is obtained linearly is possible to solve it using the Lagrange multipliers [152].

$$\begin{cases} \nabla f(\vec{x}) = \sum_{l=1}^{l} \lambda_l \nabla g_k(\vec{x}) \\ g_1(\vec{x}) = \cdots = g_k(\vec{x}) = 0 \end{cases}$$

where f is the function to minimize, g are the constraint functions and λ_l are the multipliers. As function f to minimize, CP-ILS uses the square of Δ multiplied by one half, to simplify the gradient calculations. The constraints g are derived from the latent space mapping, we have one constraint for every latent dimension.

$$\begin{cases} f = \frac{1}{2} \sum_{i=1}^{n} \Delta_i^2 \\ g_l = \sum_{i=1}^{n} W_{il}(x_i + \Delta_i) - v_l = 0 \quad l = [1, k] \end{cases}$$

where n is the number of features of x, v_l is the value of the latent space position in which we want to move for the l-th dimension with $l \in [1, k]$, and k are the dimensions of the latent space. By substituting f and g in the Lagrange multiplier equation, we obtain:

$$\begin{cases} \Delta_i = \sum_{l=1}^{k} \lambda_l W_{il} \\ \sum_{i=1}^{n} \sum_{r=1}^{k} \lambda_l W_{ir} W_{il} = v_l - \sum_{i=1}^{n} x_i W_{il} \quad l = [1, k] \end{cases}$$

This linear system can be quickly solved using a linear solver. The solutions of this system are the changes Δ that satisfy f and can be applied to the input x to modify its latent position to v. However, we do not know exactly in the prediction direction where the prediction of the black-box would change. To overcome this CP-ILS select v moving along the prediction direction of a fixed step s: $v = z + s \cdot ||w_y||$. After computing the changes, CP-ILS queries the black-box to check if the prediction $\hat{y}$ has changed, if not another step is performed. In Figure 5.5 the search is performed

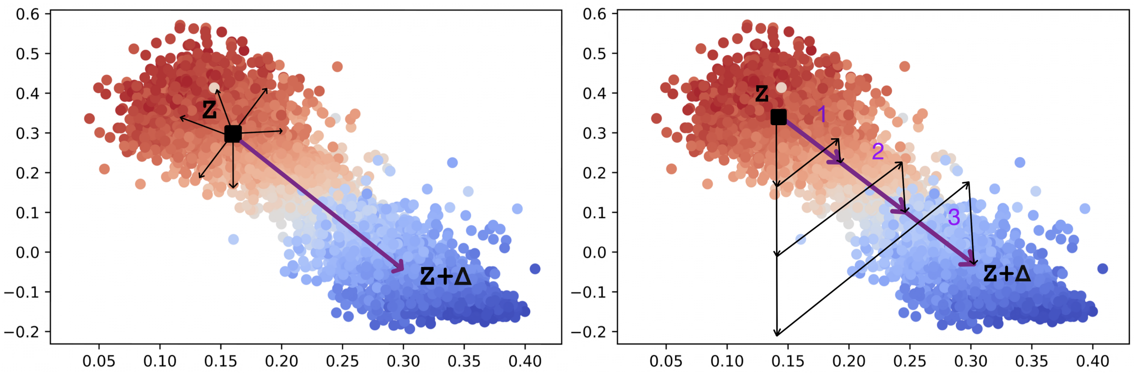


Figure 5.5: Visualization of the latent space created using two dimensions. The black square represents a sample z for which to look for counterfactuals. The violet arrows represent the path to follow, the black arrows are instead the effects of the input features on the latent space. On the *left*, we have the ideal combination of the feature to move to $z + \Delta$ while on the *right* the same movement is obtained with the progressive search.

in three steps toward the prediction direction. From the starting position z, until $\hat{y}$ changes[2].

This procedure finds only one counterfactual sample for a given input x, and the Δ are computed for all the input features. To meet the *Diversity* propriety requested for a counterfactual method [54], we apply the minimization to different reduced combinations of the features of x (Line 4). Since it would be impossible to compute all the possible combinations due to exponential computing time, we only compute the search for a fixed number of features. This is reasonable since a counterfactual explanation with fewer features modified is better to comprehend from a human standpoint [9]. Also, this enables the actionability of the counterfactual explanation. The user can select only the features he wants to modify and keep fixed other ones that are impossible to change, e.g., the age or the sex of a person. The minimization with Lagrange multipliers can be reduced to a limited set of features. The user can provide as input to CP-ILS a list of features that he or she wants to change, and minimization is only performed on that reduced set of features. CP-ILS finds the combination of these selected features that most shifts along the opposite prediction

[2]we used $\epsilon = 0.1$ as starting step and made an increment step of 0.1

direction. Thus a different counterfactual limited to that reduced set of features is found.

5.2 Experiments

In this section, we report the experiments carried out to validate CP-ILS. First, we illustrate the datasets used, the classifiers, and the experimental setup. Then, we evaluated the goodness of the latent space created by CP-ILS. Finally, we evaluate the prototype and counterfactual explanations found both in a qualitative and quantitative way.

Datasets and Black-box Models. We ran experiments on a selection of three datasets widely referenced for classification tasks and publicly available from the UCI ML repository[3]. We tested on `adult`, `compas`, and `german` datasets. In addition, we added a real-world proprietary dataset called `diva`[4]. The datasets are split: 80% training, 10% validation, and 10% test set. As black-box models, we tested four different methods including XGBoost[5]. (XGB) [153], Random Forest (RF) [154], Support Vector Machines[6] (SVM) [155], and standard dense Neural Networks[7] (NN). We performed five-fold stratified cross-validation for every black-box. Dataset descriptors and black-box performance are reported in Table 5.1.

Metrics. To evaluate the goodness of the latent space created we measured the relative positioning of neighborhoods. We sampled random observations and computed the Random Triplet Accuracy [134], 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 measured the outliers' preservation because we want an outlier in the input space to remain an outlier in the latent space. We

[3]https://archive.ics.uci.edu/ml/index.php
[4]https://kdd.isti.cnr.it/project/diva
[5]Implemented as `xgboost` https://xgboost.readthedocs.io/en/stable/.
[6]Implemented as `scikit-learn` https://scikit-learn.org/stable/.
[7]Implemented as the `PyTorch` library https://scikit-learn.org/stable/

Table 5.1: Description of datasets and performance of the black-boxes

	adult	compas	german	diva
nbr features	8	21	20	88
nbr instances	48842	6171	1000	8000
XGB	.757	.956	.759	.927
RF	.763	.966	.778	.905
SVM	.763	.944	.777	.859
NN	.755	.943	.775	.864

used the Local Outlier Algorithm (LOF) [137] 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.

For CP-ILS we used the KNN accuracy metrics described in Section 5.1.1 to select the best latent space dimensions. Since only CP-ILS can select the best latent space dimension, for other approaches we trained every algorithm on different latent space dimensions and evaluated the metrics for every dimension. Then we selected the best scores among the dimensions. We tested the following latent dimensions $K = \{2, 3, 4, 5, 7, 10, 15, 20, 25, 30\}$ to cover most of the possible dimensions while not exaggerating in computational times.

In Table 5.2 we report the results of the best latent space dimension according to space evaluation metrics. Table 5.2 (left) shows that CP-ILS is the best to preserve distances, with PCA as 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 5.2 (right), shows the results of the outlier preservation metric. We do not observe a clear winner with respect to this score. Overall, CP-ILS, PCA, and TMAP seem better than VAE and UMAP.

Table 5.2: Space quality metrics. Best scores are in bold, second best result are underlined.

	Random Triplet Accuracy					Outlier Preservation				
XGB	CP-ILS	VAE	PCA	UMAP	TMAP	CP-ILS	VAE	PCA	UMAP	TMAP
adult	**.7833**	.7218	.7709	.7318	.6093	.0025	**.0010**	.0016	.0022	.0024
compas	**.8535**	.7191	.8140	.7369	.6863	.0009	.0014	.0012	.0012	**.0001**
german	**.8042**	.7272	.7972	.7498	.7512	**.0012**	.0029	.0014	.0021	.0028
diva	**.8531**	.7215	.7593	.7122	.7286	.0005	.0006	**.0003**	.0013	.0007
NN	CP-ILS	VAE	PCA	UMAP	TMAP	CP-ILS	VAE	PCA	UMAP	TMAP
adult	**.8647**	.7525	.8234	.7167	.6167	.0019	.0013	.0018	.0029	**.0006**
compas	.8735	.7152	**.8923**	.7186	.6152	.0013	**.0009**	.0017	.0027	.0025
german	**.8134**	.7124	.8089	.7285	.7145	.0021	.0022	.0013	.0018	**.0012**
diva	**.7827**	.6242	.7635	.7527	.6204	.0004	.0007	.0012	**.0003**	.0009
RF	CP-ILS	VAE	PCA	UMAP	TMAP	CP-ILS	VAE	PCA	UMAP	TMAP
adult	**.8664**	.7109	.8103	.6067	.6129	.0020	.0022	.0022	.0011	**.0001**
compas	**.8235**	.7325	.8017	.7096	.6105	**.0009**	.0016	.0013	.0013	.0014
german	.7982	.7109	**.8196**	.7215	.6908	.0012	.0016	**.0006**	.0026	.0015
diva	.7010	.6849	.7434	**.7683**	.6737	.0004	.0009	**.0001**	.0013	**.0001**
SVM	CP-ILS	VAE	PCA	UMAP	TMAP	CP-ILS	VAE	PCA	UMAP	TMAP
adult	**.8657**	.7093	.8026	.7189	.6093	.0016	.0028	**.0002**	.0026	.0019
compas	.8220	.7109	**.8248**	.7098	.6978	**.0009**	.0023	.0015	.0013	.0028
german	**.7728**	.7267	.7708	.7590	.7098	.0012	.0006	**.0002**	.0022	.0016
diva	.7415	.7018	**.7459**	.7094	.6570	.0003	.0024	.0016	**.0001**	.0004
wins	11	0	4	1	0	3	2	5	2	5

5.2.1 Prototype Explanation Evaluation

To prove the validity of our approach we propose in this section two experiments: a simple example that makes intuitive sense of the goodness of clusters, then we compared our approach against several competitors.

Firstly we took two very simple datasets and looked at the clusters created in the latent space. We choose the `iris` dataset [156], as it is very simple and clear, and also we created a synthetic dataset using two Gaussian distributions. We then trained a `xboost` black-box model on it and create the latent space using the technique described in Section 5.1.

As seen in Figure 5.6, the clusters are well-arranged and more separated in the latent space, making it easier to identify patterns and relationships between

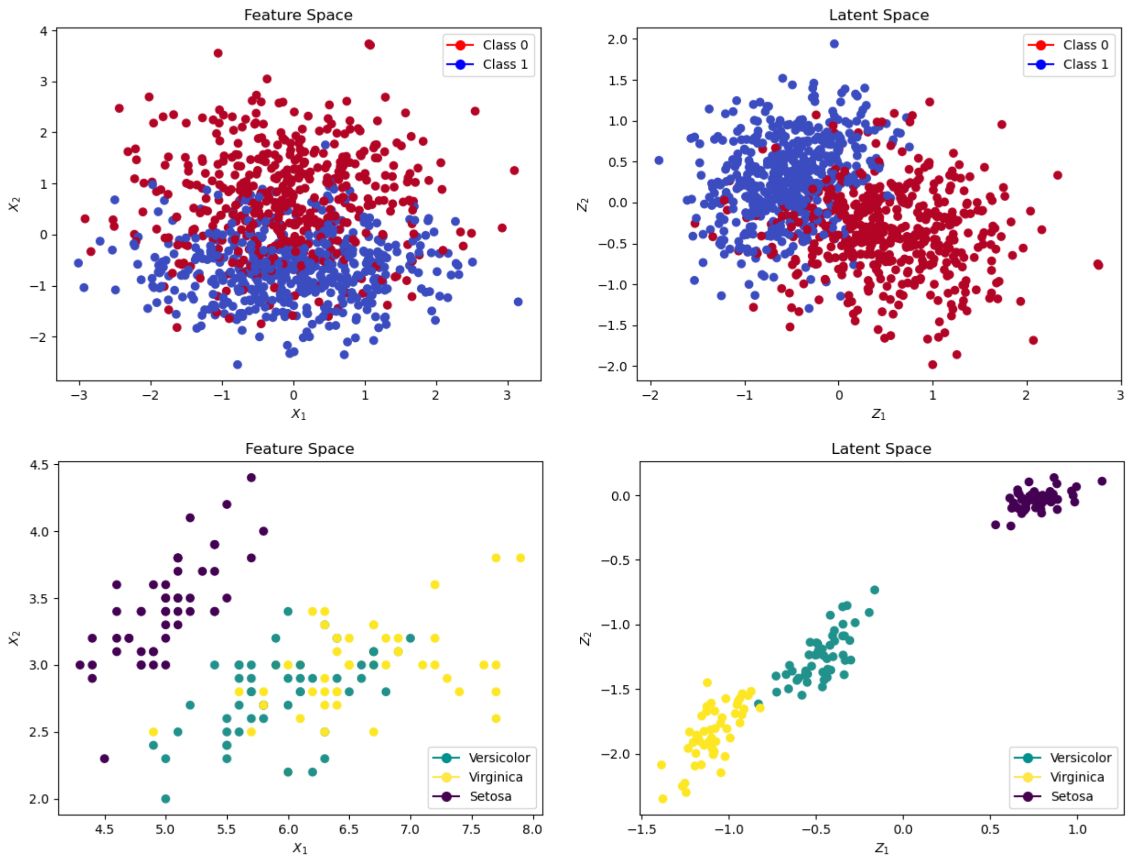


Figure 5.6: Visualization of the cluster in the latent space or `iris` and a synthetic dataset.

data points. The similarity loss helped to reduce noise and highlight meaningful features in the data. The resulting clusters are arranged in a logical and interpretable manner, allowing for more accurate clusters which will lead to meaningful insights into the underlying structure of the data. The well-arranged and well-separated clusters in the latent space demonstrate the effectiveness of these techniques in improving the accuracy and interpretability of data analysis and modeling.

Competitors. We compared our proposal against different algorithms present in the literature presented in Section 2.1.5 capable of producing prototype explanations. We selected the most popular methods that use different approaches to the problem with different focuses. In particular, we tested our approach against MMD-Critic (MMD), Proto-Select (P-Sel), and Proto-DASH (P-DASH). These algorithms were designed to explain the label of the data rather than predictions. In order to make a fair comparison with our approach, we used the prediction of the black-box rather than the labels in the searching phase of every method.

Metrics. To evaluate the goodness of the prototypes explanations, we evaluated on the test sets of every dataset two types of metrics that measure the *1-KNN Accuracy* and the *Sparsity* for every set of prototypes found by the different explainers. We measured the *1-KNN Accuracy* of a set prototype explanations P using a 1-knn classifier. The 1-knn accuracy metric is defined as the accuracy of a 1-knn classifier built on the set of prototypes found by the explainer. The classification of a new sample is defined as the classification of the most similar prototype in the set of all prototypes.

$$1 - KNNAccuracy = \frac{1}{n}\sum_{i=0}^{n} \mathbb{1}\left[b\left(\min_{p\in P}[d(X_i, p)]\right) = y_i\right]$$

where $\mathbb{1}$ returns 1 if [*condition*] is true, 0 otherwise, b is the black-box, X is the test set of n instances, P is the set of prototypes, and y are the labels of the test set. The closer to 1 the better.

We want a prototype to resemble the data it represents as much as possible, both in terms of similarity and black-box prediction. We called this propriety of a prototype explanation *Sparsity*, and we defined it as the percentage of data with the same prediction as the closest prototype. More formally:

$$Sparsitiy = \frac{1}{n}\sum_{i=0}^{n} \mathbb{1}\left[b(X_i) = b(p_{X_i})\right]$$

where $\mathbb{1}$ returns 1 if [*condition*] is true, 0 otherwise, n is the dimension of the set X, p_{X_i} is the prototype belonging to the set P closest to X_i, and b is the black-box. The closer to one the better.

Results. One crucial parameter to select is the number of prototypes to be

Table 5.3: Results of the metrics computed on `adult` and `german` datasets and on the four black-boxes. The best results are highlighted in bold, second bests are underlined.

	1-KNN Accuracy				Sparsity			
XGB	CP-ILS	P-Sel	P-DASH	MMD	CP-ILS	P-Sel	P-DASH	MMD
`adult`	**.8506**	.7506	.8094	.8137	**.8254**	.7503	.8121	.8117
`compas`	**.9212**	.8997	.8501	.9026	**.9001**	.8841	.8264	.8889
`german`	**.7804**	.7400	.7252	.7356	**.7890**	.7273	.6648	.7375
`diva`	**.7763**	.7670	.6386	.7740	**.8301**	.8059	.7524	.7573
NN	CP-ILS	P-Sel	P-DASH	MMD	CP-ILS	P-Sel	P-DASH	MMD
`adult`	**.8826**	.7602	.8431	.8451	.8292	.7605	.8422	**.8487**
`compas`	**.9588**	.9191	.8533	.9236	.9126	.8901	.8152	**.9195**
`german`	**.7959**	.7650	.7501	.7905	**.7993**	.6471	.7532	.7893
`diva`	.7876	.7275	.6302	**.7975**	.7986	.7341	.6795	**.8173**
RF	CP-ILS	P-Sel	P-DASH	MMD	CP-ILS	P-Sel	P-DASH	MMD
`adult`	**.9102**	.8057	.8446	.8483	**.8793**	.8059	.8342	.8593
`compas`	**.9218**	.9149	.8648	.8296	.8804	**.9093**	.8233	.8377
`german`	.8704	.8406	.8453	**.8705**	**.8921**	.7796	.8377	.8885
`diva`	**.8142**	.7723	.6370	.7857	**.8277**	.8027	.7605	.7565
SVM	CP-ILS	P-Sel	P-DASH	MMD	CP-ILS	P-Sel	P-DASH	MMD
`adult`	**.9432**	.8065	.8348	.8832	.8312	.8065	.8023	**.8726**
`compas`	**.9403**	.9248	.8794	.9137	**.9359**	.9000	.9000	.8853
`german`	.9308	**.9403**	.9208	.9402	**.9695**	.9419	.9012	.9600
`diva`	**.8554**	.8062	.7754	.7878	.8252	.7776	**.8677**	.8308
wins	13	1	0	2	10	1	1	4

included in the final set of explanations. Too low a value leads to an overgeneralized set of prototypes that ignore some particular data set. A value that is too high, on the other hand, carries the risk of overfitting and obtaining redundant prototypes. We measured the two metrics for several prototype set dimensions. The best results obtained for every method are reported in Table 5.3. Our approach outperforms the competitors in *1-KNN Accuracy* metric in every dataset and black-box apart from NN in the `diva` dataset. Regarding *Sparsity*, our approach is evidently better while struggling more especially when using NN and SVM black-boxes. In the `adult` and `german` datasets, MMD-critic is the second best method which is surprising due to its unsupervised nature. In `compas` and `diva` is outperformed by Proto-select and Proto-DASH. In these two datasets, the black-boxes are better performing

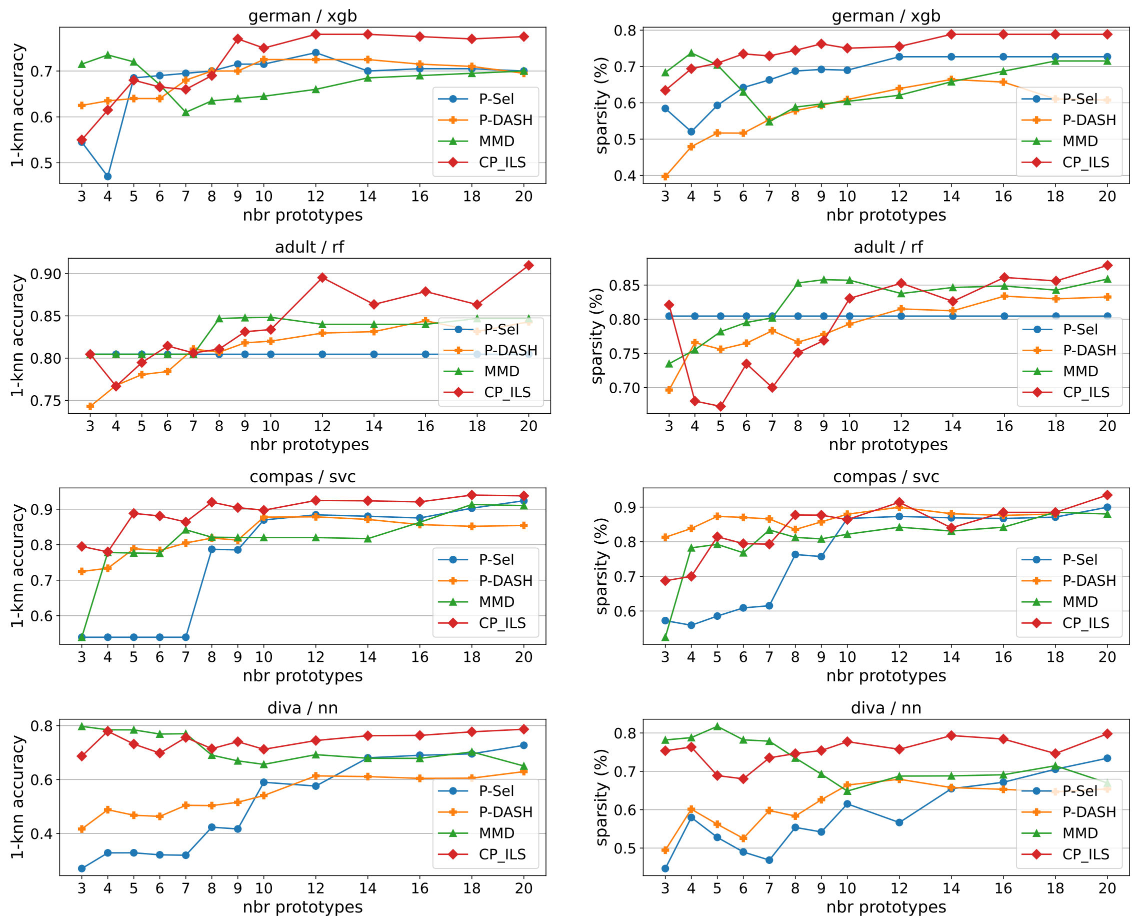


Figure 5.7: Evaluation of *1-KNN Accuracy* and *Sparsity* of prototype explanations while varying the number of prototypes on the explanation set. From top to bottom, we have `german`/XGB, `adult`/RF, `compas`/SVM, `diva`/NN.

and therefore discover more complex patterns. This results in greater difficulty to search for a set of prototypes in an unsupervised method such as MMD-critic. In Figure 5.7 are reported the behavior of the two metrics while changing the number of prototypes. As expected we observe, for both metrics, a low performance for all the methods for low prototype numbers. The scores increase to a plateau indicating that the maximum amount of information has been stored. Our approach outperforms the other methods when using a number of prototypes greater than 10. For a number less than 10, the results are fuzzy and no method is better than the others. We conclude that our approach is in line with the current state of the art when summarizing data with a low number of prototypes, but for higher numbers, it is able to discover peculiar groups of data that other approaches cannot.

5.2.2 Counterfactual Explanation Evaluation

In this section, we describe the results obtained for the counterfactual explanations produced by CP-ILS.

Competitors. We compare our proposal against different algorithms present in literature capable of producing counterfactual explanations. We selected the most popular methods that use different approaches to the problem with different focuses. We do not test any other methods [114, 51, 115] using latent space representations since they only work for image data. For GSG, we used the suggested initial ratio of 0.1 with a factor of increase of 10, and for every radius, we generated 2000 samples and checked for counterfactuals. For DiCE, we generated four counterfactuals. In the WACH algorithm, we started with $\lambda = 0.1$ and then increased by 0.1 if the algorithm failed to find a counterfactual. For CP-ILS we learned the best latent space dimension with a batch size of 1024, Adam optimizer [132] with a learning rate of 10^{-3}, and early stopping of 3. The Lagrange multiplier equation is resolved using the `scipy` library[8].

Metrics. We measure the *Proximity* of a counterfactual x' with its original sample x, its *Robustness*, *Plausibility*, and the *Diversity* of the counterfactuals found. We express all the evaluation metrics as distances, thus, for *Proximity*, *Robustness*, and *Plausibility* measures, the lower the values, the better the counterfactuals returned; for *Diversity* measures, the higher, the better. Some methods return more than one counterfactual; then we selected the best counterfactual among the set returned.

We measure proximity in two different fashions [54, 9]. The first one, named d_{dist}, is the average distance between x and the counterfactual x', the distance is

[8]https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve.html

computed using a mixed distance defined in Equation 5.2:

$$d_{dist} = \frac{1}{|X|} \sum_{x \in X} d_{mix}(x, x')$$

The second measure d_{count}, quantifies the average number of features changed between a counterfactual x' and x, i.e.,

$$d_{count} = \frac{1}{|X|m} \sum_{x \in X} \sum_{i=1}^{m} \mathbb{1}_{x'_i \neq x_i}$$

where $\mathbb{1}$ returns 1 if [*condition*] is true, 0 otherwise, and m is the number of features.

We measure *Plausibility* in terms of *Implausibility* [157] of the generated counterfactuals in terms of the distance of x' from the closest instance in the X. The lower, the better.

$$d_{impl} = \frac{1}{|X|} \sum_{x \in X} \min_{\tilde{x} \in X} d(x', \tilde{x})$$

We use the test set as reference population X. The objective is to have a counterfactual x' close to a reference population X but near samples labeled opposite to the original sample x. Besides, a counterfactual x' near instances of a wrong class will result in an adversarial sample.

With respect to *Robustness*, we define d_{adv} to measure how much the found counterfactual is similar in the class label to the reference population X. We measure the density of the classes of samples closer to x' in the reference population. First, the samples are ordered by distance (c), and then the neighborhood X is selected[9].

$$c = \text{sorted}[d(x', \tilde{x})] \quad d_{adv} = \frac{1}{|X|} \sum_{x \in X} \frac{1}{10} \sum_{i=1}^{10} \mathbb{1}_{c_i \neq \hat{x}'}$$

Finally, we measure the *Diversity* in two fashions. The first one denoted as div_{dist},

[9]We used ten samples as neighborhood estimation.

Table 5.4: Different counterfactuals found for a sample x of the `adult` dataset. For latent and DiCE we have four different counterfactuals while for GSG and WACH only one counterfactual is generated.

	Age	Hours Per Week	Education	Marital Status	Occupation	Gender	Native Country	Income
x	48.0	50.00	Prof School	Married	Prof Specialty	Male	US	0.754
CP-ILS	48.0	50.00	Prof School	Not Married	Prof Specialty	Male	US	0.350
CP-ILS	48.0	50.00	Prof School	Married	Blue Collar	Male	US	0.192
CP-ILS	40.0	50.00	Prof School	Not Married	Prof Specialty	Male	US	0.343
CP-ILS	42.0	37.10	Prof School	Not Married	Exec Managerial	Male	US	0.311
GSG	48.0	45.53	Dropout	Married	Prof Specialty	Male	US	0.186
DiCE	48.0	50.00	Community College	Married	Services	Male	US	0.318
DiCE	48.0	54.16	Dropout	Married	Prof Specialty	Male	US	0.187
DiCE	48.0	50.00	Dropout	Married	Prof Specialty	Male	US	0.186
DiCE	48.0	50.00	Community College	Married	Blue Collar	Male	US	0.197
WACH	41.0	38.09	Prof School	Not Married	Prof Specialty	Male	US	0.338

measures the average distance between the set of found counterfactuals E. In contrast, div_{count} measures the average number of different features.

$$div_{dist} = \frac{1}{|E|^2} \sum_{x' \in E} \sum_{x'' \in E} d(x', x'')$$

$$div_{count} = \frac{1}{|E|^2 m} \sum_{x' \in E} \sum_{x'' \in E} \sum_{i=1}^{m} \mathbb{1}_{x'_i \neq x''_i}$$

To complete the evaluation, we computed the Success Rate (SR) of the various algorithms when searching for a counterfactual as the percentage of the successful counterfactual found over the test set.

Qualitative Results. The first qualitative comparison in Table 5.4 shows the counterfactuals found by the four methods for an instance of the `adult` dataset and predicted using the SVM. The task of this dataset is to predict if the income of a person is greater than $50K$. The instance x under analysis is a person with a probability of an income bigger than $50K$ equal to 0.75%. This person is 48 years old married male and has done a professional school which helps prepare students

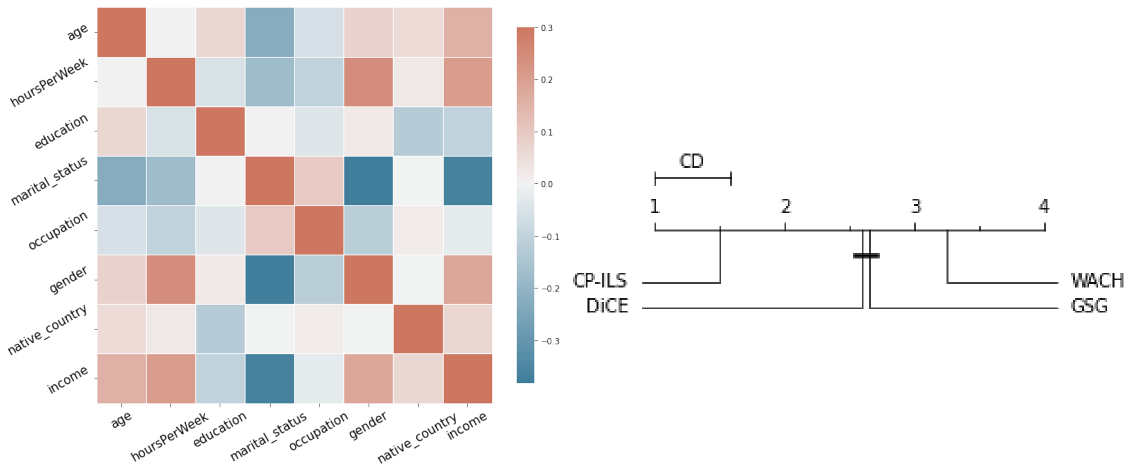


Figure 5.8: LEFT: Correlation matrix between the input features of the `adult` dataset and the label “income”. RIGHT: Critical difference plot for Nemenyi test with $\alpha = 0.05$.

for careers in specific fields. Examples include medical, law, pharmacy, business, library, and social work schools.

First, we notice that the four methods disagree on what features to change. The only unchanged features are gender and country of origin, which is a nice thing to see to remove any possible gender bias from the black-box. CP-ILS is the only approach that succeeds in identifying a single feature for which its change leads to a class change. Indeed by modifying the marital status to “Not Married” or the occupation to “Blue Collar”, the prediction decreases to 0.350 and 0.192, respectively. The term “Blue Collar” refers to jobs typically involving manual labor and compensation by an hourly wage. Thus, it is natural to think that the income decreases by moving from specialized work to a manual one. Less intuitive is the change in the marital status attribute. In this dataset, a “Married” person usually gains more than a “Not Married” one because he is usually older with a stable job. This can also be seen from the correlation matrix in Figure 5.8.

In the `adult` dataset, the features “Marital Status” and “Gender” strongly correlate with the label “Income”. Also, these two features have a strong correlation with each other. The SVM is making predictions based on one of the two and, in this case, choosing the “Marital Status” of a person.

The other counterfactual explainers found a correlation between “Occupation” and “Marital Status”. However, they also added other useless features as a modification. For example, besides “Occupation”, DiCE also changed “Education” to “Community College”, but the probability went to 0.197, the same as just changing the “Occupation”. Also, WACH changes the “Age”, “Hours Per Week”, and “Marital status”, resulting in the same income prediction as just modifying the last one. CP-ILS also lowered the “Age” in the third counterfactual. DiCE and GSG, while not founding the “Marital Status” change, found a different counterfactual modifying the “Hour Per Week” and “Education”, a feature never touched by our approach. However, they disagree on the changes to be made to the “Hour Per Week” feature, with GSG decreasing it while DiCE increasing it.

Moreover, the counterfactual found by GSG and DiCE is unrealistic. They both changed the “Education” to “Dropout” meaning that the person has left school, but they left untouched the “Occupation” feature. It is unrealistic that a person who left school now works in a specialized industry.

Quantitative Results. Table 5.5 and Table 5.6 report the means and standard deviation of the evaluation metrics for `adult` and `compas`, and `diva` and `german`, respectively. CP-ILS outperforms competitors in all datasets, particularly in the d_{count} metric. Indeed, as shown in the previous section, CP-ILS identifies which features to modify to change the black-box’s prediction better than the competitors. Moreover, the transparent latent space search helps the counterfactuals found to be more similar to the training set samples than other methods since the space is learned from the training set itself. In addition, generally, the *Diversity* of the counterfactuals is better than the one of DiCE, especially for the div_{count} metric. An interesting measure is the success rate of the creation of the counterfactuals. GSG and CP-ILS have the best success rate, close to 100%, while DiCE and WACH are the worst ones, in some cases lower than 30%.

Table 5.5: Results of the counterfactual metrics computed on the four datasets and on the four black-boxes plus the success rate (SR) of the algorithm for `adult` and `compas`. Best results are highlighted in bold, second best results are underlined, W column highlights the number of wins.

`adult`		d_{dist}	d_{count}	d_{impl}	d_{adv}	div_{dist}	div_{count}	SR	W
NN	CP-ILS	**0.604** ± 0.331	**1.681** ± 0.981	**0.069** ± 0.115	0.482 ± 0.261	**10.667** ± 8.090	3.406 ± 0.327	94%	4
	GSG	0.885 ± 0.323	4.480 ± 1.187	0.594 ± 0.229	0.640 ± 0.291	-	-	100%	0
	DiCE	1.686 ± 0.307	4.699 ± 1.084	0.114 ± 0.130	0.484 ± 0.312	1.724 ± 0.113	**4.431** ± 0.490	83%	1
	WACH	2.281 ± 1.370	7.364 ± 0.264	0.842 ± 0.538	**0.327** ± 0.322	-	-	87%	1
XGB	CP-ILS	0.385 ± 0.248	**1.550** ± 0.841	**0.022** ± 0.043	**0.366** ± 0.190	**2.054** ± 1.617	**2.996** ± 0.664	100%	5
	GSG	**0.290** ± 0.209	1.800 ± 0.748	0.161 ± 0.128	0.823 ± 0.235	-	-	100%	1
	DiCE	1.063 ± 0.457	1.697 ± 0.811	0.577 ± 0.481	0.720 ± 0.247	1.148 ± 0.256	1.978 ± 0.650	76%	0
	WACH	3.198 ± 1.190	7.000 ± 0.000	0.799 ± 0.403	0.558 ± 0.318	-	-	88%	0
RF	CP-ILS	0.492 ± 0.378	**1.917** ± 1.175	**0.039** ± 0.068	**0.264** ± 0.133	**1.897** ± 1.572	**2.837** ± 1.042	100%	5
	GSG	**0.392** ± 0.274	1.920 ± 1.026	0.230 ± 0.139	0.687 ± 0.282	-	-	100%	1
	DiCE	1.387 ± 0.564	2.447 ± 1.351	0.715 ± 0.522	0.534 ± 0.226	1.288 ± 0.285	2.314 ± 0.961	76%	0
	WACH	2.193 ± 0.976	7.000 ± 0.000	0.992 ± 0.638	0.414 ± 0.290	-	-	81%	0
SVM	CP-ILS	**0.244** ± 0.266	**1.105** ± 1.025	**0.239** ± 0.076	0.665 ± 0.209	**4.683** ± 0.631	**4.440** ± 0.883	100%	5
	GSG	0.835 ± 0.254	4.430 ± 1.283	0.673 ± 0.181	0.681 ± 0.291	-	-	100%	0
	DiCE	1.377 ± 0.199	2.105 ± 0.447	0.747 ± 0.521	0.691 ± 0.214	1.194 ± 0.170	1.809 ± 0.413	76%	0
	WACH	2.492 ± 1.319	7.000 ± 0.000	0.992 ± 0.638	**0.369** ± 0.354	-	-	88%	1
`compas`		d_{dist}	d_{count}	d_{impl}	d_{adv}	div_{dist}	div_{count}	SR	W
NN	CP-ILS	**0.510** ± 0.000	**1.000** ± 0.000	**0.225** ± 0.121	0.767 ± 0.146	**22.498** ± 9.317	6.623 ± 0.167	100%	4
	GSG	0.977 ± 0.364	9.300 ± 3.410	0.807 ± 0.285	0.587 ± 0.266	-	-	100%	0
	DiCE	1.227 ± 0.668	9.296 ± 1.629	0.718 ± 0.465	0.385 ± 0.233	-	-	27%	0
	WACH	3.576 ± 0.266	24.906 ± 1.916	2.546 ± 0.067	**0.275** ± 0.116	2.223 ± 0.477	**8.046** ± 1.138	53%	2
XGB	CP-ILS	0.400 ± 0.286	**1.230** ± 0.421	0.267 ± 0.131	0.417 ± 0.150	**8.148** ± 8.329	**6.647** ± 0.775	100%	3
	GSG	**0.328** ± 0.250	2.840 ± 1.454	0.392 ± 0.150	0.824 ± 0.228	-	-	100%	1
	DiCE	0.687 ± 0.486	1.333 ± 0.471	**0.225** ± 0.166	0.522 ± 0.253	1.114 ± 0.352	2.037 ± 0.605	27%	1
	WACH	3.492 ± 0.240	23.426 ± 1.717	2.484 ± 0.073	**0.150** ± 0.133	-	-	54%	1
RF	CP-ILS	**0.432** ± 0.372	1.800 ± 0.529	0.297 ± 0.141	0.393 ± 0.200	**7.609** ± 13.262	**6.596** ± 0.963	100%	3
	GSG	0.447 ± 0.330	2.760 ± 4.495	0.397 ± 0.170	0.711 ± 0.137	-	-	100%	0
	DiCE	0.713 ± 0.504	**1.556** ± 0.497	**0.206** ± 0.164	0.406 ± 0.208	0.976 ± 0.327	2.079 ± 0.585	27%	2
	WACH	3.619 ± 0.330	26.217 ± 4.495	2.560 ± 0.147	**0.117** ± 0.137	-	-	60%	1
SVM	CP-ILS	**0.690** ± 0.024	**1.210** ± 0.407	0.256 ± 0.138	0.702 ± 0.154	**6.149** ± 3.451	**5.198** ± 1.471	100%	4
	GSG	0.887 ± 0.344	7.130 ± 3.331	0.678 ± 0.244	0.488 ± 0.284	-	-	100%	0
	DiCE	0.749 ± 0.681	1.926 ± 0.716	**0.225** ± 0.181	0.391 ± 0.185	1.371 ± 0.438	2.551 ± 0.881	27%	1
	WACH	3.537 ± 0.232	23.922 ± 1.453	2.501 ± 0.058	**0.104** ± 0.063	-	-	51%	1

Table 5.6: Results of the counterfactual metrics computed on the four datasets and on the four black-boxes plus the success rate (SR) of the algorithm for `diva` and `german`. Best results are highlighted in bold, second best results are underlined, W column highlight the number of wins.

`diva`		d_{dist}	d_{count}	d_{impl}	d_{adv}	div_{dist}	div_{count}	SR	W
NN	CP-ILS	**0.415** ± 0.247	**5.980** ± 0.498	**0.435** ± 0.094	$\underline{0.481}$ ± 0.090	**1.650** ± 0.379	**20.596** ± 0.588	100%	5
	GSG	$\underline{0.432}$ ± 0.562	$\underline{37.800}$ ± 2.460	$\underline{0.573}$ ± 0.523	0.751 ± 0.175	-	-	100%	0
	DiCE	1.679 ± 0.350	84.381 ± 1.120	1.281 ± 0.327	0.585 ± 0.427	0.630 ± 0.197	48.406 ± 0.855	70%	0
	WACH	2.375 ± 0.684	87.364 ± 0.634	1.825 ± 0.439	**0.146** ± 0.026	-	-	63%	1
XGB	CP-ILS	$\underline{0.325}$ ± 0.284	**5.474** ± 3.903	**0.403** ± 0.113	$\underline{0.358}$ ± 0.272	0.291 ± 0.485	**44.257** ± 1.457	100%	3
	GSG	**0.290** ± 0.595	35.623 ± 2.681	$\underline{0.625}$ ± 0.453	0.769 ± 0.274	-	-	100%	1
	DiCE	0.682 ± 0.349	$\underline{9.578}$ ± 1.087	0.781 ± 0.504	0.608 ± 0.202	**1.186** ± 0.475	11.550 ± 1.237	78%	1
	WACH	2.756 ± 0.347	87.972 ± 0.009	1.864 ± 0.190	**0.236** ± 0.011	-	-	72%	1
RF	CP-ILS	$\underline{0.670}$ ± 0.470	$\underline{19.205}$ ± 7.206	**0.526** ± 0.395	$\underline{0.321}$ ± 0.177	0.889 ± 0.286	**78.994** ± 2.345	100%	2
	GSG	**0.396** ± 0.509	34.683 ± 2.070	$\underline{0.629}$ ± 0.461	0.704 ± 0.143	-	-	100%	1
	DiCE	0.960 ± 0.581	**9.834** ± 1.245	0.757 ± 0.513	0.455 ± 0.233	**1.291** ± 0.465	3.059 ± 1.337	76%	2
	WACH	2.593 ± 0.529	87.973 ± 0.010	1.902 ± 0.390	**0.142** ± 0.008	-	-	85%	1
SVM	CP-ILS	$\underline{0.403}$ ± 0.320	**5.906** ± 2.822	$\underline{0.516}$ ± 0.342	$\underline{0.570}$ ± 0.164	0.588 ± 0.216	**20.104** ± 1.989	100%	2
	GSG	**0.390** ± 0.555	37.431 ± 2.130	**0.471** ± 0.466	0.741 ± 0.184	-	-	100%	2
	DiCE	0.944 ± 0.450	$\underline{9.724}$ ± 0.974	0.757 ± 0.418	0.611 ± 0.299	**1.323** ± 0.466	3.125 ± 1.212	71%	1
	WACH	2.527 ± 0.434	87.017 ± 0.044	1.812 ± 0.398	**0.023** ± 0.082	-	-	77%	1
`german`		d_{dist}	d_{count}	d_{impl}	d_{adv}	div_{dist}	div_{count}	SR	W
NN	CP-ILS	**0.209** ± 0.000	**2.000** ± 0.000	**0.207** ± 0.064	$\underline{0.539}$ ± 0.252	**2.544** ± 1.175	4.958 ± 0.086	100%	4
	GSG	$\underline{0.448}$ ± 0.200	19.380 ± 4.125	0.499 ± 0.102	0.697 ± 0.276	-	-	100%	0
	DiCE	0.546 ± 0.456	$\underline{6.195}$ ± 2.192	$\underline{0.283}$ ± 0.089	0.627 ± 0.282	2.228 ± 0.255	**9.071** ± 1.598	77%	1
	WACH	1.324 ± 0.321	19.39 ± 24.71	0.460 ± 0.238	**0.191** ± 0.029	-	-	23%	1
XGB	CP-ILS	**0.115** ± 0.252	2.222 ± 0.462	**0.245** ± 0.108	$\underline{0.438}$ ± 0.154	**6.897** ± 6.099	**5.507** ± 0.879	100%	4
	GSG	$\underline{0.202}$ ± 0.175	**1.840** ± 0.784	0.298 ± 0.118	0.601 ± 0.211	-	-	100%	1
	DiCE	0.245 ± 0.365	$\underline{1.892}$ ± 1.420	$\underline{0.272}$ ± 0.176	0.676 ± 0.209	1.316 ± 0.399	2.758 ± 1.501	74%	0
	WACH	1.286 ± 0.270	14.713 ± 5.060	0.357 ± 0.033	**0.300** ± 0.000	-	-	27%	1
RF	CP-ILS	**0.455** ± 1.317	**2.130** ± 0.823	$\underline{0.310}$ ± 1.059	$\underline{0.479}$ ± 0.276	**6.337** ± 5.483	5.111 ± 1.334	81%	3
	GSG	0.687 ± 0.322	$\underline{3.750}$ ± 2.334	0.357 ± 0.144	0.577 ± 0.220	-	-	81%	0
	DiCE	$\underline{0.643}$ ± 0.659	4.676 ± 4.031	**0.292** ± 0.239	0.578 ± 0.176	1.792 ± 0.687	**5.552** ± 4.114	74%	2
	WACH	1.250 ± 0.227	12.525 ± 6.177	0.366 ± 0.035	**0.267** ± 0.047	-	-	29%	1
SVM	CP-ILS	**0.205** ± 0.011	**1.850** ± 0.357	**0.225** ± 0.079	$\underline{0.802}$ ± 0.194	**1.661** ± 0.966	**5.356** ± 0.539	100%	5
	GSG	$\underline{0.725}$ ± 0.334	17.210 ± 3.639	0.528 ± 0.146	0.851 ± 0.244	-	-	100%	0
	DiCE	0.707 ± 0.639	$\underline{2.635}$ ± 2.597	$\underline{0.323}$ ± 0.245	0.891 ± 0.213	1.466 ± 0.459	3.518 ± 2.063	74%	0
	WACH	1.137 ± 0.193	17.210 ± 8.143	0.365 ± 0.027	**0.038** ± 0.048	-	-	28%	1

A separate discussion must be made for d_{dist} and d_{adv}. In rare cases, a method is better in both metrics. It only occurs for `adult`, where CP-ILS produces counterfactuals that are both similar and non-adversarial. When producing a counterfactual is challenging to balance the distance with the original sample and the proximity with samples of different classes. What ends up happening is that a record closer to the original sample is more adversarial than one further away. It is not easy to judge a winner for these metrics, but CP-ILS is always first in one of the two metrics and a close second in the other. We report in Figure 5.8 the Critical Difference diagrams that highlight the statistical relevance of the results. The comparison of the ranks of all methods against each other is visually represented in Figure 5.8 with Critical Difference (CD) diagrams [158]. Two methods are tied if the null hypothesis that their performance is the same cannot be rejected using the Nemenyi test at α=0.05. CP-ILS has the average best rank considering all the measures, datasets, and black-box models with statistically significant performance.

CP-ILS Latent Dimension Impact. We further investigate the duality of being a close counterfactual without being adversarial by analyzing the variation of the metrics computed for CP-ILS while varying latent space dimensions. Results for `adult` with the SVM and for `compas` with XGB are presented in Figure 5.9 left and right respectively. Other datasets and black-boxes present similar behaviors, and we do not report them here due to lack of space. Also, increasing the number of dimensions of the latent space means adding more freedom to the creation of the latent representations. Hence, the information given by the input features is less compressed, and the information in the dataset is stored more efficiently. However, less compression results in a more meaningless latent space. It is as if the model is learning the dataset itself instead of the hidden information inside it. This process is very similar to overfitting, and it is captured by the d_{adv} metric. Indeed, after a particular dimension, this metric starts to increase until it reaches a maximum plateau. From Figure 5.9 is clear that for `adult` when using more than five latent

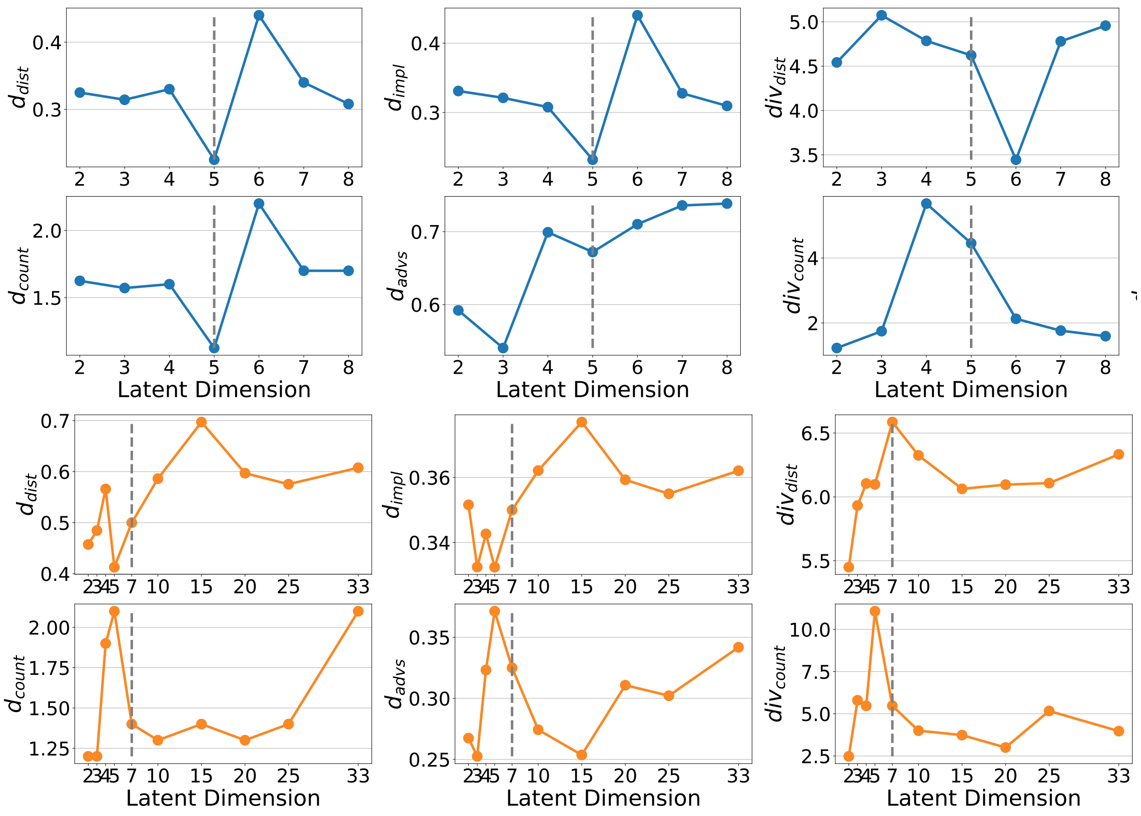


Figure 5.9: Counterfactual metrics in comparison with latent space dimension variation. On the top in blue we have the `adult` dataset with the SVM black-box. On the bottom in orange we have the `compas` dataset when using the XGB black-box. The dashed vertical lines represent the dimension used for the experiments.

dimensions the metric sharply increases. An interesting behavior occurs for d_{dist}. This metric increases after the fifth dimension, but we notice a significant drop also on higher dimensions. The best dimension for div_{dist} is three while for div_{count} is clearly four. Concerning d_{count} and d_{impl}, using five dimensions seems the best parameter to obtain a lower score on both these metrics. In conclusion, there is not a clear perfect dimension for which every metric is the best. Five seems the best overall dimension, as pointed out also with the *1-KNN Accuracy* metric in Figure 5.3. Regarding the `compas` dataset, there is not a clearly good dimension for every metric. CP-ILS has selected 7 as the best dimension using the overfitting procedure described in Section 5.1.1. This is the best dimension for d_{count} and

div_{dist}. However other metrics pointed out different dimensions: 6 for div_{count}, d_{dist}, and d_{impl}, while 15 for d_{advs}. The lower dimensions seem to produce closer and more plausible counterfactuals, which nevertheless look more like an adversarial attack rather than a true explanation.

Finally, CP-ILS has a success rate close to 100%, very similar to GSG. DiCE and WACH fail more easily. All methods have similar execution times with respect to the counterfactual search, which is in the order of minutes. However, CP-ILS must also construct the latent space representation which is time-consuming even if it only has to be done once.

5.2.3 Comparison with Rule-based Explainers

Counterfactual examples and counterfactual rules are two different types of explanations used in machine learning to generate explanations for decisions made by machine learning models (see Section 2.1.4). Counterfactual examples involve generating instances that are similar to the original input but lead to a different decision by the model. Counterfactual rules, on the other hand, involve generating rules that indicate changes to the input that would lead to a different decision by the model. However, we think it is useful to try to compare them to gain more insights into the effectiveness of our CP-ILS method.

In this section, we compared the explanation produced by CP-ILS with LORE: a very well-known method for producing counterfactual rules already presented in Section 2.1.4). Since counterfactual rules and examples are two different, but very similar types of explanations, we need a methodology to compare them. To do this, we took the rules generated by LORE and used the thresholds given in the rule as indications to produce a counterfactual example and then use the metrics already presented in the section before. For example, if the rule is: "the prediction would flip if feature A would be greater than 4.3", then we substitute the value of feature A, in the sample in analysis, with 4.3, and compute the metrics. For metrics, we computed d_{dist}, d_{count}, d_{impl}, and d_{advs}. The results are shown in Table 5.7.

The results of the comparison showed that CP-ILS is better than LORE, creating better explanations that are less adversarial (d_{advs}) and involve fewer changes to the original input features d_{impl}. However, in the other two metrics d_{impl} and d_{dist}, CP-ILS was found to be marginally better than LORE. Both CP-ILS and LORE were found to return very similar explanations to the original sample, with CP-ILS being slightly more robust than LORE.

In addition, we also provide in Figure 5.10 some qualitative two examples taken from the dataset `adult` and their corresponding explanations given by LORE and CP-ILS. In the first example on left, there is described a female worker with an income greater than 50k. The two methods were both able to identify the gender bias of the black-box, where the attribute "Gender" is mainly responsible for the wrong prediction. However, they disagree on the second feature to change, CP-ILS prefers to change only the attribute "Occupation" while LORE suggests increasing the attribute "Hours per Week". In this example, changing only the feature "Gender" highly decreases the probability output of the black-box model, but not enough to surpass the 0.5 threshold set for changing its prediction. Therefore another feature must be taken into account and CP-ILS as seen before is more inclined to select categorical features rather than continuous ones. On the right of Figure 5.10, both methods were able to identify the right feature to change.

Overall, the comparison showed that CP-ILS is a more effective model than LORE for generating explanations for tabular data that are less adversarial and involve fewer changes to the input features. However, both models are highly precise in their explanations and are efficient in generating them.

Table 5.7: Results of the counterfactual metrics computed on the four datasets and on the four black-boxes for both LORE and CP-ILS. The best results are highlighted in bold.

dataset	bb	method	d_{dist}	d_{count}	d_{impl}	d_{adv}
`adult`	NN	CP-ILS	**0.569** ± 0.327	**1.701** ± 0.979	**0.071** ± 0.133	**0.459** ± 0.279
		LORE	0.616 ± 0.350	2.728 ± 0.987	0.079 ± 0.122	0.730 ± 0.291
	XGB	CP-ILS	0.396 ± 0.234	**1.556** ± 0.830	0.040 ± 0.021	**0.399** ± 0.193
		LORE	**0.387** ± 0.279	1.799 ± 0.841	**0.023** ± 0.062	0.470 ± 0.214
	RF	CP-ILS	**0.499** ± 0.373	**1.870** ± 1.223	**0.008** ± 0.048	**0.273** ± 0.156
		LORE	0.532 ± 0.380	2.929 ± 1.198	0.083 ± 0.080	0.474 ± 0.172
	SVM	CP-ILS	0.280 ± 0.241	**1.123** ± 1.039	**0.193** ± 0.091	**0.653** ± 0.223
		LORE	**0.264** ± 0.311	2.142 ± 1.040	0.274 ± 0.078	0.895 ± 0.256
`compas`	NN	CP-ILS	**0.493** ± 0.025	**1.006** ± 0.028	**0.242** ± 0.109	**0.741** ± 0.138
		LORE	0.558 ± 0.039	2.932 ± 0.029	0.272 ± 0.162	0.902 ± 0.151
	XGB	CP-ILS	**0.402** ± 0.306	**1.190** ± 0.371	**0.248** ± 0.174	**0.401** ± 0.128
		LORE	0.402 ± 0.314	1.765 ± 0.463	0.272 ± 0.152	0.561 ± 0.173
	RF	CP-ILS	**0.435** ± 0.364	1.849 ± 0.549	**0.275** ± 0.187	0.437 ± 0.169
		LORE	0.445 ± 0.419	**1.814** ± 0.571	0.333 ± 0.151	**0.394** ± 0.206
	SVM	CP-ILS	0.731 ± 0.070	**1.221** ± 0.444	**0.232** ± 0.122	**0.711** ± 0.178
		LORE	**0.702** ± 0.059	1.555 ± 0.408	0.279 ± 0.167	0.733 ± 0.181
`german`	NN	CP-ILS	0.460 ± 0.208	**6.463** ± 1.465	**0.399** ± 0.121	**0.450** ± 0.056
		LORE	**0.416** ± 0.265	7.952 ± 1.520	0.435 ± 0.137	0.725 ± 0.116
	XGB	CP-ILS	**0.341** ± 0.291	**4.487** ± 3.930	**0.362** ± 0.089	**0.355** ± 0.225
		LORE	0.371 ± 0.326	4.518 ± 3.950	0.449 ± 0.162	0.670 ± 0.312
	RF	CP-ILS	0.706 ± 0.422	20.43 ± 3.543	0.559 ± 0.398	0.562 ± 0.193
		LORE	**0.680** ± 0.512	**19.20** ± 5.211	**0.529** ± 0.427	**0.326** ± 0.223
	SVM	CP-ILS	**0.396** ± 0.298	**3.873** ± 2.812	**0.536** ± 0.339	**0.573** ± 0.152
		LORE	0.421 ± 0.366	5.941 ± 2.849	0.554 ± 0.387	0.887 ± 0.211
`diva`	NN	CP-ILS	**0.167** ± 0.032	**2.033** ± 0.025	**0.198** ± 0.057	**0.559** ± 0.224
		LORE	0.211 ± 0.010	3.325 ± 0.037	0.220 ± 0.109	0.861 ± 0.266
	XGB	CP-ILS	**0.085** ± 0.216	2.252 ± 0.433	**0.211** ± 0.139	**0.457** ± 0.173
		LORE	0.155 ± 0.271	**2.240** ± 0.509	0.288 ± 0.109	0.562 ± 0.199
	RF	CP-ILS	**0.440** ± 1.366	**2.133** ± 0.869	0.341 ± 1.088	**0.446** ± 0.308
		LORE	0.477 ± 1.360	2.468 ± 0.846	**0.326** ± 1.107	0.523 ± 0.325
	SVM	CP-ILS	0.241 ± 0.004	**1.833** ± 0.403	**0.226** ± 0.039	**0.760** ± 0.180
		LORE	**0.238** ± 0.044	1.967 ± 0.373	0.236 ± 0.099	0.818 ± 0.239

$$\begin{array}{lll lll}
x_1 & = & \textit{Age} = \textit{32}, & x_2 & = & \textit{Age} = \textit{48}, \\
 & & \textit{HoursWeek} = \textit{32}, & & & \textit{HoursWeek} = \textit{50}, \\
 & & \textit{Education} = \textit{Bachelors}, & & & \textit{Education} = \textit{Prof-school}, \\
 & & \textit{MaritialStatus} = \textit{Married}, & & & \textit{MaritialStatus} = \textit{Married}, \\
 & & \textit{Occupation} = \textit{Prof-specialty}, & & & \textit{Occupation} = \textit{Prof-specialty}, \\
 & & \textit{Gender} = \textit{Female}, & & & \textit{Gender} = \textit{Male}, \\
 & & \textit{NativeCountry} = \textit{US} & & & \textit{NativeCountry} = \textit{US} \\
 & & & & & \\
 & & \textit{Income} \rightarrow > 50k & & & \textit{Income} \rightarrow > 50k \\
 & & bb_{pred} \rightarrow \leq 50k & & & bb_{pred} \rightarrow > 50k \\
 & & & & & \\
c_{lore} & = & \textit{Gender} = \textit{Female} & c_{lore} & = & \textit{MaritalStatus} = \textit{NotMarried} \\
 & & \textit{HoursWeek} \geq \textit{38} & & & bb_{pred} \rightarrow \leq 50k \\
 & & bb_{pred} \rightarrow > 50k & & & \\
 & & & & & \\
c_{cp\text{-}ils} & = & \textit{Gender} = \textit{Female} & c_{cp\text{-}ils} & = & \textit{MaritalStatus} = \textit{NotMarried} \\
 & & \textit{Occupation} = \textit{Exec-manager} & & & bb_{pred} \rightarrow \leq 50k \\
 & & bb_{pred} \rightarrow > 50k & & &
\end{array}$$

Figure 5.10: Two examples of explanations of LORE and CP-ILS from the `adult` dataset.

5.3 Discussion and Future Directions

In this chapter, we have presented CP-ILS, a prototype and counterfactual explanation method that exploits the predictions of the black box to obtain a better representation in the latent space. By examining prototypes and counterfactuals together, humans can better understand the strengths and limitations of the model, identify areas where it may be biased or misaligned with real-world data, and improve its performance through targeted adjustments or training. We have compared CP-ILS with several approaches that produce other types of latent space, showing that our proposal improves the state-of-the-art performance. In particular, introducing labels into the latent space helps to create better clusters and a prediction direction in the space that can be exploited to search for explanations. By following this direction is possible to find the most similar sample to the original one but with altered prediction. We evaluated the explanations returned using several metrics presented in Section 2.1.8. In addition, we tried to assess the robustness of the counterfactuals produced by measuring how closely they resemble an adversarial sample founding that our approach is one of the best. Indeed, using the prediction of the data in the creation of the latent space helps in reducing adversarial samples since data with similar predictions are forced to be mapped closer together. We have observed superior performance on several metrics assessing the quality of the latent space and the goodness of the counterfactual and prototype explanations obtained. CP-ILS is capable of resolving the problems of the previous chapter producing counterfactuals that are closer and more plausible. However, still, some problem remains, in particular how to choose the best latent space dimension. Our experiments show that different metrics perform differently as the latent space dimension change, which means that there is no perfect dimension for every metric. Our methodology selects the best among all, however, it is possible that a user is only interested in a small sample of these metrics. For example, he wants an explanation that is as close to the original as possible but it does not matter whether it is plausible or not. Or he may want an additional condition that we have not considered. We have tried to propose a methodology to select this parameter using a KNN method, however, this is an initial application and a more robust method needs to be explored.

Future research directions involve expanding the method to multi-class tabular datasets. The use of multiple labels would result in a latent space with multiple classification directions to follow, one for each label. Prototype explanation can be enhanced using natural language. The addition of natural language can help users better understand the key features and characteristics that are represented in the latent space. For example, natural language can be used to describe the prototypes and their relationship with the latent space, providing a more intuitive and accessible explanation of the underlying concepts. In addition, we would like to study the effects of our latent space when using image data. Indeed, from a particular point of view, image data are more complex than tabular data and require more complex mapping to encode. However, from some preliminary experiments, it seems complex to maintain the transparency required to apply Lagrange optimization. Therefore, it is necessary to study a new ad-hoc method for this type of data.

Chapter 6

Conclusions

Explainable AI (XAI) is a rapidly growing field that aims to make AI systems more transparent, accountable, and reliable. XAI techniques, such as feature importance explanations and counterfactual analysis, provide insight into how AI models make decisions and can help identify potential biases and errors. The importance of XAI has become increasingly evident in various areas where the consequences of incorrect decisions can be significant.

On the other hand, latent space models provide a compact and interpretable representation of the underlying data, making it possible to generate new examples, explore relationships between variables, and reduce dimensionality. One of the strengths of latent space models is their ability to capture complex relationships in a simple and interpretable way. For example, in generative models, latent space can be viewed as a compact representation of the underlying data distribution from which new examples can be generated. Latent space models can be used to discover patterns and relationships in data, making it possible to perform various tasks.

In this thesis, we have shown how it is possible to combine these two areas of research to produce systems that are not only effective but also explainable and reliable. The combination of XAI and latent space generative models have the potential to provide an even more powerful and interpretable tool for various applications.

Can we use the latent space to explore the decision boundary of the black-box? Can XAI methods help enhance such exploration?

In chapter 3 we demonstrated how XAI techniques can be used to provide explanations of how a generative model arrived at a particular sample, helping us to understand relationships in latent space and identify black-box decisions. In particular by allowing users to interact with the model and manipulate the input data, users can gain a deeper understanding of the factors that influence the model's behavior and the decision boundary. One way to use interaction is to visualize the decision boundary in the latent space and explore it through interactive visualizations. Sliders allow users to see how the model responds to changes in the input data and to observe how the decision boundary changes as different features are manipulated. XAI methods act as a guide for the user to move in the latent space and better understand model behavior. Interpretation and exploration of latent space are critical aspects of understanding the relationships among variables and understanding how black-box decision-making occurs. Exploration of latent space allows one to understand the relationships between variables and identify patterns and structures that may not be evident in the original data.

Is it possible to construct a latent space using interpretable models and still retain its proprieties? Can this new interpretable latent space be used to train ML models and produce explanations?

However, the relationships between features must be discovered a posteriori using an interaction framework because the latent space was created using a black-box machine learning model. Therefore, in Chapter 4, we created the latent space using an interpretable model by design, designed to provide insight into the underlying relationships between variables and decision-making. Specifically, we used a linear model in combination with a similarity loss to create an interpretable latent space that provides a better understanding of the relationships between the variables and the decision-making process, which can lead to improved performance and transparency of the black-box trained on it. Constructing a latent space using a linear model can provide several advantages in terms of producing better explanations for machine learning models. Linear models are simpler and more interpretable than more complex models, allowing a more compact representation of the data. Because linear models are often more interpretable than more complex models, it is easier to understand and explain the relationships between the input features and the output. The coefficients of a linear model have a real meaning in the latent space, and it is possible to identify which features are most important in determining the model's behavior. Our model called ILS can provide not only a better space in which to perform classification but also provide human-understandable counterfactual explanations of the patterns learned in it.

> ***Using the latent space as a tool for producing post-hoc explanations*** *(chapter 5). Can the latent space proprieties help us in producing better explanations than in the input space? What type of post-hoc explanations can be obtained? How do they compare with other approaches in the literature?*

However, the counterfactual explanations produced by ILS lack robustness and plausibility. The black-box is trained only after the latent space is created, and that space does not reflect or account for the predictions of the black-box. Inspired by the interpretable properties of the latent space, we added the black-box predictions directly into the creation of the latent space. This additional constraint, however, can only be applied if the black-box is trained before, turning our method into a post-hoc method. We provide our explanation as a Prototype/Counterfactual pair in order to enhance the understanding of the black-box internal reasoning. The core of CP-ILS approach is the use of the label in the creation of the latent space. By adding prediction in the creation of latent space, our CP-ILS method, presented in

Chapter 5, is able to produce better explanations. By moving in this direction in latent space, it is possible to find similar data but with altered predictions. Therefore, CP-ILS is able to find the correct changes in the input space to move along this direction and then observe how the prediction changes. Another advantage of using a prediction direction in latent space is that it allows finer control over counterfactual explanations. For example, the magnitude of the change to be made can be specified, allowing finer control over the effect of the change on the prediction. The use of balck-box predictions also allows CP-ILS to return to better separate data and return to the user local prototypes, making it easier to understand the abstract concepts of the latent space. Prototype explanation is an intentional group representation and in combination with natural language, can help users better understand latent space explanations. CP-ILS also enables a first approach to the actionability of the counterfactual explanations. The user can select which features to modify and CP-ILS is able to find the best combination that alters the prediction. This type of interaction can be particularly useful in situations where the user wants to understand why the model made a certain prediction, and what factors were most influential in that decision. By exploring counterfactual scenarios, the user can gain a deeper understanding of the underlying relationships between the input data and the model's output and can identify which features or parameters are most important in determining the model's behavior.

In all of the chapters of this thesis, we worked with tabular data. Our work is extendable to every possible data type but with some trade-offs. The use of a linear transformation in latent space models allowed us to obtain more precise and interpretable explanations; however, the latent space created is simple, and more complex data might be difficult to represent in this way. This is a trade-off between interpretability and accuracy. On one hand, non-linear transformations can capture more complex relationships between the data, leading to more accurate and complex latent spaces. On the other hand, these models can be more difficult to interpret and explain, as the underlying relationships may be more abstract.

In conclusion, XAI is a critical area of research that has great potential to improve the transparency and accountability of AI systems. Further research in XAI will be critical to build a future in which humans and AI can work together reliably and ethically and to ensure that AI is used for the benefit of society as a whole. However, there are still many open challenges in XAI, such as ensuring that XAI explanations are robust and do not mislead users, as well as protecting privacy and avoiding unwanted disclosures. Latent space modeling has great potential for building more powerful and interpretable AI systems. Further research in this area will be critical to explore new applications, improve model performance, and make latent space models more robust and accessible to non-experts.

Bibliography

[1] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28, 2015.

[2] Naveed Akhtar et al. Threat of adversarial attacks on deep learning in computer vision: Survey II. *CoRR*, abs/2108.00401, 2021.

[3] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[4] Alexandra Chouldechova. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big data*, 5(2):153–163, 2017.

[5] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[6] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[7] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

[8] Andreas Theissler, Francesco Spinnato, Udo Schlegel, and Riccardo Guidotti. Explainable AI for time series classification: A review, taxonomy and research directions. *IEEE Access*, 10:100700–100724, 2022.

[9] Riccardo Guidotti. Counterfactual explanations and how to find them: literature review and benchmarking. *Data Mining and Knowledge Discovery*, pages 1–55, 2022.

[10] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[11] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[12] Christoph Molnar. *Interpretable Machine Learning*. Lulu. com, 2020.

[13] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019.

[14] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, pages 9505–9515, 2018.

[15] Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[16] Frank Pasquale. *The black box society: The secret algorithms that control money and information*. Harvard University Press, 2015.

[17] Andrey Kurenkov. Lessons from the pulse model and discussion. the gradient, 2020.

[18] Bryce Goodman and Seth Flaxman. Eu regulations on algorithmic decision-making and a "right to explanation". In *ICML workshop on human interpretability in machine learning (WHI 2016), New York, NY. http://arxiv.org/abs/1606.08813 v1*, 2016.

[19] Sandra Wachter, Brent Mittelstadt, and Luciano Floridi. Why a right to explanation of automated decision-making does not exist in the general data protection regulation. *International Data Privacy Law*, 7(2):76–99, 2017.

[20] Giovanni Comandè. Regulating algorithms' regulation? first ethico-legal principles, problems, and opportunities of algorithms. In *Transparent Data Mining for Big and Small Data*, pages 169–206. Springer, 2017.

[21] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[22] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller. *Explainable AI: interpreting, explaining and visualizing deep learning*, volume 11700. Springer Nature, 2019.

[23] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

[24] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " 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*, pages 1135–1144, 2016.

[25] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.

[26] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

[27] Gabriel Erion, Joseph D Janizek, Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Learning explainable models using attribution priors. *arXiv preprint arXiv:1906.10670*, 2019.

[28] Deng Pan, Xin Li, and Dongxiao Zhu. Explaining deep neural network models with adversarial gradient integration. In *Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[29] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[30] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*, 2017.

[31] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[32] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 839–847. IEEE, 2018.

[33] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

[34] Andrei Kapishnikov, Tolga Bolukbasi, Fernanda Viégas, and Michael Terry. Xrai: Better attributions through regions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4948–4957, 2019.

[35] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[36] Riccardo Guidotti, Anna Monreale, Fosca Giannotti, Dino Pedreschi, Salvatore Ruggieri, and Franco Turini. Factual and counterfactual explanations for black box decision making. *IEEE Intell. Syst.*, 34(6):14–23, 2019.

[37] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI*, volume 18, pages 1527–1535, 2018.

[38] Yao Ming, Huamin Qu, and Enrico Bertini. Rulematrix: Visualizing and understanding classifiers with rules. *IEEE transactions on visualization and computer graphics*, 25(1):342–352, 2018.

[39] Mark Craven and Jude W Shavlik. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems*, pages 24–30, 1996.

[40] Eleanor H Rosch. Natural categories. *Cognitive psychology*, 4(3):328–350, 1973.

[41] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[42] Karthik S Gurumoorthy, Amit Dhurandhar, Guillermo Cecchi, and Charu Aggarwal. Efficient data representation by selecting prototypes with importance weights. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 260–269. IEEE, 2019.

[43] Alberto Blanco-Justicia, Josep Domingo-Ferrer, Sergio Martínez, and David Sánchez. Machine learning explainability via microaggregation and shallow decision trees. *Knowledge-Based Systems*, 2020.

[44] Peter Hase, Chaofan Chen, Oscar Li, and Cynthia Rudin. Interpretable image recognition with hierarchical prototypes. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 7, pages 32–40, 2019.

[45] Jacob Bien and Robert Tibshirani. Prototype selection for interpretable classification. *The Annals of Applied Statistics*, pages 2403–2424, 2011.

[46] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: deep learning for interpretable image recognition. In *Advances in neural information processing systems*, pages 8930–8941, 2019.

[47] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730*, 2017.

[48] Ruth MJ Byrne. Counterfactual thinking: From logic to morality. *Current Directions in Psychological Science*, 26(4):314–322, 2017.

[49] Umang Bhatt et al. Explainable machine learning in deployment. In *FAT**, pages 648–657. ACM, 2020.

[50] Riccardo Guidotti et al. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5):93:1–93:42, 2019.

[51] Riccardo Guidotti, Anna Monreale, Stan Matwin, and Dino Pedreschi. Black box explanation by learning image exemplars in the latent feature space. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 189–205. Springer, 2019.

[52] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 2017.

[53] Thibault Laugel et al. Comparison-based inverse classification for interpretability in machine learning. In *IPMU (1)*, CCIS, pages 100–111. Springer, 2018.

[54] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 2020.

[55] Alex Kulesza, Ben Taskar, et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

[56] Ann L McGill and Jill G Klein. Contrastive and counterfactual reasoning in causal judgment. *Journal of Personality and Social Psychology*, 64(6):897, 1993.

[57] Ilia Stepin, Jose M Alonso, Alejandro Catala, and Martín Pereira-Fariña. A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence. *IEEE Access*, 9:11974–12001, 2021.

[58] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. In *Advances in Neural Information Processing Systems*, 2018.

[59] Riccardo Guidotti et al. Factual and counterfactual explanations for black box decision making. *IEEE Intell. Syst.*, 34(6):14–23, 2019.

[60] Shubham Sharma et al. CERTIFAI: counterfactual explanations for robustness, transparency, interpretability, and fairness of artificial intelligence models. *CoRR*, abs/1905.07857, 2019.

[61] Shubham Rathi. Generating counterfactual and contrastive explanations using SHAP. *CoRR*, abs/1906.09293, 2019.

[62] Adam White et al. Measurable counterfactual local explanations for any classifier. In *ECAI*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2529–2535. IOS Press, 2020.

[63] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

[64] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[65] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1675–1684, 2016.

[66] Trevor J Hastie and Robert J Tibshirani. *Generalized additive models*. Routledge, 2017.

[67] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631, 2013.

[68] Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. An interpretable model with globally consistent explanations for credit risk. *arXiv preprint arXiv:1811.12615*, 2018.

[69] Mengjiao Yang and Been Kim. Bim: Towards quantitative evaluation of interpretability methods with ground truth. *arXiv preprint arXiv:1907.09701*, 2019.

[70] John D Lee and Katrina A See. Trust in automation: Designing for appropriate reliance. *Human factors*, 46(1):50–80, 2004.

[71] Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z. Gajos, Walter S. Lasecki, and Neil Heffernan. Axis: Generating explanations at scale with learnersourcing and machine learning. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale*, L@S '16, 2016.

[72] Adi Suissa-Peleg, Daniel Haehn, Seymour Knowles-Barley, Verena Kaynig, Thouis R Jones, Alyssa Wilson, Richard Schalek, Jeffery W Lichtman, and Hanspeter Pfister. Automatic neural reconstruction from petavoxel of electron microscopy data. *Microscopy and Microanalysis*, 2016.

[73] Been Kim, Caleb M Chacha, and Julie A Shah. Inferring team task plans from human meetings: A generative modeling approach with logic-based prior. *Journal of Artificial Intelligence Research*, 2015.

[74] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. *Advances in Neural Information Processing Systems*, 31:7775–7784, 2018.

[75] Sahil Verma, John Dickerson, and Keegan Hines. Counterfactual explanations for machine learning: A review. *arXiv preprint arXiv:2010.10596*, 2020.

[76] Stratis Tsirtsis and Manuel Gomez Rodriguez. Decisions, counterfactual explanations and strategic behavior. *Advances in Neural Information Processing Systems*, 33:16749–16760, 2020.

[77] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3681–3688, 2019.

[78] Chih-Kuan Yeh et al. On the (in) fidelity and sensitivity of explanations. *Advances in Neural Information Processing Systems*, 32, 2019.

[79] David Alvarez-Melis and Tommi S Jaakkola. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049*, 2018.

[80] Przemyslaw Biecek and Tomasz Burzykowski. *Explanatory Model Analysis*. Chapman and Hall/CRC, New York, 2021.

[81] Francesco Bodria, Fosca Giannotti, Riccardo Guidotti, Francesca Naretto, Dino Pedreschi, and Salvatore Rinzivillo. Benchmarking and survey of explanation methods for black box models, 2021.

[82] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[83] Andrea Vedaldi and Stefano Soatto. Quick shift and kernel methods for mode seeking. In *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part IV*, volume 5305 of *Lecture Notes in Computer Science*, pages 705–718. Springer, 2008.

[84] Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[85] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[86] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1):55–63, 1968.

[87] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[88] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[89] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *Iclr*, 2(5):6, 2017.

[90] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[91] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[92] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[93] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[94] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[95] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[96] David Berthelot, Thomas Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[97] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[98] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[99] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems*, pages 3308–3318, 2017.

[100] Mansheng Chen, Ling Huang, Chang-Dong Wang, and Dong Huang. Multi-view clustering in latent embedding space. In *AAAI*, pages 3513–3520, 2020.

[101] Ming Yin, Weitian Huang, and Junbin Gao. Shared generative latent representation learning for multi-view clustering. In *AAAI*, pages 6688–6695, 2020.

[102] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. *arXiv preprint arXiv:1711.00848*, 2017.

[103] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[104] Tom White. Sampling generative networks. *arXiv preprint arXiv:1609.04468*, 2016.

[105] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

[106] Erik Bernhardsson. Analyzing 50k fonts using deep neural networks. *URL https://erikbern. com/2016/01/21/analyzing-50k-fontsusing-deep-neural-networks. html*, 2016.

[107] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[108] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning*, pages 1558–1566. PMLR, 2016.

[109] Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3619–3629, 2021.

[110] Benjamin Shickel and Parisa Rashidi. Sequential interpretability: methods, applications, and future direction for understanding deep learning models in the context of sequential data. *arXiv preprint arXiv:2004.12524*, 2020.

[111] Wolfgang Stammer, Marius Memmel, Patrick Schramowski, and Kristian Kersting. Interactive disentanglement: Learning concepts by interacting with their prototype representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10317–10328, 2022.

[112] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European conference on computer vision*, pages 597–613. Springer, 2016.

[113] Arnaud Van Looveren et al. Interpretable counterfactual explanations guided by prototypes. In *ECML/PKDD (2)*, volume 12976 of *Lecture Notes in Computer Science*, pages 650–665. Springer, 2021.

[114] Rachana Balasubramanian et al. Latent-cf: A simple baseline for reverse counterfactual explanations. *CoRR*, abs/2012.09301, 2020.

[115] Pau Rodríguez et al. Beyond trivial counterfactual explanations with diverse valuable explanations. In *ICCV*, pages 1036–1045. IEEE, 2021.

[116] Michael Downs, Jonathan L Chu, Yaniv Yacoby, Finale Doshi-Velez, and Weiwei Pan. Cruds: Counterfactual recourse using disentangled subspaces. *ICML WHI*, 2020:1–23, 2020.

[117] Shalmali Joshi, Oluwasanmi Koyejo, Been Kim, and Joydeep Ghosh. xgems: Generating examplars to explain black-box models. *arXiv preprint arXiv:1806.08867*, 2018.

[118] Pouya Samangouei, Ardavan Saeedi, Liam Nakagawa, and Nathan Silberman. Explaingan: Model explanation via decision boundary crossing transformations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 666–681, 2018.

[119] Wesam Barbakh and Colin Fyfe. Online clustering algorithms. *International Journal of Neural Systems*, 18(03):185–194, 2008.

[120] Akiva Kleinerman, Ariel Rosenfeld, David Benrimoh, Robert Fratila, Caitrin Armstrong, Joseph Mehltretter, Eliyahu Shneider, Amit Yaniv-Rosenfeld, Jordan Karp, Charles F Reynolds, et al. Treatment selection using prototyping in latent-space with application to depression treatment. *PloS one*, 16(11):e0258400, 2021.

[121] Jie Liu, Kechen Song, Mingzheng Feng, Yunhui Yan, Zhibiao Tu, and Liu Zhu. Semi-supervised anomaly detection with dual prototypes autoencoder for industrial surface inspection. *Optics and Lasers in Engineering*, 136:106324, 2021.

[122] Bo Pang, Tian Han, Erik Nijkamp, Song-Chun Zhu, and Ying Nian Wu. Learning latent space energy-based prior model. *Advances in Neural Information Processing Systems*, 33:21994–22008, 2020.

[123] Davide Abati, Angelo Porrello, Simone Calderara, and Rita Cucchiara. Latent space autoregression for novelty detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[124] Chi Su, Shiliang Zhang, Fan Yang, Guangxiao Zhang, Qi Tian, Wen Gao, and Larry S Davis. Attributes driven tracklet-to-tracklet person re-identification using latent prototypes space mapping. *Pattern Recognition*, 66:4–15, 2017.

[125] Kelvin Guu, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450, 2018.

[126] Alison Smith-Renner, Ron Fan, Melissa Birchfield, Tongshuang Wu, Jordan Boyd-Graber, Daniel S Weld, and Leah Findlater. No explainability without accountability: An empirical study of explanations and feedback in interactive ml. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[127] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[128] Stefano Teso and Kristian Kersting. Explanatory interactive machine learning. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 239–245, 2019.

[129] Hubert Baniecki, Wojciech Kretowicz, Piotr Piatyszek, Jakub Wisniewski, and Przemyslaw Biecek. dalex: Responsible Machine Learning with Interactive Explainability and Fairness in Python. *arXiv:2012.14406*, 2020.

[130] Yunchen Pu et al. Variational autoencoder for deep learning of images, labels and captions. In *NIPS*, pages 2352–2360, 2016.

[131] Laurens Van der Maaten et al. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[132] Diederik P. Kingma et al. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

[133] Solomon Kullback et al. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[134] Yingfan Wang et al. Understanding how dimension reduction tools work: An empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. *J. Mach. Learn. Res.*, 22:201:1–201:73, 2021.

[135] Leland McInnes and John Healy. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR*, abs/1802.03426, 2018.

[136] Ehsan Amid and Manfred K. Warmuth. Trimap: Large-scale dimensionality reduction using triplets. *CoRR*, abs/1910.00204, 2019.

[137] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *SIGMOD Conference*. ACM, 2000.

[138] Pang-Ning Tan, Michael S. Steinbach, Anuj Karpatne, and Vipin Kumar. *Introduction to Data Mining (Second Edition)*. Pearson, 2019.

[139] André Artelt and Barbara Hammer. On the computation of counterfactual explanations–a survey. *arXiv preprint arXiv:1911.07749*, 2019.

[140] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23(1):18, 2020.

[141] George Baryannis, Samir Dani, and Grigoris Antoniou. Predicting supply chain risks using machine learning: The trade-off between performance and interpretability. *Future Generation Computer Systems*, 101:993–1004, 2019.

[142] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Computing*, 13(10):959–977, 2009.

[143] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[144] Martin Wattenberg et al. How to use t-sne effectively. *Distill*, 1(10):e2, 2016.

[145] Mariana Curi et al. Interpretable variational autoencoders for cognitive models. In *IJCNN*, pages 1–8. IEEE, 2019.

[146] Przemysław Biecek. Dalex: explainers for complex predictive models in r. *The Journal of Machine Learning Research*, 19(1):3245–3249, 2018.

[147] Bill Waggener et al. *Pulse code modulation techniques*. Springer Science & Business Media, 1995.

[148] Nasir Ahmed et al. Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93, 1974.

[149] Kien Mai Ngoc et al. Finding the best k for the dimension of the latent space in autoencoders. In *International Conference on Computational Collective Intelligence*, pages 453–464. Springer, 2020.

[150] Evelyn Fix et al. Discriminatory analysis. nonparametric discrimination: Consistency properties. *ISR*, 57(3):238–247, 1989.

[151] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[152] R Tyrrell Rockafellar. Lagrange multipliers and optimality. *SIAM review*, 35(2):183–238, 1993.

[153] Tianqi Chen et al. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794. ACM, 2016.

[154] Leo Breiman. Random forests. *Machine learning*, pages 5–32, 2001.

[155] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[156] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[157] David Alvarez-Melis et al. Towards robust interpretability with self-explaining neural networks. In *NeurIPS*, pages 7786–7795, 2018.

[158] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.