UNIVERSITÀ DI PISA

DEPARTMENT OF INFORMATION ENGINEERING

MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE AND DATA ENGINEERING

MASTER'S THESIS

Design and Evaluation of an Ensemble of Latent Spaces to Generate Counterfactuals for Explainable Multiclass Emotion Recognition

Candidate:
Davide Vigna

First Relator:
Ing. Antonio L. Alfeo

Other Relators:
Prof. Mario G.C.A. Cimino

Table of Contents

Abstract ... **3**

1 The need of Explainability ... **4**

2 Related works ... **12**

2.1 Dimensionality Reduction ... 12

2.2 Counterfactuals explanations methods ... 13

3 Design and implementation ... **17**

3.1 CPILS ... 17

3.1.1 The generation of the latent space ... 17

3.1.2 Counterfactuals generation ... 21

3.2 CPILS for multi-class problems ... 24

3.2.1 Generation of the latent spaces: multiclass scenario ... 25

3.2.2 Counterfactuals generation: multiclass scenario ... 26

3.2.3 Different modalities ... 27

3.2.4 Counterfactual Classification and Post-Processing ... 30

4 Case study ... **32**

4.1 Benchmark datasets ... 33

4.2 K-EmoCon datasets ... 36

4.3 Pre-processing activities ... 41

4.4 Black box models ... 42

4.5 Feature Actionability ... 44

5 Experimental Results ... **45**

5.1 Experimental Settings ... 45

5.2 Evaluation Metrics for multi-class CPILS ... 47

5.3 Evaluation Metrics for state-of-the-art comparison ... 52

5.4 Experimentation Reports ... 55

5.4.1 Report 1: Multi-Class CPILS - different working modalities ... 55

5.4.2 Report 2: Multi-Class CPILS - Benchmark Datasets ... 60

5.4.3 Report 3: Multi-Class CPILS - KEmoCon Datasets ... 67

5.4.4 Report 4: Multi-Class CPILS - impact of latent space dimension .. 73
5.4.5 Report 5: Multi-Class CPILS - comparison with DiCE 77
5.4.6 Report 6: Multi-Class CPILS - multi latent dimensions vs DiCE . 83

6 Conclusions and Future Works **86**

A Appendix A: User Manual and Software Implementation **89**

A.1 User Manual 90
Quick Start 90
Initialization 91
Latent Space Generation 92
Plotting Functionalities 93
Counterfactuals Generation 95
Ranking Counterfactuals 97

A.2 Source Code 99

B Appendix B: Tables and Additional Details **113**

B.1 Hardware Architecture 113

B.2 Black-box testing accuracy and Normalization effect 114

Bibliography **120**

Abstract

This thesis presents a post-hoc XAI-based method for generating local counterfactual explanations for any multi-class classifiers for tabular datasets (model-agnostic). The proposed approach exploits lower-dimensional latent spaces to produce class-specific counterfactuals. To assess its effectiveness, the method is evaluated on both benchmark datasets and a real-world dataset (i.e., physiological data for emotion recognition). A complete analysis is conducted to explain how a multi-class black box model can be explained through an approach that leverages ensembles of binary linear latent spaces to generate counterfactuals. In each specific latent space, the research of counterfactuals is guided by a prediction direction internally encoded maintaining the property of interpretability. It has enabled the discovery of more representative counterfactuals that are less sensitive to perturbations and situated in denser regions. The nature of this method has led to distinguishing and introducing some new definitions in the context of counterfactuals, differentiating specific counterfactuals from acceptable (more generic). They both aim to explain specific input instances but with different meanings. The way in which the combination of classes can be considered to transform the multi-class scenario into several binary ones has opened the possibility to conduct various types of searches introducing different working modalities. This work focuses explicitly on the modalities One-v-One and One-v-Rest as they result in fewer discarded counterfactuals, reduced computational time, and closer counterfactuals with respect to other proposed modalities. To consider as many different prospects as possible, several evaluation metrics have been employed to cover concepts of generation capability, specificity, minimality, acceptability, diversity, robustness, anomaly score, and execution time. The contribution of a different latent space dimensionality is also considered and have led to better results in some cases. A detailed comparison is conducted with (DiCE), one of the most famous state-of-the-art XAI algorithms, highlighting important results in terms of robustness especially on Benchmark datasets but also on the specific medical (physiological data). Regarding other metrics, the proposed method strikes a balance between DiCE's random and genetic implementations.

1 The need of Explainability

In recent years, Artificial Intelligence has made extraordinary progress, becoming an integral part of many aspects of society. Its exponential growth is driven by two key factors: the rapid advancement of computational power, maximized through parallel and distributed computing, and the widespread proliferation of Big Data. These two elements have made possible the practical realization of some of the algorithms and studies that in the past years, were designed just theoretically. They have evolved over time into sophisticated systems capable of recognizing complex and intricate patterns through advanced architectures, often surpassing human capabilities. As a result, they have been increasingly engaged in autonomous and critical decision-making processes, making their performances and accuracies top priorities. However, the race to achieve important confidence scores has impacted the understandability and interpretability of these systems making their behaviour more difficult to understand for those who use them. The main difference between a traditional computer program and an AI system lies in their creation process. The first is coded by a human who defines the rules and logic step by step, making it readable, understandable, and easily modifiable. In contrast, an AI system is driven by data - it is trained using machine learning or deep learning techniques, which results in limited transparency and requires a certain level of trust in its decision-making. This lack of transparency allows AI systems to potentially perpetuate or even amplify existing biases and injustices, as they can inadvertently learn flawed patterns from the data. If the training dataset contains biased decision records, misleading classifications, or artifacts from data collection errors, the resulting AI model is likely to inherit these biases, leading to discriminatory or inaccurate recommendations. For these reasons, there is a growing and urgent demand for greater explainability in these systems. As suggested in [1]:

Given an audience, an ***eXplainable Artificial Intelligence (XAI)*** *is one that produces details or reasons to make its functioning clear or easy to understand.*

This definition emphasizes the importance of the audience, the individuals who interact and use AI systems. This is an important aspect because depending on the purposes and skills of the utilizer, the explainability intended as the clarification process of the system's internal functioning could be different from subject to subject. E.g. A machine learning engineer may focus on understanding the low-level details of a complex predictive system whereas an executive manager may be more concerned about how the model outputs and how it guides him in important decisions. As a result, different forms of explainability must be considered to meet the needs of diverse users. In literature, the term Explainability is often used interchangeably with other synonyms such as understandability, comprehensibility, interpretability, and transparency but they have

deep different meaning. A model is considered *understandable* if it is able to make a human understand its functions without the need to explain its internal structure. This can be thought of as a passive characteristic, where no additional intervention is needed to enhance the user's comprehension. A model is considered *transparent* if it is by itself, it is understandable. A model is considered *comprehensible* if it can represent its internal knowledge in a human-understandable fashion such as symbolic description. A model is *interpretable* if it has the ability to provide the meaning in understandable terms. Interpretability is an active property of a model, enabling it to explain its knowledge in a way that is accessible to humans. More generally, explainability can be seen as the interface between humans and decision maker.

On the other side, the term *black box* has emerged in the field of XAI to describe machine learning models that function as opaque systems, where their internal workings are neither easily accessible nor interpretable [2]. In contrast, the term white box is used for those systems characterized by transparent property. Examples of black box models include neural networks, random forests, or extreme gradient boosting. These techniques implement advanced mechanisms that are hardly complex but, in some cases, can recognize more intricate patterns. On the other hand, methods such as simple decision trees, k-nearest neighbor, and rule-based learning are considered the most transparent ones, because they require a lower cognitive effort to be understood by humans. The trade-off between transparency and accuracy is well known in literature and can be well represented in Figure 1. However, increased complexity does not always lead to greater accuracy. Instead, more complex models provide greater flexibility, enabling them to approximate more intricate functions than simpler ones [1].

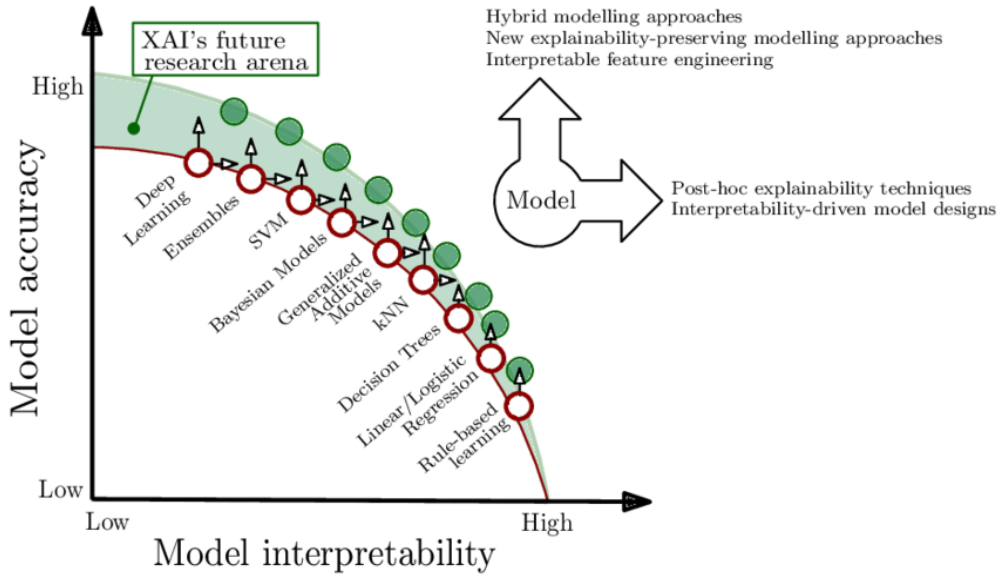

Figure 1: The trade off between model performance and interpretability

The objectives and research efforts of XAI encompass various goals beyond just model explainability and can be summarized as follows:

- Reliability (trustworthiness): the main characteristic of the XAI model, representing the confidence that a system will perform as expected when faced with a given problem. However, objectively defining and measuring trust remains a significant challenge.
- Causality: the ability to identify causal relationships among variables and in combination with domain knowledge, provide important insights rather than trivial conclusions.
- Transferability: the capability to transfer the knowledge learned from a specific problem to another one, while clearly defining the model's limitations and potential applications for improved problem-solving in a new context.
- Confidence: the level of robustness in decision-making also called stability is crucial for trustworthy interpretations and an explainable model should indicate confidence in its performance. The robustness ensures that even with small perturbations in the input data, the model remains consistent in its predictions.
- Fairness and Safety: ensuring that biased or unfair decisions are avoided by collecting and processing input data in a homogeneous and aggregated manner. This preserves anonymity, protects privacy, and ensures that no individual or group is unfairly disadvantaged by the model's decisions.
- Security: maintaining the confidentiality of knowledge discovered by mining algorithms is crucial, as other companies could use XAI techniques to launch adversarial attacks aimed at stealing information and bypassing data protection measures, these attacks aim to manipulate models by identifying minimal input changes needed to produce a specific output, such as a thin modification to a stop signal that confuses an autonomous vehicle's vision system.

Governments worldwide have introduced new regulations to enhance the trustworthiness of Artificial Intelligence systems, as presented by the European Union's AI Act in 2024 [3]. This regulation addresses AI-related risks by categorizing them into four levels: unacceptable, high, specific, and minimal. Stricter requirements have been fixed for transparency and explainability as the risk level increases.

In this context, interpretability plays a crucial role in ensuring compliance with these regulations. As proposed in [1] there can be distinguished two kinds of interpretability:

1. Perceptive interpretability consists of all the forms of interpretability that are intuitively understood by humans, often through visual representations. While these methods provide clear and accessible explanations, they fall short of fully addressing the core goal of XAI, as the underlying black-box model remains opaque. An example of perceptive interpretability is saliency mapping, which highlights the relative importance of input features and explanations can take various forms, such as probabilities or heatmaps.

2. Mathematical interpretability consists of all the approaches that unveil the inner workings of black box models such as deep neural networks in which complex information are stored inside their deeper layers. Other techniques, such as t-distributed Stochastic Neighbour Embedding (t-SNE) help map input subspaces, enhancing model understanding and improving prediction accuracy. This kind of interpretability requires a more advanced cognitive effort.

In general, all the non-transparent models need to be interpreted so different categories of approaches can be distinguished based on the scope, the stage or the kind of example used to make an explanation [2].

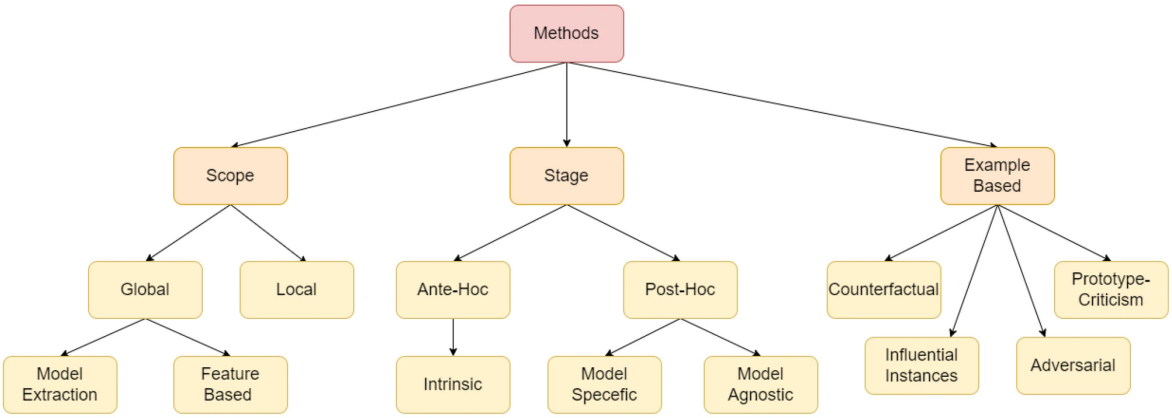

Figure 2: The main categories of interpretability methods

The *scope* defines how the XAI technique operates. It can be *global*, if the XAI approach aims to explain the overall behavior of a black box system by approximating it using simpler models such as linear models or decision trees. The resulting simpler model is called "surrogate model" and it is trained using a training dataset and as labels, the prediction produced by the original black box model. Well-known model extraction techniques include rule-based methods and model distillation. Alternatively, techniques based on feature importance can be used to understand the contribution to the predictions. One of the most famous methods that exploit this aspect is PFI (permutation feature importance). It can be *local*, if the explanation is focused on specific individual predictions rather than the overall model behavior. Local explanations aim

to clarify why a model made a particular decision for a given instance by analyzing the influence of input features on that specific prediction. These kinds of techniques are particularly useful when a used needs a case-by-case interpretability, e.g. to understand why a loan is rejected or not. Some of the most famous local techniques are LIME (Local Interpretable Model-agnostic Explanation) and SHAP (Shapley Additive Explanations).

The *stage* defines when the XAI technique is used. It can be used on the pre-processed data, *Ante-Hoc*, before the training, using classical machine learning techniques for data analysis and visualization to uncover possible relationships among data and possible correlations that may influence the further prediction of the black box model. Some of the most commonly used techniques are Linear Regressions, rule-based learning, and KNN. Their inherent interpretability often comes at the cost of accuracy, which is typically low. For this reason, post-hoc techniques are generally preferred, as they allow for improved model performance while providing explanations after training. It can be used after the training of the black box model, *Post-Hoc*. These methods interpret the model's behavior without modifying its structure. Depending on whether they leverage internal model knowledge or not, they are classified as *Model-Specific* (tailored to a particular model type) or *Model-Agnostic* (applicable to any machine learning model). The last type is more diffuse due to its flexibility, allowing to be applied to any model, including highly complex architectures like deep neural networks. Some of the most famous post-hoc approaches are, as cited in [4], text explanations (with symbolic representation of the knowledge), visual explanations (exploiting dimensionality reduction), feature relevance explanations (assigning a score to each feature), local explanations (focusing on a subset of the solution space less complex) and explanation by simplification (via surrogate model).

The *example-based* interpretability technique is a way in which an XAI can explain the behavior of a black box model by leveraging specific instances from the dataset used to train the model itself. In this category, different kinds of samples can be defined:

- *Prototypes* – a group of chosen training samples able to represent all the data. For each new data sample, the goal is to identify the most representative prototype, the one that shares the same class prediction as the sample and meets a specified similarity criterion.
- *Counterfactuals* - samples that highlight the minimal changes needed to the input feature values that would alter the model's prediction. In other words, they suggest what should be different in the input instance to change the outcome

of an AI system. These methods have rapidly grown in popularity and gained significant interest across various application domains and from different stakeholders. For example, in the case of a bank using AI for loan approvals, both the bank and the applicant are interested in understanding the reasons behind a decision. The bank can assess whether the decision aligns with regulations and objectives, while the applicant can determine how to respond to a rejection.

- *Adversarial* – instances accompanied by a minor perturbation with the intention to deceive an ML model [5]. The idea is to manipulate with very small variation the input data to force the model to have an incorrect prediction. These kinds of operations are also called adversarial attacks, and they can be used to analyze vulnerabilities in the model and to reinforce its robustness with further training.

- *Influential Instances* – it refers to all the samples that have a significant impact on the model's prediction. These instances are the ones that contribute most to shaping the model's decision for a particular outcome. Identifying influential instances helps to understand the model's behavior by highlighting which data points are most responsible for the model's predictions.

XAI is primarily applied in areas where the intersection of ethics and technology plays a crucial role, particularly in ensuring accountability for AI-driven decisions. Next are presented the key domains where the need for explaining how machine learning models generate automated decisions is most prevalent today.

Financial

The financial sector uses artificial intelligence techniques to implement its services, in particular, the granting of mortgages and financing to customers. These operations are often subject to discriminatory decisions and controversies from those who find themselves excluded from certain guaranteed terms or excessively limited in terms of amounts that can be disbursed in a mortgage or financing. The use of XAI is essential to ensure the transparency of the process in which the bank grants a loan and, at the same time, contributes to improving the system's training model, maintaining a certain discretion with respect to the bank policies.

Self-Driving

Autonomous driving is becoming increasingly feasible. Vehicles with autonomous driving can substitute human drivers in different levels of automation. However, many people are sceptical about this fact and do not trust these systems because they do not risk their lives with something that they do not fully understand. XAI, therefore, plays a key role by making AI-driven decisions more transparent. By providing clear

explanations of how autonomous vehicles ensure safety and reliability on the road, XAI helps build public trust and facilitates broader acceptance of this technology.

Military

Every military decision is critical and must be justified with strong reasoning. Similarly, major economic decisions require transparency, ensuring taxpayers understand the rationale behind significant investments. XAI plays a vital role in military strategy by aiding in the evaluation of possible battlefield choices that directly impact soldiers. By providing explainability and insight into AI-driven decisions, XAI becomes an essential tool for enhancing accountability, strategic planning, and trust in autonomous systems.

Medical and Bio-Medical

In the medical field, AI is widely used to detect and prevent numerous diseases, often identifying patterns beyond human capability. It also supports critical medical decisions that can save lives. Given its direct impact on patient health, AI-driven recommendations must be thoroughly justified. This makes interpretability in healthcare not just a matter of academic interest but a crucial necessity, ensuring transparency, trust, and accountability in life-critical decisions. Similarly, the biomedical field benefits from AI, particularly in areas like **Brain-Computer Interfaces** (BCIs), where XAI helps interpret brain signals and enhance human- computer interaction. These advancements bridge the gap between neuroscience and technology, enabling innovative applications in patient care, rehabilitation, and assistive technologies. BCIs are often directed at researching, mapping, assisting, augmenting, or repairing human cognitive or sensory-motor functions. They are often conceptualized as a human-machine interface that skips the intermediary of moving body parts (e.g., hands or feet), although they also raise the possibility of erasing the distinction between brain and machine [6].

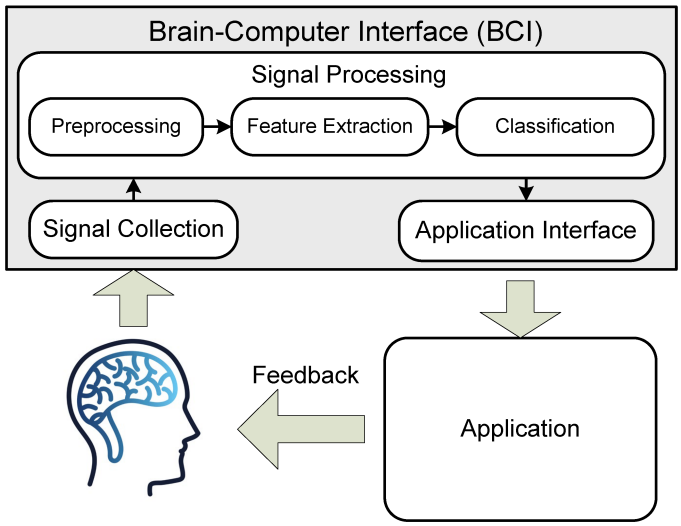

Figure 3: BCI working schema. Image from [7]

BCI is also strongly connected to *Emotional Intelligence*, which refers to the ability for people to recognize, understand, and perceive emotions effectively in everyday life [8]. High emotional intelligence includes emotional recognition of the emotions of the self and others, using emotional information to guide thinking and behavior, discerning between and labeling different feelings, and adjusting emotions to adapt to environments [9]. Emotion recognition research seeks to enable computers to identify emotions [10] by analyzing physiological signals, facial expressions, voice patterns, and brain activity. Its measurement is a challenging task, but with the support of XAI, this technology is finding impactful applications across various fields.

In healthcare, it can aid in detecting conditions such as depression, anxiety, and PTSD by interpreting physiological and neurological signals, enabling early intervention and personalized treatment. In education, emotion-aware systems can enhance learning by adapting to student engagement levels and frustration, providing tailored support for those with learning difficulties. In autonomous systems and robotics, emotion recognition can improve driver safety by detecting a driver's emotional state, such as stress or fatigue. Autonomous vehicles could then adjust their behavior or issue warnings, leading to safer interactions between human drivers and AI-driven systems.

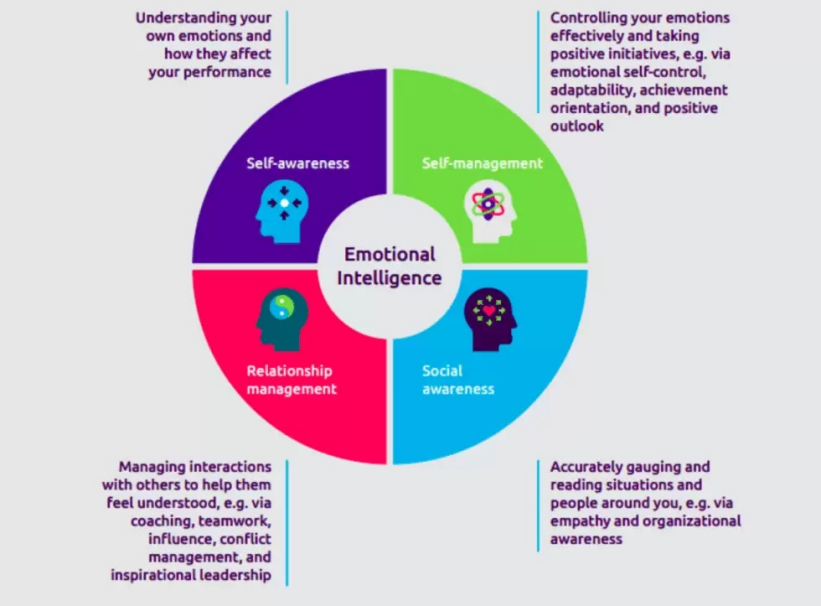

Figure 4: The four main skills of Emotional Intelligence

2 Related works

This section provides a review of the most significant studies on counterfactual explanation generation, along with research on latent space methods. The reviewed techniques have been carefully analyzed to get a clear picture of the state of the art and have played an important role in shaping this thesis. Some of them have also been practically tested to explore their potential for future re-utilization and extendibility.

2.1 Dimensionality Reduction

Generating a latent space involves reducing the dimensionality of the data. The well-known curse of dimensionality poses significant challenges, as high-dimensional spaces can lead to sparsity, making it difficult to model and analyze data effectively.
Various techniques, such as principal component analysis (PCA)[11], are commonly employed to address this issue by capturing the most relevant features while discarding noise. PCA is one of the most famous and it identify the principal components, orthogonal directions that capture the most variance in the data, allowing for efficient data compression and noise reduction. However, it cannot capture non-linear transformations, which led to the development of t-SNE[12]. It preserves local relationships by mapping similar data points close together in a lower-dimensional space, making it particularly useful for clustering and pattern recognition. Other nonlinear techniques, such as autoencoders, leverage neural network architectures to capture the underlying data by using an encoder to compress the input into a lower-dimensional latent space and a decoder to reconstruct the original data from this representation. From this basic structure, several architectures have been developed, such as the VAE[13], which learns the input as a distribution (typically Gaussian), rather than a fixed point, and ensures generative capabilities by sampling from the latent space. While non-linear data reduction techniques can enhance data compression, they often lead to less interpretable results. Additionally, using neural networks to explain other black-box models do not address the core issue of the original model's explainability.

2.2 Counterfactuals explanations methods

In literature there are several approaches useful for the generation of specific counterfactuals. Next are mentioned some of the most famous and reliable for the target of this thesis.

Diverse Counterfactual Explanations (**DICE**)[14] is a method focused on generating diverse and valid counterfactual explanations with the goal of offering a broader perspective on the different ways the model can make decisions. It consists of an optimization problem, where the concepts of diversity and proximity are combined in a mathematical formula, in addition to accuracy, to obtain a set of counterfactuals with these properties. Diversity is defined as the ability to generate counterfactuals that are not similar to each other, encouraging exploration of different ways to change the black-box decision. Proximity refers to how close a counterfactual is to the original sample while still being able to change the model's decision. Accuracy measures how well the generated counterfactuals match the target outcome. Below is the loss function from the original paper:

$$C(x) = \arg\min_{c_1,\ldots,c_k} \left(\frac{1}{k}\sum_{i=1}^{k} \text{yloss}(f(c_i), y) + \lambda_1 \frac{1}{k}\sum_{i=1}^{k} \text{dist}(c_i, x) - \lambda_2 \cdot \text{dpp\_div}(c_1, \ldots, c_k) \right) \tag{1}$$

where c_i is a counterfactual example (from CF counterfactual set), k is the total number of CFs to be generated, $f(\bullet)$ is the ML model (a black box to end users), $yloss(\bullet)$ is a metric that minimizes the distance between $f(\bullet)$'s prediction for c_is and the desired outcome y, $dist$ is a metric to measure the distance from samples, x is the original input, and dpp_div is the diversity metric. λ_1 and λ_2 are hyperparameters that balance the three parts of the loss function.

Another interesting method is Growing Spheres Generation (**GSG**)[15]. This approach consists in gradually building a sphere composed of synthetic instances with the sample instance as the center, trying to discover where the closest counterfactuals are located, and minimizing a specific cost function through a heuristic search strategy. It focuses mainly on the minimality metric detecting what is the minimal change to apply on the original instance to obtain a proper counterfactual. The search is performed randomly in all possible directions until the decision boundary is crossed. This method is simple and intuitive but highly dependent on parameters such as radius size, increment step size, and sampling density. It may not be the ideal choice when dealing with complex or nonlinear decision boundaries. The selection of appropriate parameters significantly impacts both computation time and the minimality of the generated counterfactuals.

A more specific solution is SHapley Additive exPlanations (**SHAP**)[16] with leverage the cooperative game-theory approach to explain the output of any ML model via global and local explanations. SHAP calculates feature contributions, treating each feature as a "player" in a coalition (alias for a group of players in a cooperative game). It gives the possibility to visualize the outcome (SHAP values) in some plots and makes it easy to understand the importance and contribution of each feature in the model decision. This concept is used and extended in another related recent work for the generation of counterfactuals Shapley Values for the explainability of reinforcement learning agent behaviors [17].

An alternative fascinating technique that also exploits counterfactual explanations and merit to be cited is LOcal Rule-based Explanations (**LoRE**)[18]. This method is able to provide local explanations in a very efficient and intuitive way. It first leans a local interpretable predictor on a synthetic neighborhood generated by a genetic algorithm. Then, it derives from the logic of the local interpretable predictor a meaningful explanation consisting of a decision rule, which explains the reasons of the decision and a set of counterfactual rules, suggesting the changes in the instance's features that lead to a different outcome exploiting counterfactual explanations. This work has demonstrated that LORE outperforms state-of-the-art existing methods and baselines in both the quality of explanations and the accuracy in replicating the black box model's behavior. Although it is one of the clearest and most informative model-agnostic solutions, in its first version, it was able to work only with tabular datasets and was used for binary classification problems. These limitations have been addressed in the new version, **LoreSa**, proposed in [19], where "Sa" stands for "stable" and "actionable". This work aims to enhance comprehensibility, fidelity, minimality, and generality while also incorporating stability and actionability. A key improvement is the emphasis on actionability, allowing users to specify which features can be considered during the explainability process, thereby preventing the generation of impractical counterfactuals. For example, it ensures that counterfactuals do not involve modifications to immutable attributes such as race or sex. The performance improvement is achieved by constructing a simple, interpretable local decision tree predictor.

This process involves three key steps:

1. Generating an ensemble of balanced neighbor instance sets around the target sample using an ad-hoc genetic algorithm.

2. Training a decision tree classifier for each generated set.

3. Merging the ensemble of decision trees into a single decision tree classifier.

A (counter)factual explanation is then derived from the final decision tree, which locally approximates the behavior of the black-box model around the target instance. Stability is ensured through this bagging-like approach. The final output consists of:

1. A rule-based explanation clarifying why a specific instance was classified with a given label.

2. A counterfactual rule indicating what modifications to the instance would lead to a different outcome.

Additionally, LoreSa supports image and text data, making it capable of explaining multi-class classification problems.

A recent advancement in counterfactual generation methods is introduced in [20], where the **CP-ILS** is proposed. The acronym stands for Counterfactual and Prototypical Explanations for Tabular Data via Interpretable Latent Space. This novel approach aims to research explanations not directly in the input space, where most of the algorithms do, but in a created latent space. CP-ILS optimizes a transparent feature space whose similarity and linearity properties enable a better extraction of local and global explanations for any pre-trained black-box model in the form of counterfactual/prototype pairs. The authors demonstrate that this method is able to preserve pair-wise similarities like well-known dimensionality reduction techniques, proving that it is able to obtain more robust, plausible, and accurate explanations than its competitors under most experimental conditions. The latent space plays a crucial role because, in a reduced dimensional space, it helps to better recognize what are real and robust counterfactuals with respect to adversarial samples. These adversarial examples are inadequate as counterfactuals due to their location in low-density regions, outside the distribution of similarly labeled examples. This misplacement fails to represent their predicted class, making them both implausible and non-robust as explanations. Prototype explanations analyze black-box behavior through analogy, a reasoning approach commonly used by humans. Prototypes represent data records that capture key characteristics of a predicted class. An instance is classified based on its similarity to these prototypes, allowing key decisions of the black-box model to be understood through comparison. This approach aims to generate feasible and effective counterfactuals and prototypes while ensuring actionability, enhancing transparency in latent space representation. However, it is currently limited to tabular datasets for binary classification problems. As the authors suggest, improving the optimization method could expand its applicability to other types of data. On the next page, a clear graphical example is provided to demonstrate the capabilities of the method.

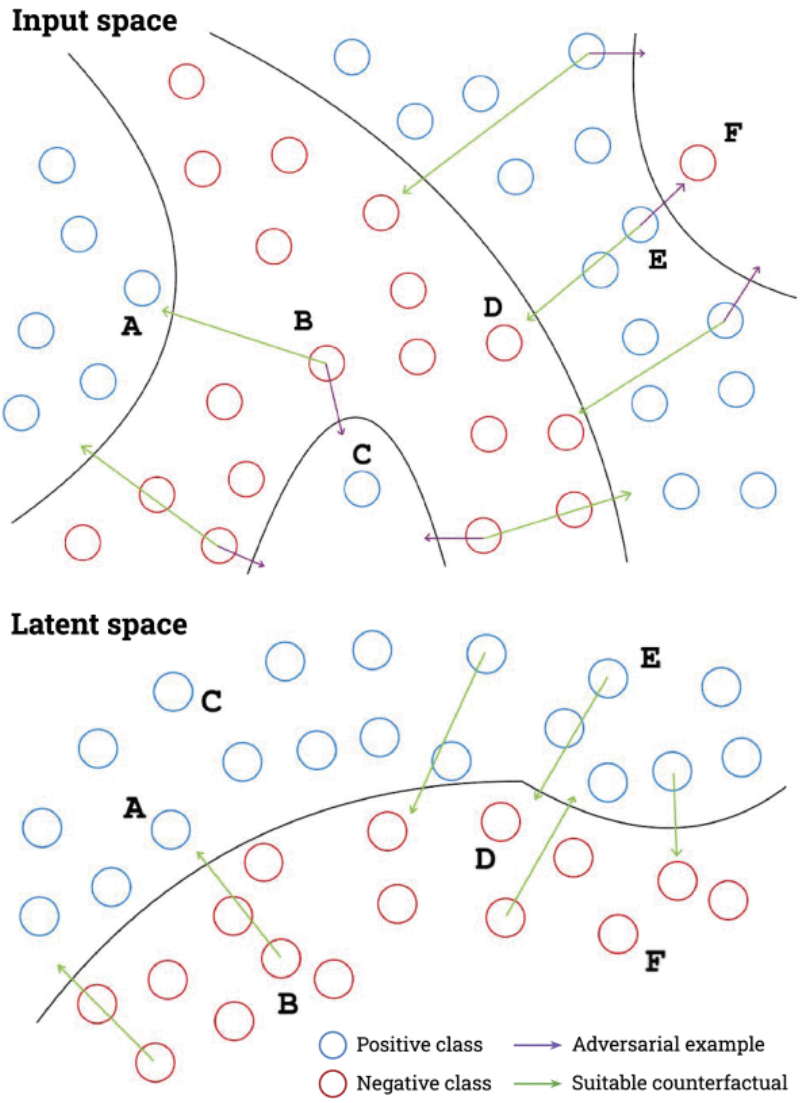

Figure 5: Illustration in 2D of an adversarial problem in counterfactual explanations and the proposed solution with CP-ILS taken from [20].

As the figure proposes, given the input samples B and E in the original data space, the adversarial examples C and F are the closest instances that change the model's decision and might be mistaken for acceptable counterfactuals. These examples lie in low-density regions that are not representative of the predicted class, limiting the efficacy of the explanation. In contrast, instances A and D are located in dense regions of examples with a different predicted class, resulting in more robust and plausible counterfactuals for B and E. On the right, by learning the latent space according to feature and prediction similarities, the previously adversarial points C and F are no longer retrieved as good counterfactuals because they are not among the closest points to the decision boundary for B and E, respectively. In contrast, instances A and D remain robust and plausible counter-exemplars, as they are in the input space.

3 Design and implementation

The work presented in [20] is the most relevant to the objectives of this thesis. The key idea is to leverage the contributions of that work to conduct targeted research in the latent space, with a focus on generating specific and minimal counterfactuals in a multi-class setting while striving to preserve the performance achieved in the binary classification context. The original method, hereafter referred to as CPILS, is addressed in this thesis only in relation to counterfactual generation, with the global explanation of prototypes left for future research. Before discussing the enhancements introduced in this work, it is essential to first explain how the CPILS operates for binary classification to provide a clear understanding of its extension.
The following paragraph explains the two main steps of the method: latent space creation and counterfactual generation, delving into the mathematical intuition that underpins its validity as a counterfactual approach. The subsequent paragraph then introduces the enhancements required for adapting the method to a multi-class scenario, discussing the newly introduced modalities and features.

3.1 CPILS

3.1.1 The generation of the latent space

The first step applies a dimensionality reduction technique to learn a compact representation of the original input space while preserving interpretability. This is achieved through a Linear Encoder, which learns the weights that map the original input data to a lower-dimensional space. This reduced space, known as the Latent Space, is inherently transparent due to its reliance on linear properties. Moreover, it preserves the proximity relationships among data points, effectively mapping high-dimensional input samples to lower-dimensional latent representations while maintaining their structural integrity. Through this linear transformation, the contributions of each feature to the prediction can be better analyzed, facilitating a seamless transition between the latent space and the original input space. Additionally, the latent space is enriched by incorporating predictions from a black-box model, augmenting the data with predictive information and integrating it directly into the weights matrix. This additional information is crucial for the next step, which focuses on identifying valid counterfactuals.

The latent space learning is realized through gradient optimization techniques, defining an objective function that, similar to t-SNE (Stochastic Neighbor Embedding), aims to preserve similarities among data points in a reduced space. The similarity between two instances $\mathbf{x}$ and $\mathbf{u}$ is defined as the probability that $\mathbf{x}$ would select $\mathbf{u}$ as its neighbor, assuming neighbors are chosen in proportion to their probability density under a

Gaussian distribution centered on $\mathbf{x}$. In formula:

$$S_d(\mathbf{x}, \mathbf{u}) = \frac{\exp\left(-\frac{d(\mathbf{x},\mathbf{u})}{2\sigma^2}\right)}{\sum_{\mathbf{v}\in X, \mathbf{v}\neq\mathbf{x}} \exp\left(-\frac{d(\mathbf{x},\mathbf{v})}{2\sigma^2}\right)} \tag{2}$$

where X is the set containing the instances, $\mathbf{x}$ and $\mathbf{u}$ are the two instances used to compute the similarity, σ is the variance of the Gaussian, and d represents the distance metric between $\mathbf{x}$ and $\mathbf{u}$.
The distance metric used is specifically designed to accommodate the varying features within tabular data. It combines three distinct distance measures.

- **Hamming distance**[21] for categorical features:

$$d_H(x, y) = \sum_{i=1}^{n} \mathbb{1}(x_i \neq y_i) \tag{3}$$

  The Hamming distance $d_H(x, y)$ between two vectors x and y is defined as the sum of indicator functions over all components, where:

  - x_i and y_i are the components of the vectors x and y, respectively.
  - $\mathbb{1}(x_i \neq y_i)$ is the indicator function, which returns 1 if $x_i \neq y_i$ (i.e., the components differ) and 0 if $x_i = y_i$ (i.e., the components are the same).
  - The summation runs over all i from 1 to n, where n is the number of components in the vectors x and y.

- **Cosine distance**[22] for numerical features:

$$d_{\cos}(x, y) = 1 - \frac{x \cdot y}{|x||y|} \tag{4}$$

  The cosine distance $d_{\cos}(x, y)$ measures the dissimilarity between two vectors based on the cosine of the angle between them (larger distances correspond to greater dissimilarity). It is the difference between 1 and the cosine similarity and is defined as:

  - $x \cdot y$: The dot product of the vectors x and y, calculated as $x \cdot y = \sum_{i=1}^{n} x_i y_i$.
  - $|x|$ and $|y|$: The Euclidean (L2) norm of vectors x and y, computed as:

$$|x| = \sqrt{\sum_{i=1}^{n} x_i^2}, \quad |y| = \sqrt{\sum_{i=1}^{n} y_i^2}$$

  - $\frac{x \cdot y}{|x||y|}$: This term represents the concept of cosine similarity

- **Prediction contribution** accounts for the impact of the black-box model's predictions.

$$d_{\text{pred}}(x, y) = |b(x) - b(y)|^2 \tag{5}$$

This distance quantifies the dissimilarity between instances x and y in the model's output space as the squared difference of their predictions:

  - $b(x)$, $b(y)$: Predictions of the black-box model for x and y.
  - $|b(x) - b(y)|^2$: Squared difference, amplifying larger discrepancies.

The final distance metric used inside (2) take into account the contribution of (3) (4) and (5) with different weights:

$$d_{\text{mix}}(x, y) = \left(\frac{h}{m}\right) d_H(x, y) + \left(\frac{h - m}{m}\right) d_{\cos}(x, y) + d_{\text{pred}}(x, y) \tag{6}$$

where h represents the number of categorical features, $h - m$ the number of numerical features, and m the total number of features.

All pairwise similarities between points are stored in a matrix, conventionally denoted as S. This process is applied during training in both the high-dimensional input space and the lower-dimensional latent space, producing two distinct similarity matrices. To differentiate them, S_X represents the similarity matrix in the input space, while S_Z corresponds to the similarity matrix in the latent space.
The training procedure optimizes a Kullback-Leibler (KL)[22] divergence loss function to ensure optimal performance by aligning the similarity structures in both spaces. The goal is to minimize the discrepancy between the similarity matrices in the input and latent spaces, making them as comparable as possible. By reducing this difference, the loss function decreases, indicating better preservation of similarity relationships from the high-dimensional input space to the lower-dimensional latent space. A lower loss value signifies a more faithful representation of the original structure. In formula:

$$KL(S_X \parallel S_Z) = \frac{1}{|X|} \sum_{\substack{x \in X \\ u \in X \setminus x}} S_X(x, u) \log \frac{S_X(x, u)}{S_Z(z(x), z(u))} \tag{7}$$

where:

- $x, u \in X$ are two points from the input space X.
- $S_X(x, u)$ is the similarity between the points x and u in the input space.
- $S_Z(z(x), z(u))$ is the similarity between the corresponding points $z(x)$ and $z(u)$ in the latent space.

- $|X|$ is the total number of points in the input space.
- The logarithmic term measures the ratio of the similarities in the input space to those in the latent space, indicating how well the latent space approximates the input space's similarity structure.

To determine an appropriate latent dimension, which cannot be predefined, a method based on overfitting detection is proposed. Beginning with the smallest possible dimension (2), the approach incrementally increases the dimension while assessing performance using a KNN algorithm[24]. Overfitting is detected by monitoring accuracy as the dimension increases. This method also provides flexibility for others to implement their own approach, allowing for the exploration of model behavior across different latent dimensions. The pseudo-code summarizing the latent space generation process, including the determination of the latent space dimension, is provided below.

Algorithm 1 LatentSpaceLearning(X, Y, b)

Input: X - data samples, Y - data labels, b - black-box
Output: W - trained transformation model

$n \leftarrow \text{features}(X)$ ▷ Compute feature length
$k \leftarrow 2$ ▷ Initialize latent space dimensions
$ACC_1 \leftarrow 0$ ▷ Initialize latent space score
$\hat{Y} \leftarrow b(X)$ ▷ Obtain black-box predictions
$S_X \leftarrow \text{PairwiseSimilarity}(X)$ ▷ Original similarity matrix
while $k < n \wedge ACC_{k-1} < ACC_k$ **do**
    $W \leftarrow \text{init}()$ ▷ Initialize model weights
    repeat
        $Z \leftarrow W^T \cdot [X, \hat{Y}]$ ▷ Compute latent representation
        $S_Z \leftarrow \text{PairwiseSimilarity}(Z)$ ▷ Compute latent similarity matrix
        $L \leftarrow \text{KLD}(S_X \parallel S_Z)$ ▷ Compute KL-Divergence
        $W \leftarrow \text{BackProp}(W, L)$ ▷ Update weights
    until convergence
    $Z \leftarrow W^T \cdot [X, \hat{Y}]$ ▷ Compute latent space
    $ACC_k \leftarrow \text{KNN}(Z, Y)$ ▷ Compute validation score
    $k \leftarrow k + 1$
end while
return W

3.1.2 Counterfactuals generation

This phase assumes that the latent space has already been created, and the weight matrix W is ready to be used in the counterfactual search. As previously mentioned, the weight matrix also encodes information about the black-box model's predictions in the original input space, referred to as the "prediction direction" (the last row of the weight matrix W). This prediction direction is now leveraged to guide the counterfactual search within the generated latent space. For simplicity, from now on, it is used $z(x)$ and $M(x, \hat{y})$ interchangeably to indicate that the latent mapping always depends on both x (the input instance) and $b(x)$ (the prediction of the black box for that specific instance). Through linear transformations, it is possible to isolate the contribution of each feature to the prediction. Each input feature x_i determines a direction of movement in the latent space, contributing to a vector of k dimensions. When summed together, these vectors determine the final latent position:

$$z_l = \sum_{i=1}^{n} w_{il} x_i + w_{\hat{y}l} \hat{y} \tag{8}$$

where the rows $w_i = [w_{il}]_{l=1...k}$ of the weight matrix W represent the latent directions corresponding to the features x_i. This equation describes how each latent component z_l is a linear combination of the input features x_i and the predicted value $\hat{y}$, with the respective weights determining their contributions in the latent space. At this point, all latent samples along the prediction direction **share similar input features** but **differ in their prediction labels**.

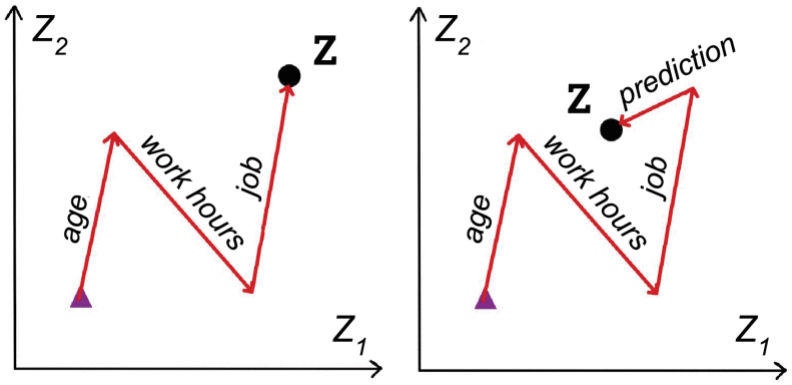

Figure 6: 2D latent space representation highlighting the contributions of input features.

On the left, each latent point is the sum of the projections of the input features onto the directions w_i, defined by the rows of the transformation matrix. On the right, concatenating the black-box prediction to the input generates a prediction direction, where similar attributes $w_{\hat{y}}$ yield different predictions.
To meet the actionability requirement, the user can specify a list of features to remain

fixed before initiating the search. This ensures that the selected features are excluded from the counterfactual generation process, preventing the creation of implausible results.

The research begins with the latent mapping of the original input sample into the latent space $M(x, \hat{y})$. Then, by considering only the features that are accountable, it determines which ones cause the most shift along the prediction direction. The intuition is to use an iterative approach, where the prediction direction is followed in small predefined steps, and at each iteration, a new latent position is incrementally determined. This position is identified by the vector v:

$$v_k = z + k \cdot s \cdot w_{\hat{y}}, \quad k \in \mathbb{N} \tag{9}$$

where:

- v_k represents the latent position at step k,
- z is the latent sample,
- s is the degree of increment (established a-priori),
- $w_{\hat{y}}$ is the prediction direction,
- k is a positive integer (indicating the different iteration).

Once this vector is obtained, Lagrange optimization is applied to determine the minimal changes Δ (in input space) required to transform the original instance into a new sample that corresponds to v in latent space. Given the infeasibility of computing all possible feature combinations among the accountable features, only a subset of combinations, denoted by τ, is considered. This approach is also preferable from a human interpretability perspective, as counterfactuals with fewer feature changes are typically more understandable. Given the linearity of the mapping, the Lagrange multipliers method [25] is used to optimize the squared norm of feature modifications $f = \|\Delta\|^2$ subject to constraints $\{g_l\}$ that guarantee the final latent position v exactly matches the input feature perturbation $x + \Delta$ after being mapped through W. The

$$\begin{cases} f = \frac{1}{2} \sum_{i \in \tau} \Delta_i^2 \\ g_l = \sum_{i=1}^{n+1} w_{il}(x_i + \Delta_i) - v_l \end{cases} \Rightarrow \begin{cases} \nabla f(\vec{x}) = \sum_{l=1}^{k} \lambda_l \nabla g_l(\vec{x}) \\ g_l(\vec{x}) = 0, \quad l = 1, \ldots, k \end{cases}$$

Lagrange multipliers are identified in the formula as λ_l, and by substituting f and g into the Lagrange equations, the problem becomes a linear system whose solutions are the minimal changes Δ to be applied to the original input feature.

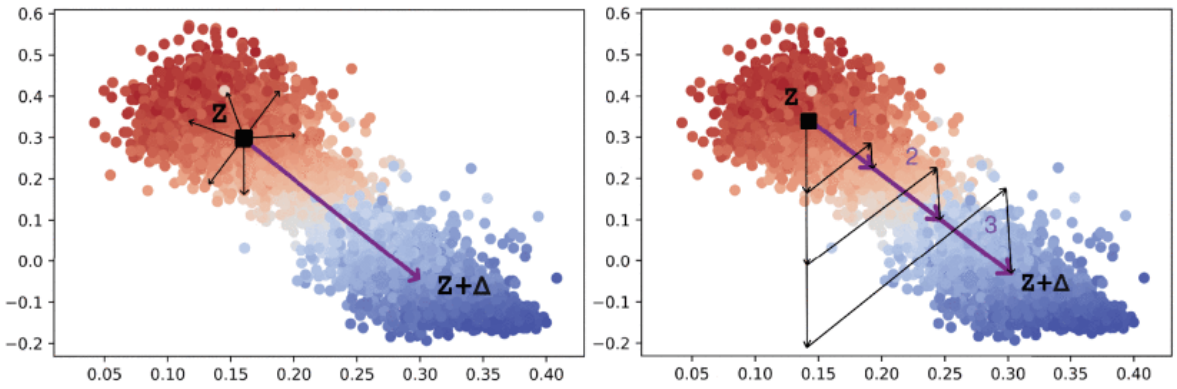

Figure 7: Visualization of a 2D latent space emphasizing the prediction direction followed in the search for counterfactuals.

In this figure, the black square represents a sample z for which counterfactuals are being explored. Violet arrows indicate the intended trajectory, while black arrows illustrate the influence of input features on the latent space. On the left, an optimal feature combination guides movement toward the decision boundary. On the right, the same transition is achieved through a progressive search. To highlight the different classes, latent points are color-coded: red for one class and blue for the other, reflecting the binary classification context. The color intensity represents the black-box model's prediction confidence, with lighter shades indicating lower probability and darker shades indicating higher confidence. The pseudo-code summarizing the counterfactuals research process is provided below.

Algorithm 2 CounterfactualSearch$(\vec{x}, W, b, F, s, \epsilon)$

Input: $\vec{x}$ - sample to explain, W - latent space transformation, b - black-box
Parameters: F - feature set to modify, s - starting step, ϵ - step increment
Output: C - set of counterfactual explanations

$\hat{y} \leftarrow b(\vec{x})$ ▷ Compute sample prediction
$\vec{z} \leftarrow W^T \cdot [\vec{x}, \hat{y}]$ ▷ Compute latent space position
$C \leftarrow \emptyset$
for $\tau \in$ combinations(F) **do**
    $\Delta \leftarrow \vec{0}$
    while $b(\vec{x} + \Delta) = \hat{y}$ **do** ▷ Until prediction changes
        $\vec{v} \leftarrow \vec{z} + s \cdot \vec{w}_{\hat{y}}$ ▷ Set the destination
        $\Delta \leftarrow$ Lagrange$(\vec{v}, \vec{x}, W, \tau)$ ▷ Optimize step
        $s \leftarrow s + \epsilon$ ▷ Compute new step
    end while
    $C \leftarrow C \cup \{\vec{x} + \Delta\}$ ▷ Store counterfactuals
end for
return C

3.2 CPILS for multi-class problems

The CPILS method is an effective approach for exploring counterfactuals in tabular datasets, specifically for binary classification problems. Its core intuition guides the counterfactual search within a linear transparent latent space, achieved through the introduction of the prediction direction within that reduced space. This approach is inherently distinctive and original.

This thesis builds upon this foundation, extending the concept from binary classification and adapting it to multi-class scenarios. The goal is to approximate the behavior of multi-class black-box classifiers by decomposing them into multiple binary classification problems within latent spaces. By applying the standard CPILS counterfactual generation method to these binary latent spaces, the original decision model can be explained through these counterfactual approximations. With this approach, the prediction direction between two classes is preserved, guiding the counterfactual search in specific directions.

In counterfactual search, the objective is to cross the decision boundary, as this is what generates counterfactuals. However, in a multi-class classifier, this boundary is significantly more complex. This approach aims to simplify the process of crossing the decision boundary, as it is generally easier with a binary classifier. In contrast, a standard multi-class classification scenario involves a much more intricate decision boundary.

An alternative approach could involve extending the CPILS method directly to operate within a unified multi-class latent space, where multiple prediction directions are considered simultaneously. However, this would require additional theoretical validation and the development of more efficient and effective methods to determine the minimal change to each counterfactual. This remains an area for future research.

The following paragraphs provide a detailed explanation of how latent space generation and counterfactual search are adapted to a multi-class scenario. To avoid confusion, from this point onward, a binary black-box model designed to explain the original multi-class black-box model will be referred to as **BBB**, while the original multi-class black-box model will be denoted as **MBB**.

3.2.1 Generation of the latent spaces: multiclass scenario

It first starts detecting what are the classes involved in the problem and it decomposes the multi-class dataset into multiple binary classification problems. For each specific pair of classes, it trains a BBB with the same characteristics as the original MBB. Once trained, a dedicated latent space is generated for each binary classifier, considering only the two involved classes using the original CPILS latent space learning method. Within this *binary latent space* the prediction direction is clearly defined, allowing counterfactual generation to follow the original CPILS approach. This method returns all the data structures related to the generated latent spaces and the BBBs. Here is reported the pseudo-code that summarizes the procedure.

Algorithm 3 MultiClass_LatentSpaceLearning(X, Y, MBB)

Input: X - data samples, Y - data labels, MBB - multi-class black-box trained model
Output: LSS - trained latent spaces set, BBS - binary black box set

$LSS \leftarrow \emptyset$ ▷ Initialize the latent spaces set
$BBS \leftarrow \emptyset$ ▷ Initialize the binary black box set
$classes \leftarrow \text{unique}(Y)$ ▷ Obtain unique classes
for $(c_1, c_2) \in \text{combinations}(\text{classes})$ **do**
    $X_{\text{comb}}, Y_{\text{comb}} \leftarrow \text{splitDataset}(X, Y, c_1, c_2)$ ▷ Extract samples for this comb.
    $BBB \leftarrow \text{copyStructure}(MBB)$ ▷ Initialize BBB as MBB
    $BBB \leftarrow \text{train}(X_{\text{comb}}, Y_{\text{comb}})$ ▷ Train BBB
    $W_{\text{comb}} \leftarrow \text{CPILS\_Latent\_Space\_Learning}(X_{\text{comb}}, Y_{\text{comb}}, BBB)$
    $LSS \leftarrow LSS \cup W_{\text{comb}}$ ▷ Update latent space set
    $BBS \leftarrow BBS \cup BBB$ ▷ Update binary black box set
end for
return LSS, BBS

3.2.2 Counterfactuals generation: multiclass scenario

This method assumes that the sets of latent spaces and BBBs have already been created in the previous step and are provided as input parameters. The first operation involves using the MBB to predict the current outcome and determining which latent space will be consulted to generate counterfactuals. Then, for each possible combination of classes, it checks if the predicted class is involved. It retrieves the corresponding latent space weights and BBB classifier and subsequently calls the CPILS counterfactual generation method. In a binary classification context, the counterfactual search for a specific feature combination concludes when either the prediction of a newly generated input sample (derived from the latent space) differs from that of the original target sample or when the maximum number of predefined iterations is reached, ensuring that the identified solution qualifies as a counterfactual. However, in the multi-class scenario, the counterfactual candidates obtained through binarization must be validated by the original black-box multi-class model to determine whether they are indeed true counterfactuals.

Algorithm 4 MultiClass_CounterfactualSearch($\vec{x}, MBB, LSS, BBS, b, F, s, \epsilon$)

Input: $\vec{x}$ - sample to explain, MBB - multi-class black-box trained model, LSS - trained latent spaces set, BBS - binary black box set
Parameters: F - feature set to modify, s - starting step, ϵ - step increment
Output: CS - set of counterfactual explanations

$CS \leftarrow \emptyset$ ▷ Initialize empty counterfactuals set
$y_{\text{pred}} \leftarrow MBB(\vec{x})$ ▷ Predict label for the sample
for $(c_1, c_2) \in$ combinations(classes) **do**
    if $isInvolved(y_{\text{pred}}, c_1, c_2)$ **then**
        $W_{\text{comb}} \leftarrow$ getLatentSpace(LSS, c_1, c_2) ▷ Retrieve latent space
        $BBB \leftarrow$ getBinaryBlackBox(BBS, c_1, c_2) ▷ Retrieve BBB
        $C \leftarrow$ CPILS_CounterfactualSearch($\vec{x}, W_{\text{comb}}, BBB, F, s, \epsilon$)
        $CS \leftarrow CS \cup C$ ▷ Store counterfactuals
    end if
end for
for cf $\in CS$ **do** ▷ Counterfactuals validation
    if $BBB(\text{cf}) = y_{\text{pred}}$ **then** ▷ Ensure it is a valid counterfactual for BBB
        discard(cf)
    else
        if $MBB(\text{cf}) = y_{\text{pred}}$ **then** ▷ Ensure it is a valid counterfactual for MBB
            discard(cf)
        else
            $PSC \leftarrow PSC \cup \text{cf}$ ▷ Add to the set of acceptable counterfactuals
        end if
    end if
end for
return PSC

3.2.3 Different modalities

Since many popular binary classification algorithms are commonly adapted for multi-class contexts using similar strategies, this method follows the same approach. The two functions introduced in the previous sections are designed for generic class combinations in a binary context, although the way these pairs of classes are formed can vary depending on the method. This paragraph outlines three approaches adopted to handle class combinations: One-vs-One (1v1), One-vs-Rest (1vAll), and Rest-vs-One (Allv1), and provides an explanation of how each approach functions.

One-vs-One (1v1)

In this mode, a binary classifier is trained for every possible pair of classes. If there are N classes, the number of binary classifiers required is $\frac{N(N-1)}{2}$.

- The `splitDataset(X, Y, c_1, c_2)` function extracts only the samples from the original dataset that belong to the specified classes c_1 and c_2.
- The `combinations(classes)` function generates all possible pairs of classes. For each pair, a binary latent space is trained and utilized to guide the search in a specific direction where the counterfactual label is involved.
- The function `isInvolved(y_pred, c_1, c_2)` returns true if *y_pred* corresponds to either c_1 or c_2.

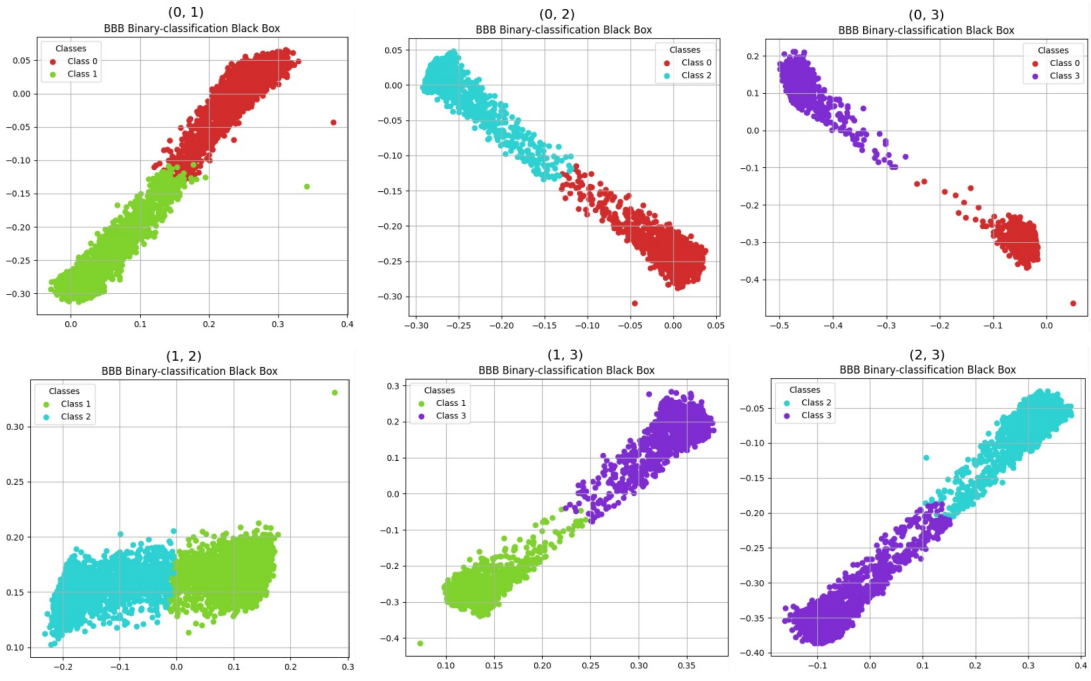

Figure 8: Visualization of 2D latent spaces for a 4-class classification in 1v1 mode.

One-vs-Rest (1vAll)

In this mode, a separate binary classifier is trained for each class. Each classifier learns to distinguish one specific class from all the others combined. If there are N classes, the approach requires N binary classifiers.

- The function `splitDataset(X, Y, c_1, c_2)` is designed to return a subset of the dataset where instances of the target class c_1 (one) are retained, while instances from other classes c_2 (rest) are sampled randomly in equal proportion. This ensures a balanced representation between the target class and the competing classes, improving the classifier's ability to differentiate them effectively.

- The function `combinations(classes)` returns the complete set of N possible classes.

- The function `isInvolved(y_pred, c_1, c_2)` returns true if y_{pred} matches only the target class c_1, also referred to as the starting class. In this setting, the goal is to conduct a counterfactual search, exploring a direction where instances are labeled differently from the original class.

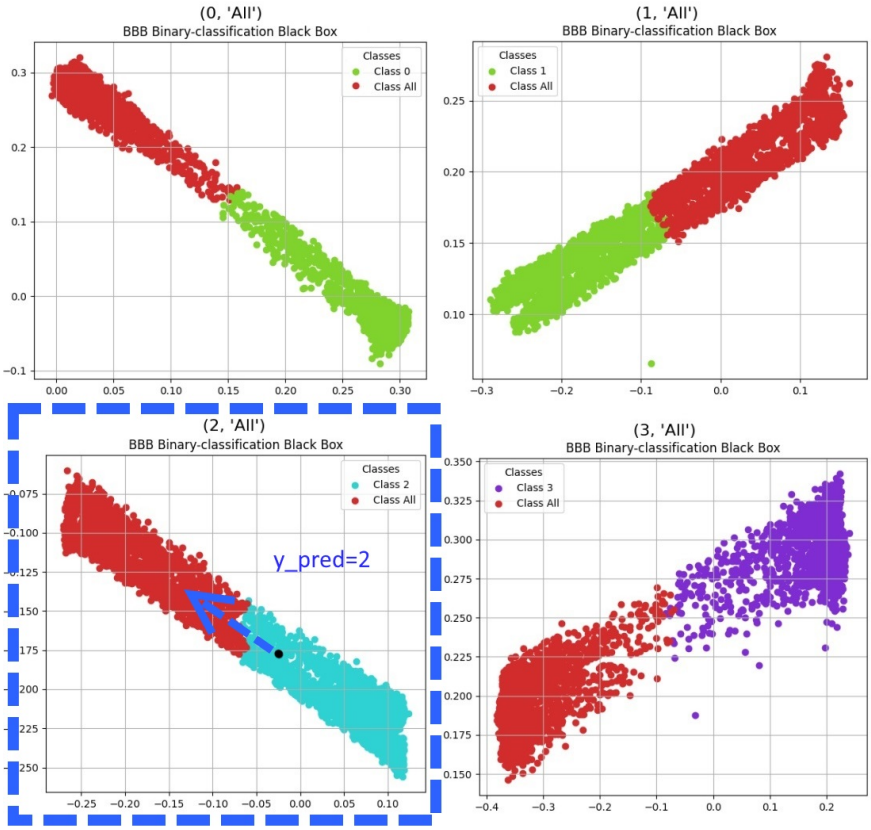

Figure 9: Example of 2D latent spaces for a 4-class classification in 1vAll mode, where $y\_pred = 2$. The only latent space required for counterfactual generation, in which class 2 is on the (one) side, is highlighted in blue.

Rest vs One (Allv1)

This approach operates symmetrically to the One-vs-Rest (1vAll) method. A separate binary classifier is trained for each class, where each classifier learns to distinguish one specific class from all others combined. If there are N classes, this approach also requires N binary classifiers.

- The `splitDataset` and `combinations` functions work identically as in One-vs-Rest.
- The function `isInvolved(y_pred, c_1, c_2)` returns true if y_{pred} matches only the target class c_2 (rest), which in this case serves as the starting class. This setup enables a counterfactual search in the opposite direction of One-vs-Rest, focusing on the perspective of the (rest) class while navigating toward the (one) class. The predicted label is represented within the latent space on the (rest) side, guiding the search along a specific direction within the latent space.

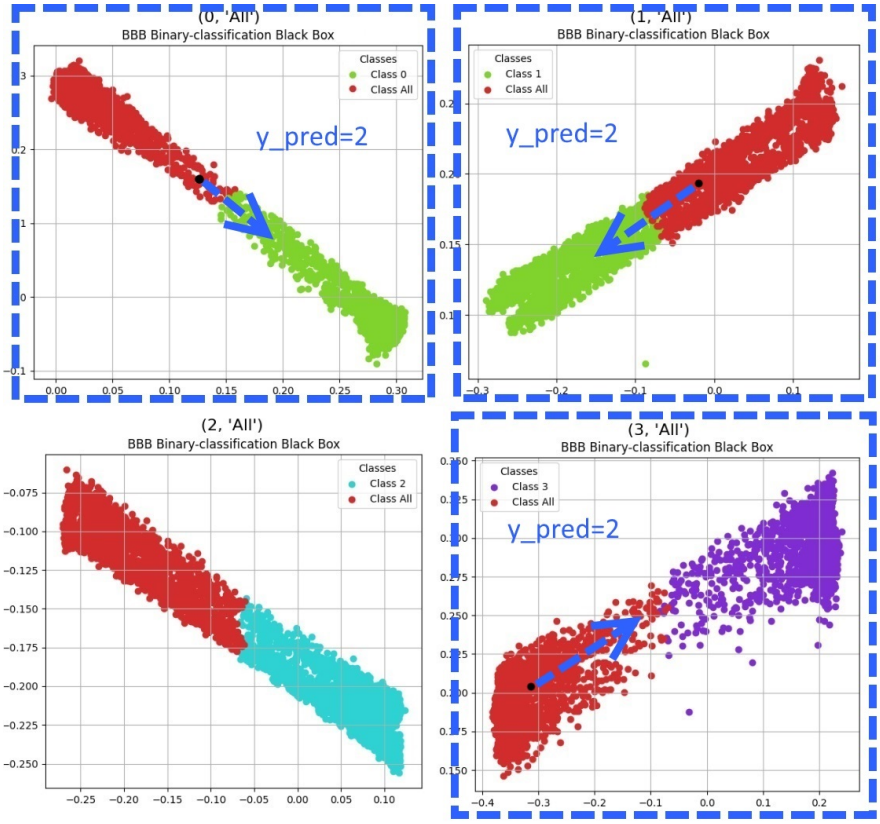

Figure 10: Example of 2D latent spaces for a 4-class classification in Allv1 mode, where $y\_pred = 2$. All latent spaces in which class 2 is on the (rest) side are considered for counterfactual research and are highlighted in blue.

3.2.4 Counterfactual Classification and Post-Processing

Unlike traditional XAI counterfactual approaches, this novel technique introduces a crucial distinction between different types of counterfactuals. In this work, counterfactuals are categorized based on their predicted label and the latent space in which they are found. During a specific counterfactual generation, a BBB may generate counterfactuals that are identified as such in a binarized latent space but receive a different label when evaluated by the original MBB. If the assigned label matches the original predicted label of the test instance, the counterfactual is considered **NOT ACCEPTABLE** and is discarded. Conversely, if its predicted class differs from that of the initial sample, it is always classified as **ACCEPTABLE**. Among the acceptable counterfactuals, a further distinction is made:

- If its predicted class is exactly the class searched in the specific latent space, the counterfactual is classified as **SPECIFIC**.
- Otherwise, it is labeled as **PLAUSIBLE**

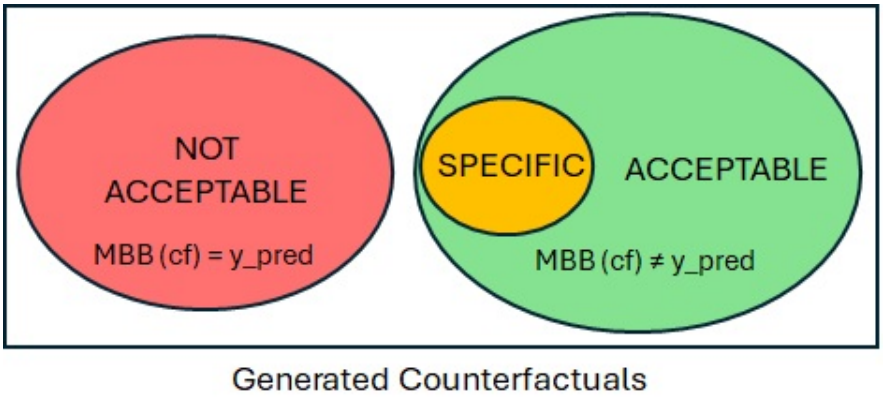

Figure 11: Different kind of counterfactuals.

These two types of counterfactuals provide different kinds of information. The specific ones indicate that they align precisely with the target class in the given latent space and suggest that the transformation applied to the original sample, following a specific prediction direction, has successfully captured the essential characteristics required for classification into the desired category. The others, instead, represent instances that, while distinct from the original sample, do not strictly belong to the intended target class. Instead, they demonstrate how an input can be modified to reach an alternative yet meaningful classification. These counterfactuals still provide valuable information about the flexibility of the model's decision-making process and the continuity of class representations in the latent space. Analyzing both types, detailed research can be done by taking into account aspects such as interpretability, minimality, robustness,

and plausibility. It is important to note that, in the specific 1vALL setting, *only acceptable counterfactuals* are considerable without further distinction. This is because the prediction direction, representing the "rest" class, corresponds to a group of samples drawn from different classes, each selected equally.
Considering these aspects, this approach offers the possibility to choose what kind of counterfactuals to research specific or acceptable (both specific and plausible), and for those modalities where it is allowed to consult multiple latent spaces, the specific class to look for, saving computational time in unnecessary generations. As the default CPILS method, it also extends the possibility to maintain some fixed features during the counterfactual search, preserving the **actionability** property.

Once the acceptable counterfactuals are generated and ready for use, the method introduces an additional, optional post-processing phase that includes:

- **Ranking** the counterfactuals based on:
  - A proximity metric (12)
  - Prediction probability
- **Grouping** counterfactuals by their assigned class
- **Limiting** the number of counterfactuals to a predefined value

For the user manual, software code, key components and additional details, please refer to Appendix (A).

4 Case study

This section provides an overview of the experimental context and the datasets used to test the proposed method with the aim of clarifying the context and the techniques for data collection, as well as the structure of the employed data. To ensure that this approach is applicable across various contexts and performs well in problems of different natures, two types of datasets have been used:

- **Benchmark datasets**, where their common ground truth allows evaluation and reproducibility of the experimentations allowing comparisons of methodologies in terms of efficacy, efficiency, robustness and facilitating the research progress.
- **K-EmoCon datasets**[26], a specific collection of data in the field of Emotional Intelligence with the target of the identification and recognition of the emotions thought physiological signals. This aligns with the primary objective of this thesis.

Both types of datasets are used to address multi-class classification problems and include data in a tabular format, featuring both numerical and categorical attributes. In particular, for the benchmark datasets, different levels of class imbalance are explored to better understand the behavior of the proposed method. The following paragraphs provide a detailed description of the information they contain and their composition.

4.1 Benchmark datasets

The following datasets were sourced from two of the most renowned and reliable repositories on the web: Kaggle[29] and the UCI Machine Learning Repository (University of California, Irvine)[30].

Body performance[31]
This dataset contains information about people's performance executing a physical exercise. The number of classes is equal to **4**, corresponding to different possible outcomes in performing the exercise. These data were originally collected by the Korea Sports Promotion Foundation, and as remarked in the download site, some post-processing and filtering have been done on the raw data. The dataset is evenly class **balanced**, and it is composed of a total of 13388 instances, 70% (9371) is used for training and 30% (4017) for testing. It is composed of 13 features (2 categorical and 11 numerical). These features are reported in the following table.

Feature	**Type**	**Description**
Age	Numerical	Participant's age, range: 20-64
Gender	Categorical	Biological sex of the participant: {F, M}.
Height_cm	Numerical	Height of the participant in centimeters.
Weight_kg	Numerical	Weight of the participant in kilograms.
Body fat %	Numerical	Percentage of body fat composition.
Diastolic	Numerical	Min blood pressure recorded (mmHg).
Systolic	Numerical	Max blood pressure recorded (mmHg).
GripForce	Numerical	Strength of hand grip (kg).
Sit and bend forward_cm	Numerical	Measure of flexibility in centimeters.
Sit-ups counts	Numerical	# of sit-ups performed in a given time.
Broad jump_cm	Numerical	Distance covered in this exercise (cm).
Class	Categorical	Performance classification: {A, B, C, D} (A: best, D: worst).

Table 1: Description of dataset features.

Thyroid[32]

This dataset contains information about patients with thyroid disease. Due to the sensitivity of the data, attribute names are anonymized. It consists of **3** classes, representing the following conditions: hyperthyroidism (0), hypothyroidism (1), and normal thyroid levels (2). The training set includes 3722 instances, while the testing set contains 3428 instances. Both the training and test sets have already been prepared and are available in the downloaded folder. There is a significant class **imbalance**, with the classes (0, 1, 2) distributed in the following proportions (2%, 5%, 93%) in both the training and test sets. The dataset includes 22 attributes: 1 categorical (the class), 6 numerical, and 15 binary (with values 0/1).

Cars[33]

This dataset contains information on car evaluations, including typical details and an associated range of price data. It consists of **4** classes representing evaluation levels: unacceptable, acceptable, good, and very good. The dataset exhibits significant class **imbalance**, with class distributions of 70%, 22%, 4%, and 4%, respectively. The total number of instances is 1728, with 70% (1209) used for training and 30% (519) for testing. The dataset includes 7 attributes, all of which are categorical, and one of them is the class. These features are reported in the following table.

Feature	**Type**	**Description**
Price	Categorical	Price category: {low, med, high, vhigh}.
Maintenance	Categorical	Maintenance cost category: {low, med, high, vhigh}.
Doors	Categorical	Number of doors: {2, 3, 4, 5more}.
Persons	Categorical	Passenger capacity: {2, 4, more}.
Lug_boot	Categorical	Luggage boot size: {small, med, big}.
Safety	Categorical	Safety rating: {low, med, high}.
Class	Categorical	Car evaluation class: {unacc, acc, vgood, good}.

Table 2: Description of the car evaluation dataset features.

Cars2

This dataset is identical to the "Cars" dataset, except that it has been pre-processed by combining the last two classes (good and very good) into a single class. Consequently, the number of classes is reduced to **3**. The class distribution now exhibits an imbalance of (70%, 22%, 8%) respectively

Wine[34]

The dataset contains information about various wines, categorized into two groups: red and white wines. The original class represents the evaluation of each wine, based on specific characteristics, with a score ranging from 1 to 10. To address the high level of class imbalance, the ratings are grouped into **3** classes: insufficient (scores lower than 6), sufficient (score equal to 6), and very good (scores greater than 6). This transformation reduces the **imbalance**, resulting in a distribution of (36.66%, 43.69%, 19.65%) across the three classes. The dataset consists of 6497 instances, with 70% (4547) allocated for training and 30% (1950) for testing. It contains 13 features, 2 of which are categorical and 11 are numerical. These features are:

Feature	**Type**	**Description**
Fixed acidity	Numerical	Contributes to wine taste.
Volatile acidity	Numerical	Relates to vinegar-like taste.
Citric acid	Numerical	Enhances citrus flavor.
Residual sugar	Numerical	Affects sweetness and taste.
Chlorides	Numerical	Indicates salt content and water quality.
Free sulfur dioxide	Numerical	Acts as an antioxidant, ensuring stability.
Total sulfur dioxide	Numerical	Fermentation byproduct contributing to stability.
Density	Numerical	Measures mass per unit volume, affecting consistency.
pH	Numerical	Represents acidity level.
Sulphates	Numerical	Helps control microbial growth.
Alcohol	Numerical	Ethanol content, determining wine strength.
Category	Categorical	Wine type: {red, white}.
Class	Categorical	Quality classification: {unsuff, suff, vgood}.

Table 3: Description of the wine dataset features.

Dataset Name	**# Classes**	**# Instances**	**# Train**	**# Test**	**# Attr.**	**# Num.**	**# Cat.**	**Status**
Body Performance	4	13,388	70%	30%	13	11	2	Balanced
Thyroid	3	7,150	52%	48%	22	6	16	High Imbalance
Cars	4	1,728	70%	30%	7	0	7	High-Med Imbalance
Cars2	3	1,728	70%	30%	7	0	7	Medium Imbalance
Wine	3	6,497	70%	30%	13	11	2	Medium Imbalance

Table 4: Overview of the benchmark datasets used in the experiments.

4.2 K-EmoCon datasets

These datasets aim to be used for emotion recognition. As explained in section (1), this subject is useful for several fields. Let the computers be able to identify emotions is not a trivial task because it is also complicated for humans to determine a correct emotional state. Measuring and quantifying emotions is inherently difficult, and acquiring accurate data further adds to the complexity.
Traditionally, emotions have been perceived as distinct psychological states expressed through facial expressions. However, research suggests otherwise, that facial expressions are often ambiguous, context-dependent, and misleading rather than universally distinct. A recent review highlights that facial expressions lack reliability, specificity, and generalizability, reinforcing prior studies on the contextual and individual variability of emotions. Most emotion datasets are created in controlled laboratory environments, where emotions are induced using selected stimuli. While this approach ensures experimenters have full control over data collection, it may not fully capture the nuances of real-world emotional expressions.

K-EmoCon is a multimodal dataset collected from 32 participants engaged in 16 paired debates on a social issue: the acceptance of Yemeni refugees on Jeju Island, South Korea. Prior to the debates, each participant received two articles via email, one supporting and one opposing the issue, to ensure familiarity with the topic. Debate pairs were formed randomly, considering participants' availability. The dataset includes physiological sensor data recorded using three commercially available wearable devices, audiovisual recordings of participants during the debates, and continuous emotion annotations. Differently from other existing datasets, this one takes into account emotions from 3 different perspectives: the subject himself, the debate partner, and an outside observer.

- The *subject* is the source who experiences emotions firsthand and produces self-annotations, particularly the "felt sense" of the emotions.
- The *partner* is the person who interacts with the subject, experiencing the subject's emotions second-hand; thus, he/she has contextual knowledge of the interaction that induced the subject's emotions and produces partner annotations based on that.
- The *external observers* are people who observe the subject's emotions without the exact contextual knowledge of the interaction that induced the emotions, producing external observer annotations.

In this way, it is possible to have a complete overview of the analyzed sentiment and different interesting researches can be conducted through these different points of view.

For the following experimentations, only the point of view of the subject directly involved in the debate is considered. The analysis of other points of view is left for further future examinations.

During a debate, participants wore a suite of wearable sensors, which included:

- *Empatica E4 Wristband* – Captured photoplethysmography (PPG), 3-axis acceleration, body temperature, and electrodermal activity (EDA). Heart rate and the inter-beat interval (IBI) were derived from Blood Volume Pulse (BVP) measured by a PPG sensor.
- *Polar H7 Bluetooth Heart Rate Sensor* – Detected heart rates using an electrocardiogram (ECG) sensor and was used to complement the PPG sensor in the E4, which is susceptible to motion.
- *NeuroSky MindWave Headset* – Collected electroencephalogram (EEG) signals via two dry sensor electrodes, one on the forehead (fp1 channel-10/20 system at the frontal lobe) and one on the left earlobe (reference).
- *LookNTell Head-Mounted Camera* – A camera attached at one end of a plastic circlet, worn on participants' heads to capture videos from a first-person POV.

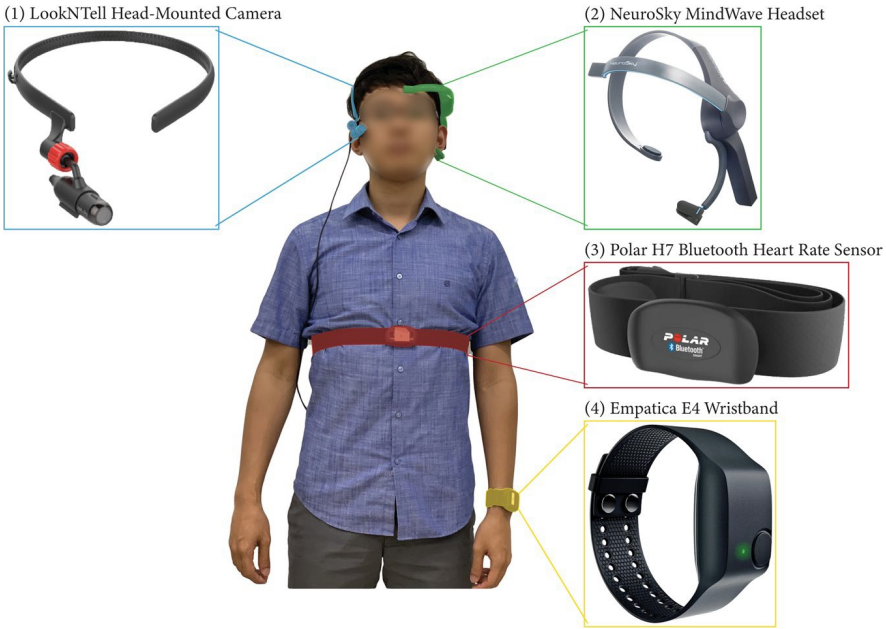

Figure 12: The devices used to collect physiological data

These datasets consist of 19 attributes, with only one being categorical, the class, which is directly annotated by the subject. The remaining attributes are collected from the enhanced devices. The original dataset attributes "seconds", "external_arousal",

"partner_arousal", "partner_valence", "self_arousal", "external_valence" are removed as they are not relevant when analyzing data from the perspective of an individual subject. The next table summarizes the effective attributes involved, what they represent, and from what device they are retrieved.

Attribute Name	**Attribute Description**	**Device**
x	Acceleration along the x-axis during arm movement	Empatica E4 Wristband
y	Acceleration along the y-axis during arm movement	
z	Acceleration along the z-axis during arm movement	
E4_BVP	Blood Volume Pulse, indicating blood flow	
E4_EDA	Skin impedance, indicating emotional response	
E4_IBI	Time interval between consecutive heartbeats	
E4_TEMP	Body temperature in °C	
E4_HR	Heart rate (bpm), obtained via ECG	Empatica E4 Wristband and Polar H7 Bluetooth
Attention	User's attention level (1-100)	NeuroSky MindWave Headset
delta	Delta brain wave activity (0.5 - 2.75 Hz)	
lowAlpha	Low Alpha brain wave activity (7.5 - 9.25 Hz)	
highAlpha	High Alpha brain wave activity (10 - 11.75 Hz)	
lowBeta	Low Beta brain wave activity (13 - 16.75 Hz)	
highBeta	High Beta brain wave activity (18 - 29.75 Hz)	
lowGamma	Low Gamma brain wave activity (31 - 39.75 Hz)	
middleGamma	Middle Gamma brain wave activity (41 - 49.75 Hz)	
theta	Theta brain wave activity (3.5 - 6.75 Hz)	
Meditation	Relaxation level (0-100)	
self_valence	The class, emotionality level of subject {1, 2, 3, 4, 5}	Self-annotated

Table 5: Description of the KEmoCon attributes.

Figure 13: Illustration of the data collection scenario

It is mandatory to explain that in literature emotion can be categorized with discrete values such as (happiness, disgust, fear and anger) or can be represented in a bi-dimensional space where on x-axis refers to the level of *valence* and the y-axis refers to the level of *arousal*. By combining these two coordinates it is possible to determine specific emotions. Valence quantifies the sentiment, with lower values indicating negative emotions and higher values indicating positive ones. Arousal, on the other hand, measures the intensity of the emotion. In this thesis work, only the level of valence is considered for the experimentations. The following image illustrates how changing a coordinate affects the emotion perceived by a subject.

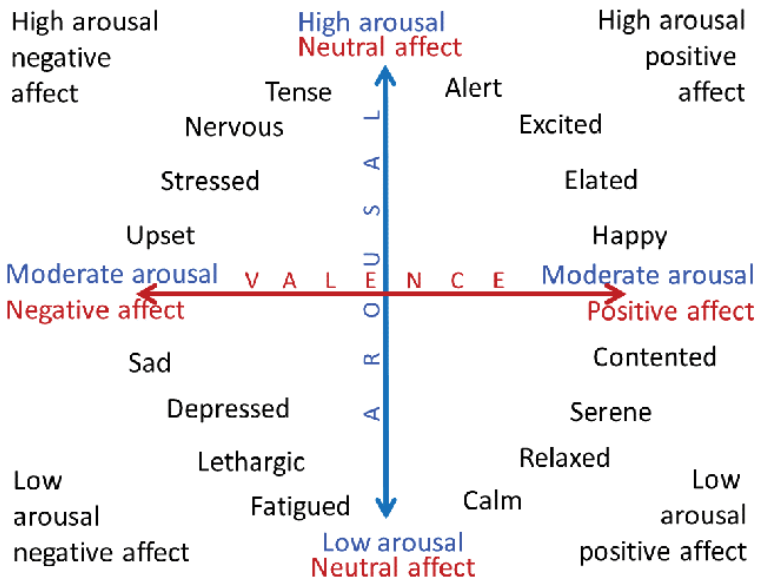

Figure 14: The Valence-Arousal plane illustrating the relationship between emotional valence and arousal levels[27].

In this next part are reported the subjects selected for the experimentation of the proposed novel approach. Each one has different characteristics in terms of dataset composition and number of valence levels (classes). Most of them are class imbalanced; however, differently from the used benchmark datasets, the number of data collected is sufficiently enough to apply an under-sampling technique to the majority classes in order to build a real **balanced situation** without the need to adopt smoothing techniques.

Subject 28

This dataset contains information related to the subject 28 that registers **3** (valence levels) of emotion during the experiment. It consists of 74390 instances, with a highly imbalanced distribution across valence levels: 27678, 31934, and 14778 instances for classes 2, 3, and 4, respectively. After the re-balancing, the resulting dataset comprises 44334 instances, with 70% (31033) allocated for training and 30% (13301) for testing.

Subject 5

This dataset contains information related to the Subject 5 that registers **4** (valence levels) of emotion during the experiment. It includes a total of 91511 instances, distributed unevenly across valence classes: 4416, 24268, 34639, and 28188 instances for classes 2, 3, 4, and 5, respectively. After the re-balancing, the final dataset consists of 17664 instances, with 70% (12364) allocated for training and 30% (5300) for testing.

Subject 17

This dataset contains information about Subject 17, who recorded **5** (valence levels) of emotion during the experiment. The dataset comprises a total of 65949 instances, with a moderately imbalanced distribution across valence levels (1-5): 6003, 8547, 25130, 4797, and 21472 instances, respectively. For this dataset, the re-balancing operation is computed considering only *1k instances* per valence level. The new dataset consists of 5000 instances, with 70% (3500) used for training and 30% (1500) reserved for testing.

Subject 27

This dataset contains information about Subject 27, who recorded **3** (valence level) of emotion during the experiment. The dataset comprises a total of 74886 instances, with a moderately imbalanced distribution across valence levels: 14058, 47052, 13776 for levels (2,3,4), respectively. For this dataset, the re-balancing operation is computed considering only *10k instances* per valence level. The new dataset consists of 30000 instances, with 70% (21000) used for training and 30% (9000) reserved for testing.

Subject 9

This dataset contains information about Subject 9, who recorded **4** (valence levels) of emotion during the experiment. The dataset comprises a total of 87503 instances, with a very strong imbalanced distribution across valence levels: 2905, 70274, 13595, 729 for levels (2,3,4,5), respectively. Under-sampling technique is applied to the majority classes, considering 700 instances per each class. As a result, the new dataset consists of 2800 instances, with 70% (1960) used for training and 30% (840) reserved for testing.

Dataset Name	**# Classes**	**# Instances**	**# Train**	**# Test**	**# Attr.**	**# Num.**	**# Cat.**	**Status**
			70%	30%				
Subject 28	3	44,334	31,033	13,301				
Subject 5	4	17,664	12,364	5,300				
Subject 17	5	5,000	3,500	1,500	19	18	1	Balanced
Subject 27	3	30,000	21,000	9,000				
Subject 9	4	2,800	1,960	840				

Table 6: Overview of the K-EmoCon datasets used in the experiments.

4.3 Pre-processing activities

In this paragraph are specified all the pre-processing techniques applied on all the aforementioned datasets before feeding the data in the proposed method.

- **Missing values removal**: all the rows containing empty, null, or missing values are removed at the cost of having a lower number of records in the set.
- **One-hot-encoding**: all categorical features are transformed into binary ones through one-hot encoding, where a new column is created for each category. A value of 1 indicates the presence of the category, while 0 indicates its absence. This process eliminates ordinality in the data, preventing the model from misinterpreting categorical values as a ranked sequence, which could lead to biased predictions. It also allows a larger compatibility of usage for most of the black box models. However, this transformation is applied carefully, as the CPILS method can inherently handle categorical attributes, provided their names follow a structured format such as (Attr_valueX) like Attr_Val1, Attr_Val2, or Attr_Val3. This ensures that CPILS correctly identifies them as categorical features and treats them as single modifiable attributes during feature perturbation.
- **Min-Max Normalization**: most of the presented datasets contain numerous features, each with a wide range of values. This introduces sparsity inside data and reduces the capability of a model to identify patterns in a correct way. The Min-Max Normalization technique is applied to all the numerical features in the datasets in order to guarantee that every feature contributes equally. This prevents features with larger magnitudes from dominating others while also enhancing the convergence of optimization-based algorithms and distance-based methods.
- **Renaming and Label encoding**: the target class attribute varies in name across different datasets. Once identified, it is renamed to simply *class* and repositioned as the last column in the dataset. This standardization ensures that the method consistently recognizes its name and location. Additionally, to ensure compatibility with most black-box models, the class labels are encoded into unique integers, assigning a distinct numerical value to each category.
- **Re-balancing**: the data splitting into training and test sets has already been discussed in the previous paragraph and varies depending on the specific case.

4.4 Black box models

The proposed method is model-agnostic, meaning it can be applied to any existing black-box model. To ensure a representative selection that spans the entire spectrum of artificial intelligence supervised classification models, the following black-box models are employed in the experiments.

- **XGB** (eXtreme Gradient Boosting) is an advanced machine learning algorithm in the category of ensemble methods, which combines multiple weak models (decision trees) sequentially to form a stronger model. The particularity of this approach consists in training each new tree to correct the errors made by the previous one. This process is called boosting. A specific set of hyperparameters is used:
  - 60 decision trees as number of estimators
  - Logarithmic loss as evaluation metric
  - Other parameters are kept as default values
- **RF** (Random Forest) is a powerful technique in machine learning that consists of a collection of decision trees that work together to make predictions through a voting mechanism. It takes different random parts of the dataset to train each tree, and then it combines the results by averaging them. It is in the category of the ensemble methods. A specific set of hyperparameters is used:
  - 100 decision trees as number of trees in the forest
  - Other parameters are kept as default values
- **MLP** (Multi-Layer Perceptron) is an artificial neural network that consists of fully connected dense layers that transform input data from one dimension to another. The purpose of an MLP is to model complex relationships between inputs and outputs, making it a powerful tool for various machine learning tasks. A specific set of hyperparameters is used:
  - Maximum number of iterations for training equal to 200
  - A single hidden layer of 100 hidden units
  - Relu as activation function
  - Adam as optimizer
  - Other parameters are kept as default values

- **SVM** (Support Vector Machine) is a different kind of supervised machine learning algorithm that aims to find the optimal hyperplane in an N-dimensional space to separate data points into different classes. While it may seem intuitive and easy to understand in lower-dimensional spaces, this becomes less clear in higher-dimensional spaces, particularly when a non-linear kernel function is used. A specific set of hyperparameters is used:
  - RBF (Radial Basis Function) as Kernel
  - Other parameters are kept as default values

Each method is implemented in the scikit-learn Python library and it is well documented at the following link [35]. For the objective of this thesis work, it is not necessary to find the best set of hyperparameters that maximize a multi-class black box model. The aim is to see how the proposed approach is able to explain as much as possible black box classifiers with different performances in different conditions. This also allows homogeneous comparisons among different tests and datasets. For this reason, the training procedure of those models involves a simple preliminary division of the original dataset into two main sets (train-test split), applying a stratification strategy to preserve the percentages of classes in the original set. The first set (training set) is used to "train" the black box model's ability to recognize a particular task, and the second set (test set) is used to evaluate the model's learned capabilities on unseen data. Once the model is trained, it can be practically used by feeding it new unseen samples. This phase is also known as "inference", in which it returns the label that best represents the class associated with that specific sample, based on the highest prediction probability. To ensure the reproducibility of the experiments and guarantee that the same sets of data are used to train and test the black-box models, a fixed random seed has been set (`seed = 42`).

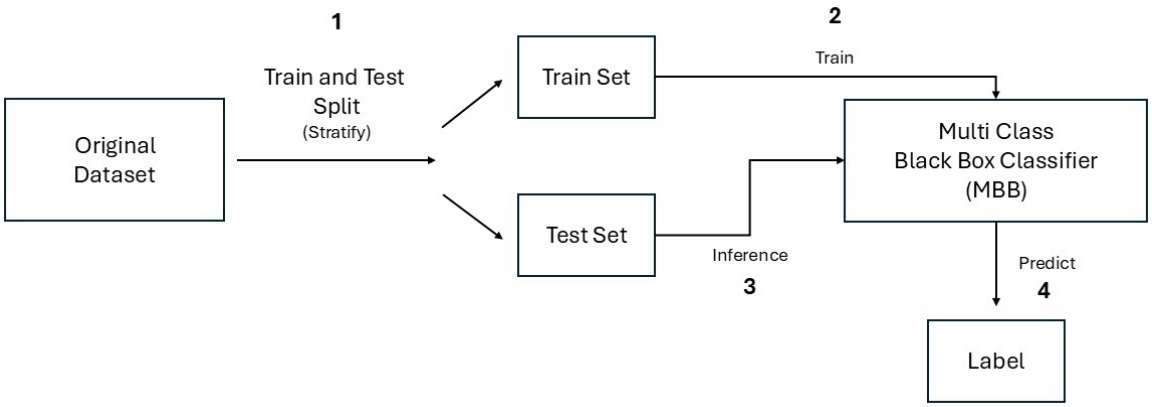

Figure 15: Illustration of the training and inference processes in a black-box classifier.

4.5 Feature Actionability

The described datasets contain a wide variety of features, but not all of them can be considered actionable. Feature actionability is an important property of the proposed method that allows specifying which features should remain fixed during the counterfactual search and which can be modified. This section explains and justifies the selection of features that are kept fixed in the experimentation phase in order to conduct a sensible and as realistic experimentation as possible. The following table provides a summary of the fixed features for each dataset, including their names, total count, and a brief motivation of their selection.

Dataset	**#Attr**	**Fixed Features (#)**	**Free Features**	**Motivation**
Body Perf.	13	Age, Gender, Height, Weight (4)	body fat_% diastolic systolic gripForce sit and bend forward_cm sit-ups broad jump_cm	Fixed features are intrinsic and cannot be modified. Free features can be improved through training (e.g., reducing body fat).
Thyroid	22	None (0)	Attr1 ... Attr22	Attribute names are anonymized, so fixing features is meaningless.
Cars/Cars2	7	Doors, Persons, Lug_boot (3)	Price Buying Price Maint Safety	Fixed features reflect user preferences. Free features depend on external factors (market, safety ratings, etc.).
Wine	13	category_red, Fixed acidity, Citric acid, Density, pH, Alcohol (6)	Volatile acidity Residual sugar Chlorides Free sulfur dioxide Total sulfur dioxide, Sulphates	Fixed features define a wine type. Free features can be adjusted to enhance wine quality.
KEmoCon	19	x, y, z (3)	The others (15)	x, y, z determine agitation but don't alter emotions. Changing them wouldn't meaningfully modify emotional states.

Table 7: Summary of fixed and free features for each dataset with motivations.

5 Experimental Results

The following chapter offers a comprehensive overview of the experimental phase of this study. The first section introduces the configurations and the metrics used to compare different modalities of the multi-class counterfactual generation proposed method (see paragraph 3.2.3), to assess their advantages and limitations. The next part explains the metrics used to specifically compare this approach to other state-of-the-art similar methods. This phase of the work consists of a series of reports, each focusing on specific aspects of the proposed method in various contexts, different datasets, black-box classifiers, and parameter configurations. Each experiment is presented in a separate paragraph, starting with an explanation of its purpose and specific objectives and following with a presentation of the experimental results in tables. Finally, it concludes with a discussion of the findings and observations.

5.1 Experimental Settings

The general experimental setup is based on running a specific function that takes as input a list of test instances for which counterfactuals need to be generated, along with a set of configurable parameters that enable flexibility in the proposed solutions. For each instance passed, the function calls the proper method to generate counterfactuals according to the parameters and modalities established. At the end, it produces 3 different files that are saved and stored separately on disk.

- **Test instance File**: contains all test instances used for counterfactual generation, each labeled with a unique ID.
- **Counterfactuals File**: stores all counterfactuals generated by the method, each assigned a specific ID.
- **Result File**: maps each test instance to its corresponding counterfactual(s), as the method can generate multiple counterfactuals per instance. This file also includes the original class label of the test instance and the predicted class of the counterfactual (as determined by both MBB and BBB).

These files allow to perform some post-processing operations such as computing specific metrics or performing detailed analyses to assess the quality of the results.

To make the experiments comparable, the following set of parameters have been used for the standard CPILS method in the various binary latent spaces.

Generation of latent space:

- Latent space dimension size fixed at 2 (except in specific cases where indicated).
- Early stopping parameter set to 10 to prevent overfitting.
- Other parameters retained in their default configuration.

Counterfactual generation

- Number of counterfactuals to be generated set to -1, indicating no limitation.
- Feature set to modify (variable per dataset), with 10 features sampled as τ.
- Other parameters (e.g., s - starting step, ϵ - step increment, max. number of search iterations in the prediction direction) kept at their default values.

The counterfactual research type is set to "acceptable", meaning that all the (plausible and specific) counterfactuals are included and evaluated in the experiments. This allows wider post-processing operations. To provide a comprehensive analysis of the proposed solution, non-acceptable counterfactuals are also recorded in these output files, even though they are not subjected to specific evaluation metrics. This information helps quantify the actual number of discardable counterfactuals, as a filtering operation is performed before their final use. For a detailed overview of the hardware architecture used in the experiments, see Appendix B.1.
The following two sections introduce the metrics used to evaluate the generated counterfactuals.

5.2 Evaluation Metrics for multi-class CPILS

As discussed in the paragraph 3.2.4, differently from other approaches, this method is able to return different kinds of counterfactuals. The following metrics aim to quantify that number, giving meaning to concepts like capability, minimality (proximity), plausibility, and monitoring the execution time.

Capability

The metric of capability is introduced to quantify the number of test instances that are able to generate any kind of acceptable counterfactual. It is important to note that this method (like others) may not always be able to generate at least one counterfactual due to various factors that could negatively impact the process, such as the dataset structure, the black-box model involved, or an implicit preference for generating specific counterfactuals. This metric is presented in the reports in the opposite form (NOT ABLE), indicating the number of instances that fail to generate any counterfactuals at all. For greater detail, small asterisks (n) are used to indicate the number of instances that fail to generate exactly n out of (num_classes $-$ 1) counter-classes.
In formula:

$$\text{Instances\_NOT\_ABLE} = \frac{\#\text{instances\_fail\_generation\_any\_cfs}}{\#\text{instances\_correctly\_classified\_by\_MBB}}$$

Consider a 4-class classification problem. If a test instance is assigned label 3 by the MBB and fails to generate counterfactuals for classes 0, 1, and 2, it falls within this defined metric. However, if the instance is unable to generate counterfactuals for only one specific class (e.g., class 1), it is marked in the report as (*1) but is not included in the overall metric calculation.

Amount of counterfactuals

This metric counts the total number of counterfactuals of any type (non-acceptable and acceptable) generated during the experimental procedure. Its purpose is to assess the model's performance by distinguishing between the levels of acceptability and specificity, as well as quantifying the number of plausible counterfactuals. For both specificity and acceptability, a percentage is provided in rounded brackets, referring to the total number of counterfactuals in the corresponding column.
Next, the definitions of the total number of counterfactuals, as well as the number of acceptable, specific, plausible, and not acceptable counterfactuals, are presented in formula. For convention, let the original target label assigned by the MBB to the original test instance be denoted as y_{pred}, a generic counterfactual be identified as cf and let $\hat{y}$ represent the exact searched arrival class in the binary latent space.

Total Counterfactuals

$$\text{Tot.Cf.} = \sum_{i=1}^{N} C_i$$

where N is the number of test instances and C_i represents the total number of counterfactuals (of any kind) generated for the i-th instance.

Total Acceptable Counterfactuals

$$\text{Tot.Cf.}_{\text{ACCEPTABLE}} = \sum_{i=1}^{N} C_{\text{acc},i}$$

where $C_{\text{acc},i}$ is the number of acceptable counterfactuals generated for the i-th instance, those that satisfy the condition:

$$\text{MBB}(cf) \neq y_{\text{pred}}$$

Total Specific Counterfactuals

$$\text{Tot.Cf.}_{\text{SPECIFIC}} = \sum_{i=1}^{N} C_{\text{spec},i}$$

where $C_{\text{spec},i}$ is the number of specific counterfactuals generated for the i-th instance, those that satisfy the condition:

$$\text{MBB}(cf) \neq y_{\text{pred}} \quad \text{and} \quad \text{MBB}(cf) = \hat{y}$$

Total Plausible Counterfactuals

$$\text{Tot.Cf.}_{\text{PLAUSIBLE}} = \text{Tot.Cf.}_{\text{ACCEPTABLE}} - \text{Tot.Cf.}_{\text{SPECIFIC}}$$

As defined in Section (3.2.4), the number of plausible counterfactuals is obtained either by difference or by considering all counterfactuals that satisfy the condition:

$$\text{MBB}(cf) \neq y_{\text{pred}} \quad \text{and} \quad \text{MBB}(cf) \neq \hat{y}$$

Total Not Acceptable Counterfactuals

$$\text{Cf.}_{\text{NOT\_ACCEPTABLE}} = \text{Tot.Cf.} - \text{Tot.Cf.}_{\text{ACCEPTABLE}}$$

The number of "not acceptable" (or discardable) counterfactuals is obtained by difference, comparing the total number of counterfactuals with the acceptable ones. Alternatively, it can be defined as all counterfactuals that satisfy the condition:

$$\mathrm{MBB}(cf) = y_{\mathrm{pred}}$$

Minimality

The metric of minimality is highly important because it expresses the distance between a counterfactual from its originated test instance. A lower value indicates a closer point, highlighting that the required change is local and limited. To express the concept of distance with tabular data, a measure similar to (6) is proposed to account for numerical and categorical attributes. For numerical instances, the classical Euclidean distance is used to measure the distance between two points in Euclidean space:

$$d_e\left(x, y\right) = \sqrt{\sum_{i=1}^{n} \left(y_i - x_i\right)^2} \tag{10}$$

where $x, y \in \mathbb{R}^n$ are the two points (a counterfactual and its corresponding generated instance) in the space.

For categorical instances, the Jaccard Distance Measure is used to compute the dissimilarity among two sets and it is defined as follows:

$$D_j\left(A, B\right) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{11}$$

where A and B are two sets of elements, $|A \cap B|$ represents the intersection, and $|A \cup B|$ represents the union. This distance measure ranges from 0 to 1, where 0 means the sets are identical, and 1 means they are completely disjoint. Finally, the final distance metric is obtained combining (10) + (11):

$$d\left(x, y\right) = \left(\frac{h}{m}\right) D_j\left(x, y\right) \; + \left(\frac{h-m}{m}\right) d_{\mathrm{e}}\left(x, y\right) \tag{12}$$

where h represents the number of categorical features, $h - m$ the number of numerical features, and m the total number of features. In these experiments, the average proximity distance to the Nth closest counterfactual is analyzed. For simplicity, the average distances are computed for the first, second, and third nearest counterfactuals. This helps to emphasize the relative distances and provides insight into how far apart the generated counterfactuals are.

Anomaly Score

The anomaly score is designed to evaluate how effectively the proposed method preserves outliers within the generated counterfactuals set. This score reflects how realistic the generated samples are with respect to the original set of instances. The closer this score is to the one computed on the original training set, the more reliable the generated results can be considered. To compute this metric, the **Isolation Forest**[36] machine learning algorithm is used. It is designed for anomaly detection, and it distinguishes anomalies in data by isolating observations through a process of random partitioning and isolation paths within isolation trees. This process is responsible for creating partitions or "trees" that aim to separate anomalies from normal observations. Typically, anomalies are fewer in number and require fewer splits to be isolated, making them easier to detect. As reported by the official paper[28], this score is computed according to this formula:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

where:

- $s(x, n)$ is the anomaly score of a single instance x in a dataset of size n.
- $h(x)$ is the path length of x, the number of splits required to isolate x in the tree.
- $E(h(x))$ is the "expected" path length, representing the average path length over a collection of isolation trees.
- $c(n)$ is the path length normalization, representing the average path length of unsuccessful searches in binary tree searches:

  $$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

  where $H(n)$ is the harmonic number approximated by $H(n) \approx \ln(n)+0.5772156649$.

The Average Isolation Score computed on a generic set of N instances is defined as:

$$\text{Avg\_Isolation\_Score} = \frac{1}{N}\sum_{i=1}^{N} s(x_i, n)$$

In sklearn's implementation, the anomaly scores are the opposite of the anomaly score defined in the original paper that range from [0,1]. The result of this implementation is subtracted by a constant of 0.5, redefining the output range into [-0.5, 0.5]. This makes the interpretation like, the lower the value, the more abnormal. Negative scores represent outliers, positive scores represent inliers.

Execution Time

This metric simply measures the execution time of the script for generating counterfactuals based on the previously described execution conditions. The duration may vary depending on the number of latent spaces the method needs to use in counterfactuals research. The *tdqm* Python library is employed to monitor the test advancements and to report the final execution time.

5.3 Evaluation Metrics for state-of-the-art comparison

To ensure a fair comparison with state-of-the-art counterfactual generation methods, it is necessary to introduce generic metrics that are independent of any specific context. The goal is to assess the method's performance in terms of proximity, plausibility, robustness, diversity, and execution time. While some of these concepts were previously defined 5.2, they are now re-proposed in a different perspective.

It is important to note that in the following formulas, $\hat{x}$ is used to specify the original test instance associated with the produced counterfactuals. Note that multiple counterfactuals can be associated with the same test instance due to the ability to generate multiple counterfactuals. The formula used to compute proximity is (12) and is denoted as d_{mix}. X_C represents the entire set of specific counterfactuals generated.

Proximity

The objective is to evaluate the distance of generated counterfactuals to their original sample instance and quantify the number of feature changes. It's the same concept of minimality already discussed but now applied to the entire counterfactual set X_C. For this concept, two distinct metrics are proposed.

- **Ddist**: compute the average distance from counterfactuals to their original sample instance. For each counterfactual in the set, compute its distance from the corresponding test sample $\hat{x}$, then calculate the average distance across all counterfactuals in the set.

$$d_{\text{dist}} = \frac{1}{|X_C|} \sum_{x \in X_C} d_{\text{mix}}(x, \hat{x})$$

- **Dcount**: compute the average number of features that change among all the counterfactuals. For each counterfactual, compute how many features change compared to its test sample instance, then compute the average.

$$d_{\text{count}} = \frac{1}{|X_C|} \sum_{x \in X_C} \sum_{i=1}^{n} \nVdash_{[x_i \neq \hat{x}_i]}$$

  where $\nVdash_{[x_i \neq \hat{x}_i]} = 1$ if the i-th feature of counterfactual x is different from the i-th feature of its related test sample, and n is the number of features.

Plausibility

The purpose is to measure how faithfully the generated counterfactuals represent the information according to the initial training set. Two different metrics are taken into account to better represent this concept.

- **IF (Isolation Forest)**: the previously well-described metric is still used to assess how well outliers are preserved or not. The score ranges from -0.5 to 0.5, where negative values indicate that the target is significantly more outlier-like than faithfully representative. The closer this score is to the one computed on the original training set, the more representative it is.

- **Dimpl**: this metric computes the plausibility in terms of implausibility of the generated counterfactuals relative to their closest instance in the *training set*. For each counterfactual, compute the distance from its nearest training sample, then calculate the average of these distances.

$$d_{\text{impl}} = \frac{1}{|X_C|} \sum_{x \in X_C} \min_{u \in X_{\text{train}}} d_{\text{mix}}(x, u)$$

  The closer an instance is to its corresponding training sample, the lower this value is, indicating that the set of generated counterfactuals is more truthful.

Robustness

The objective is to assess how well the generated counterfactuals are stable in presence of noise and how well they represent counterfactuals in a denser region, avoiding to be confused by adversarial samples. Two measures are introduced, where the lowest value indicates a more robust set of counterfactuals.

- **Drob1**: it is a Monte Carlo perturbation method that introduces small variations in continuous features following a random normal distribution (mean = 0, standard deviation = 0.01), useful to defend against adversarial attacks. This formula calculates the mean absolute deviation (MAD) from the original prediction. It directly quantifies how much the predictions change due to small perturbations, making it easier to interpret the impact of noise on the counterfactual instance.

$$d_{\text{rob1}} = \frac{1}{|X_C|} \sum_{x \in X_C} \frac{1}{|N|} \sum_{i=1}^{N} |y_b(x_i') - y_b(x)|$$

  where x_i' represents a generic perturbed instance generated by adding Gaussian noise, $x_i' = x + \epsilon, \quad \epsilon \sim \mathcal{N}(0, 0.01^2)$, $N = 1000$ artificially generated instances by perturbation, and $y_b(\bullet)$ is the label predicted by the black-box classifier.

- **Drob2:** it computes how robust the counterfactuals are in relation to the K-nearest training samples. Differently from Drob1, this measure gives an assurance that the instance used to compare the counterfactuals exists and is one of the nearest to it. For each counterfactual, locate the N-nearest-neighbour instances in the original training set with $N = 10$, and count the number of instances that are labeled differently by the black box from the target counterfactual, then calculate the average of these values for all the counterfactuals.

$$d_{\text{rob2}} = \frac{1}{|X_C|} \sum_{x \in X_C} \frac{1}{|N|} \sum_{u \in N(x)} \nVdash [y_b(u) \neq y_b(x)]$$

where $\nVdash = 1$ if the counterfactual has a different predicted label than its neighbor instance u.

Diversity

This metric evaluates the diversity of generated counterfactuals by measuring their distance within the set and how many changes are present in terms of features. Two measures are introduced, where a higher value indicates a greater ability to generate diverse counterfactuals.

- **Ddiv1**: for each pair of counterfactuals in the set, compute their distance and then compute the average of these results.

$$d_{\text{div1}} = \frac{1}{|X_C|^2} \sum_{x \in X_C} \sum_{y \in X_C} d_{\text{mix}}(x, y)$$

where (x, y) is a specific pair of counterfactuals in the set.

- **Ddiv2:** for each pair of counterfactuals in the set, calculate the number of feature changes between them, then compute the average of these results.

$$d_{\text{div2}} = \frac{1}{|X_C|^2} \sum_{x \in X_C} \sum_{y \in X_C} \frac{1}{n} \sum_{i=1}^{n} \nVdash [x_i \neq y_i]$$

where (x, y) is a specific pair of counterfactuals in the set, where $\nVdash = 1$ if counterfactual x-feature is different from counterfactual y-feature, and n is the number of features.

5.4 Experimentation Reports

In the following paragraph are presented the reports carried out in order to understand the conclusions of this thesis work. A guided analysis of these reports is done with a particular focus on the meaning of each report, on how it is carried out operationally, and on the results obtained.

5.4.1 Report 1: Multi-Class CPILS - different working modalities

This report consists in experimenting with the different working modalities of the multi-class CPILS method (1v1, 1vAll, Allv1) with the purpose of analyzing the generated counterfactuals in terms of acceptability, specificity, plausibility, and minimality. In this experiment, only the Body Performance benchmark dataset is considered with the XGB as multi-class black box to be explained. In this specific report, the number of NOT acceptable counterfactuals is also proposed to highlight possible limitations of some modalities. Additionally, heatmaps are employed to understand if there are some preferences in the generation of specific counterfactual classes or not.

To better understand the impact of the instance sampling in the modalities (1vAll) and (Allv1), two additional modalities are proposed in this experimentation called respectively (1vAll_Spec) and (Allv1_Spec). They both refine the selection process by choosing instances from the ALL side that are closest to the samples labelled as belonging to class (targeted on 1 side), rather than relying on random sampling as in the original solution.

A total of 1000 test instances are selected for generating counterfactuals using a stratified sampling technique to ensure an equal representation of each class. For a clearer analysis, only counterfactuals derived from instances that are correctly classified by the black-box model are considered in the results.

```
Total test instances: 1000
Class labels: [0 1 2 3]
 250 instances of class 0
 250 instances of class 1
 250 instances of class 2
 250 instances of class 3
Total number of attributes: 12
Max number of changeble attributes: 7
Attributes that have been left fixed: ['age' 'gender_F' 'gender_M' 'height_cm' 'weight_kg']

Total correct classifications: 738 -- Accuracy: 73.80 %
Total miss_classifications: 262 / 1000 26.20 %
Missclassification per class

(Original Class -> Rows, MBB Predicted Class -> Columns)
         Class 0  Class 1  Class 2  Class 3   Total    Perc
Class 0      0.0     31.0      1.0      0.0    32.0   3.20%
Class 1     60.0      0.0     32.0     13.0   105.0  10.50%
Class 2     16.0     49.0      0.0     16.0    81.0   8.10%
Class 3      6.0     12.0     26.0      0.0    44.0   4.40%
Total       82.0     92.0     59.0     29.0   262.0  26.20%
Perc       8.20%    9.20%    5.90%    2.90%  26.20%      0

Test instances considered for following statistic: 738
 218 instances of class 0
 145 instances of class 1
 169 instances of class 2
 206 instances of class 3
```

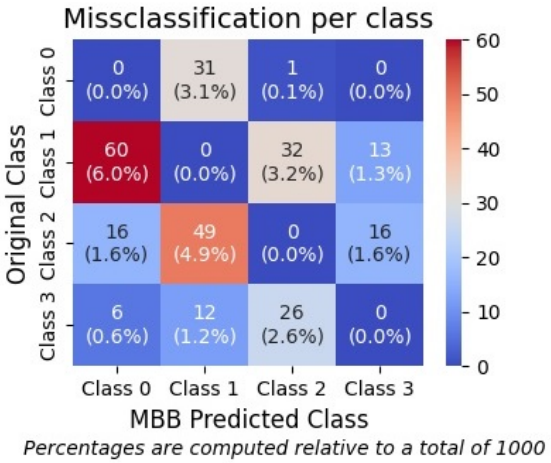

Figure 16: XGB Multi-class misclassification distribution by class

Method	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. NOT ACCEPTABLE	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC	Tot. cf. PLAUSIBLE
1v1	2/738 (*148) (**32)	8126 Avg. 11.04	1294 (15,92%)	**6832 (84.08%)** **Avg. 9.26**	2877 (35.40%) Avg. 3.90	3955 (48.67%) Avg. 5.36
1vALL	**0/738**	3335 Avg. 4.52	533 (15,98%)	2802 **(84.02%)** Avg. 3.80	-	2802 **(84.02%)** Avg. 3.80
1vALL_SPEC	0/738 (*59)	**2677** Avg. 3.63	428 (15.99%)	2249 (84.01%) Avg. 3.05	-	2249 (84.01%) Avg. 3.05
ALLv1	0/738 (*34)	11276 Avg. 15.28	4622 (40.99%)	6654 (59.01%) Avg. 9.59	**5493 (48.71%)** **Avg. 7.44**	1161 (10.30%) Avg. 2.15
Allv1_SPEC	0/738 (*124)	**11383** **Avg. 15.42**	5941 **(52.19%)**	5442 (47.81%) Avg. 7.85	3833 (33.67%) Avg. 5.19	1609 (14.14%) Avg. 2.66

Table 8: Report 1 overview: analysis of Capability, Counterfactual quantity, Specificity, and Plausibility across different working modalities.
* Total instances unable to generate a counterfactual for exactly 1 out of 3 counter-classes
** Total instances unable to generate a counterfactual for exactly 2 out of 3 counter-classes

On the next page are reported the heatmaps highlighting which are the counterfactual classes that most fail to generate a specific (desired) class for each modality and beside what are the most counterfactual classes produced differentiating from plausible and specific.

1v1

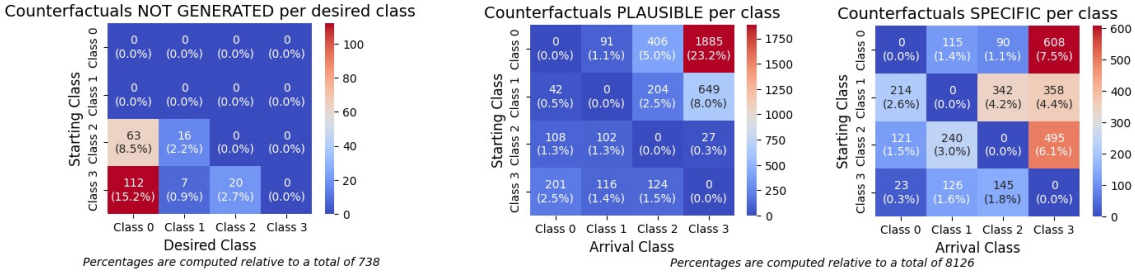

Figure 17: Heatmaps for 1v1 counterfactual generation mode.

1vAll and 1vAll_Spec

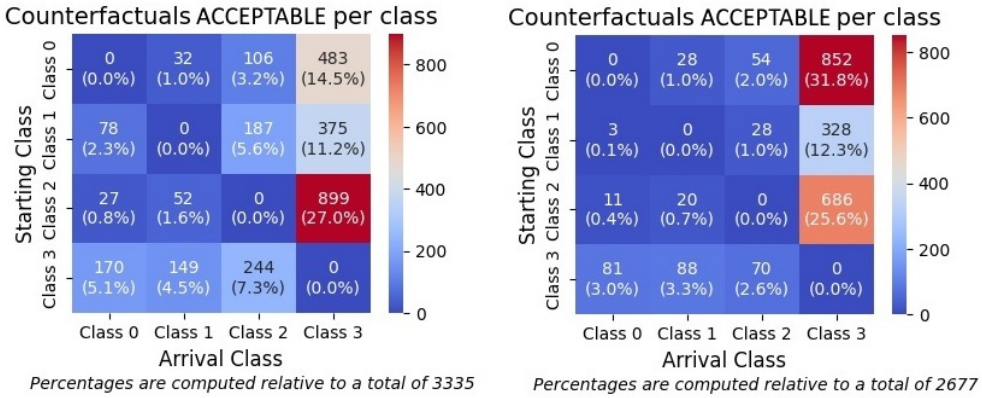

Figure 18: Heatmaps for 1vAll (on the left) and 1vAll_Spec (on the right) counterfactual generation modes. As discussed in 3.2.4, in these modalities, there is no distinction for the counterfactual category.

Allv1 and Allv1_Spec

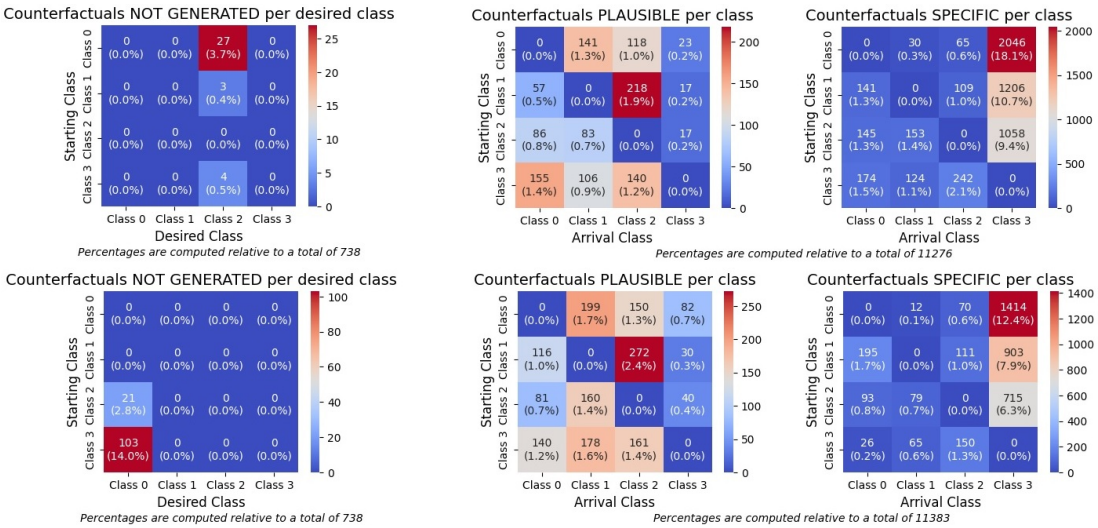

Figure 19: Heatmaps for Allv1 (on the top) and Allv1_Spec (on the bottom) counterfactual generation modes.

	PLAUSIBLE			SPECIFIC		
	Avg. Proximity Distance from the Nth closest cf.			Avg. Proximity Distance from the Nth closest cf.		
Method	1st	2nd	3rd	1st	2nd	3rd
1v1	57.61 ± 58.26	65.61 ± 61.89	68.76 ± 51.59	**38.97 ± 36.94**	**44.45 ± 35.38**	64.23 ± 47.03
1vALL	35.66 ± 20.10	60.51 ± 38.66	100.66 ± 53.51	-	-	-
1vALL_SPEC	**29.06 ± 23.43**	**45.30 ± 25.28**	**62.26 ± 32.39**	-	-	-
ALLv1	**96.55 ± 61.24**	143.77 ± 56.69	158.14 ± 47.53	44.90 ± 37.15	56.23 ± 41.47	57.21 ± 36.94
Allv1_SPEC	**78.68 ± 58.24**	123.40 ± 62.66	142.74 ± 53.20	43.54 ± 32.04	48.89 ± 27.82	**52.01 ± 25.85**

Table 9: Report 1 overview: analysis of Minimality based on average proximity distance to the first, second, and third nearest counterfactuals across all modalities

Considerations

The results are in general quite good. In most of the cases it is possible to generate a considerable number of counterfactuals.

- In Table 8, the **1VALL** modality is the **best** one in terms of **generation capability**, because there is no case where it is not able to generate at least 1 of any classes. The 1v1 mode is one of the worse cases, because for some class of counterfactuals some instance is not able to generate counterfactuals, but just in 2 instances is not possible to generate none (for all the 3 cases). The very negative case is the 1VAll_Spec because 59 instances are not able to generate any counterfactual, in the only latent space it can look.

- Regarding the generated number of counterfactuals, it's obvious that **1v1, Allv1** and **AllSpecV1 are advantaged** because they look each time in 3 different latent spaces, respect to 1vAll. The proportion is about **3.5**.

- The negative aspect of the last point is that although **Allv1 and AllSpecV1** have a significative higher number of generated counterfactuals the **40%** and **50%** needs to be **discarded**; this does not happen in the other 3 cases where the percentage is about 16%.

- On the other hand, **1v1, 1vAll and 1vAll_Spec** have important values of **acceptability, about 84%**. In particular, the **1vALL approach** is a good choice when a **lightweight solution** is preferred, even at the cost of generating fewer counterfactuals. On the other hand, if specific counterfactuals are the focus, the 1v1 approach is an excellent option. It offers a high percentage of both specific and plausible counterfactuals, achieving the best balance performance among all the methods between plausibility and specificity at the cost of more overhead due to training all the combinations of latent spaces.

- AllV1 and AllSpecV1 achieve comparable results in specificity, less in plausibility, this also at cost of a huge counterfactual to be discarded.

- From a point of view of **minimality**, it is evident from Table 9 that the introduced **SPEC version performs better** in all the cases with respect to the standard versions. Considering only plausible counterfactuals is important to highlight that the 1vALL version is the one with the best results because it follows a specific counterfactual direction (direct direction), independently of the class of counterfactuals. It detects better the minimal changes necessary. Contrary to the opposite version that instead starts from an agglomeration of samples with various classes, the values of proximity are quite worse. The 1v1 version instead seems to be a compromise between the two approaches. In the specific context, it is the best option. In general, **specific counterfactuals** result to be **closer** to the corresponding **testing instance** compared to plausibles.

- However, without considering minimality, there is **no significant advantage** in **using the SPEC mode**, as utilizing the nearest points does not show noticeable improvement compared to the classic mode.

  - the number of instances that are not able to generate any counterfactuals increases in both cases.
  - less counterfactuals acceptable in percentage
  - it does not take any improvement in the number of specifical class generated (combination of starting, arrival classes)

- Looking at the heatmaps, it is quite evident in all the cases that it is **easier** to **generate** a **counterfactual** of **class 3**. This is not a good thing because it influences all the results, but it also highlights that it depends on the specific dataset and the specific black box capabilities in classification. To be a good and even situation, all the percentages should be equally disposed; instead, in this case, the classes '0' and '1' seem the least supported.

5.4.2 Report 2: Multi-Class CPILS - Benchmark Datasets

The goal of this experimentation is to test the performances of the multi-class CPILS, in terms of counterfactuals generation capability, acceptability, specificity, isolation score, minimality and execution time on different conditions: on Benchmark datasets (4.1) and black box models (4.4). In this report, only the 1v1 and 1vAll modalities are considered as they provided in the previous analysis a significant lower percentage of discarded counterfactuals. This report follows the same interpretation modalities as the previous one with the addition of the anomaly score metric and execution time. The results that perform better under specific conditions are highlighted in black. A total of 100 test instances are selected for generating counterfactuals using a stratified sampling technique to ensure an equal representation of each class. Only the counterfactuals obtained from instances correctly classified are considered in the results.

Mode 1v1: capability, acceptability and specificity

Dataset	BB Classifier	Instances NOT ABLE	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC
	XGB	0/69 (*6)	872	723 (82.91%)	399 (45.76%)
Body Performance	RF	**0/74 (*4)**	780	631 (80.90%)	343 (43.97%)
	MLP	0/64	1244	911 (73.23%)	448 (36.01%)
	SVM	0/65 (*2)	1208	**1175 (97.27%)**	**673 (55.71%)**
	XGB	0/96 (*1)	933	907 **(97.21%)**	849 **(91.00%)**
Thyroid	RF	**0/100 (*1)**	1345	1066 (79.26%)	829 (61.64%)
	MLP	0/76	**2353**	**1827** (77.65%)	**1121** (47.64%)
	SVM	0/50	1205	737 (61.16%)	196 (16.27%)
	XGB	19/94 (*20) (**7)	**793**	**789** (99.50%)	**483** (60.91%)
Cars	**RF**	18/88 (*21) (**10)	644	642 **(99.69%)**	446 **(69.25%)**
	MLP	19/94 (*17) (**2)	780	773 (99.10%)	445 (57.05%)
	SVM	**18/91 (*33) (**1)**	612	595 (97.22%)	392 (64.05%)
	XGB	11/100 (*1)	721	721 **(100.00%)**	518 (71.84%)
Cars2	RF	11/97 (*8)	771	769 (99.74%)	482 (62.52%)
	MLP	11/97 (*1)	**904**	**896** (99.12%)	**679 (75.11%)**
	SVM	**10/98 (*1)**	793	784 (98.87%)	536 (67.59%)
	XGB	**0/69 (*10)**	462	358 (77.49%)	253 (54.76%)
Wine	RF	0/71 (*18)	366	239 (65.30%)	165 (45.08%)
	MLP	0/53 (*3)	**483**	**377** (78.05%)	**281** (58.18%)
	SVM	4/51 (*33)	263	259 **(98.48%)**	231 **(87.83%)**

Table 10: Report 2 overview: analysis of Capability, Counterfactual quantity, Specificity in 1v1 Mode on Benchmark datasets. * Total instances unable to generate a counterfactual for exactly 1 out of 3 counter-classes. ** Total instances unable to generate a counterfactual for exactly 2 out of 3 counter-classes. In Car dataset, 5 instances of class 2 and 6 instances of class 3, used only for generating counterfactuals, are created via SMOTE technique due to address data scarcity.

Mode 1vAll: capability, acceptability and specificity

Dataset	BB Classifier	Instances NOT ABLE	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. DETAILS			
					Class 0	Class 1	Class 2	Class 3
Body Performance	**XGB**	9/69	354	312 **(88.14%)**	18 (0.56 ± 0.20)	21 (0.53 ± 0.23)	43 (0.64 ± 0.20)	**230 (0.48 ± 0.20)**
	RF	9/72	**477**	**414** (86.79%)	10 (0.61 ± 0.17)	47 (0.60 ± 0.24)	51 (0.58 ± 0.17)	**306 (0.49 ± 0.20)**
	MLP	**0/64**	362	270 (74.59%)	34 (0.59 ± 0.16)	21 **(0.37 ± 0.11)**	58 (0.52 ± 0.14)	**157** (0.44 ± 0.19)
	SVM	2/65	312	257 (82.37%)	38 (0.62 ± 0.18)	21 (0.40 ± 0.11)	23 **(0.39 ± 0.11)**	**175** (0.48 ± 0.14)
Thyroid	XGB	15/96	234	234 **(100.00%)**	33 **(0.15 ± 0.03)**	45 (0.23 ± 0.19)	**156** (0.76 ± 0.41)	-
	RF	**0/100**	473	463 (97.89%)	93 (0.73 ± 0.46)	63 (0.72 ± 0.38)	**307 (0.70 ± 0.44)**	-
	MLP	0/76	**2088**	**1029** (49.28%)	341 **(0.83 ± 0.42)**	308 (1.19 ± 0.38)	**380** (1.10 ± 0.44)	-
	SVM	0/50	1159	814 (70.23%)	35 (1.10 ± 0.19)	8 **(0.46 ± 0.25)**	**771** (1.26 ± 0.26)	-
Cars	XGB	19/94	310	309 (99.68%)	92 (0.32 ± 0.08)	146 (0.37 ± 0.10)	42 (0.34 ± 0.10)	29 (0.38 ± 0.11)
	RF	18/88	329	329 (99.40%)	128 (0.40 ± 0.11)	127 (0.38 ± 0.12)	46 (0.37 ± 0.11)	28 (0.39 ± 0.11)
	MLP	**18/94**	**432**	**430 (99.54%)**	**183** (0.42 ± 0.11)	157 (0.36 ± 0.10)	37 **(0.31 ± 0.08)**	53 (0.43 ± 0.10)
	SVM	21/91	225	196 (87.11%)	15 (0.47 ± 0.08)	**107** (0.35 ± 0.10)	44 **(0.34 ± 0.09)**	30 (0.43 ± 0.12)
Cars2	XGB	11/100	409	409 **(100.00%)**	107 **(0.33 ± 0.10)**	**216** (0.34 ± 0.09)	86 (0.39 ± 0.12)	-
	RF	10/97	**469**	**465** (99.15%)	136 (0.38 ± 0.11)	**215 (0.32 ± 0.08)**	114 (0.43 ± 0.12)	-
	MLP	10/97	397	396 (99.75%)	127 (0.40 ± 0.11)	**194 (0.34 ± 0.09)**	75 **(0.37 ± 0.10)**	-
	SVM	**10/98**	437	374 (85.58%)	47 (0.46 ± 0.09)	**248 (0.36 ± 0.11)**	79 (0.38 ± 0.11)	-
Wine	**XGB**	**0/69**	**320**	**241 (75.31%)**	**120** (0.73 ± 0.30)	99 (0.64 ± 0.30)	22 **(0.62 ± 0.49)**	-
	RF	5/71	246	153 (62.20%)	68 **(0.42 ± 0.29)**	**78** (0.44 ± 0.24)	7 (0.80 ± 0.29)	-
	MLP	0/53	233	171 (73.39%)	65 (0.70 ± 0.25)	**95** (0.73 ± 0.27)	11 **(0.60 ± 0.36)**	-
	SVM	0/51	218	147 (67.43%)	**84** (0.82 ± 0.26)	63 **(0.66 ± 0.20)**	0 (-)	-

Table 11: Report 2 overview: analysis of Capability, Counterfactual quantity, Specificity in 1vAll Mode on Benchmark datasets. In this mode, it is not possible to retrieve specific counterfactuals, so a detailed view of the generated results is proposed considering the count number and the (average proximity distance ± standard deviation).

Modalities 1v1 and 1vAll: minimality

Method	Dataset	BB Classifier	PLAUSIBLE Avg. Proximity Distance from the Nth closest cf.			SPECIFIC Avg. Proximity Distance from the Nth closest cf.		
			1st	2nd	3rd	1st	2nd	3rd
1v1	Body Perf.	XGB	0.41 ± 0.30	0.47 ± 0.35	0.56 ± 0.27	0.23 ± 0.20	0.44 ± 0.20	0.59 ± 0.20
		RF	0.40 ± 0.31	0.53 ± 0.30	0.54 ± 0.30	0.26 ± 0.26	0.48 ± 0.26	0.53 ± 0.21
		MLP	0.33 ± 0.21	0.48 ± 0.20	**0.52 ± 0.18**	0.22 ± 0.19	0.32 ± 0.19	0.37 ± 0.16
		SVM	**0.33 ± 0.17**	**0.44 ± 0.14**	0.53 ± 0.16	**0.16 ± 0.11**	**0.28 ± 0.16**	**0.35 ± 0.18**
	Thyroid	**XGB**	0.88 ± 0.24	1.14 ± 0.08	-	**0.24 ± 0.31**	**0.38 ± 0.37**	**0.52 ± 0.39**
		RF	0.94 ± 0.28	1.03 ± 0.37	1.18 ± 0.19	0.15 ± 0.22	0.54 ± 0.37	0.88 ± 0.28
		MLP	0.71 ± 0.34	**0.73 ± 0.29**	**0.86 ± 0.33**	0.25 ± 0.37	0.46 ± 0.41	0.70 ± 0.47
		SVM	**0.56 ± 0.39**	0.77 ± 0.43	0.92 ± 0.44	0.86 ± 0.24	0.73 ± 0.24	0.86 ± 0.23
	Cars	XGB	0.42 ± 0.14	0.44 ± 0.14	**0.29 ± 0.00**	**0.29 ± 0.00**	0.30 ± 0.06	**0.30 ± 0.06**
		RF	**0.37 ± 0.11**	0.57 ± 0.09	0.29 ± 0.00	0.29 ± 0.00	**0.30 ± 0.05**	0.31 ± 0.07
		MLP	0.45 ± 0.13	0.45 ± 0.11	0.30 ± 0.07	0.36 ± 0.11	0.30 ± 0.05	0.30 ± 0.07
		SVM	0.37 ± 0.11	**0.39 ± 0.12**	0.29 ± 0.03	0.42 ± 0.16	0.30 ± 0.07	0.32 ± 0.08
	Cars2	**XGB**	0.42 ± 0.14	**0.30 ± 0.05**	**0.29 ± 0.00**	**0.29 ± 0.00**	0.30 ± 0.06	0.31 ± 0.07
		RF	0.39 ± 0.11	0.30 ± 0.05	0.30 ± 0.06	0.31 ± 0.08	**0.29 ± 0.00**	0.31 ± 0.07
		MLP	**0.31 ± 0.07**	0.30 ± 0.06	0.41 ± 0.11	0.29 ± 0.00	0.31 ± 0.07	**0.29 ± 0.04**
		SVM	0.44 ± 0.13	0.33 ± 0.11	0.34 ± 0.09	0.29 ± 0.00	0.29 ± 0.00	0.32 ± 0.08
	Wine	XGB	**0.49 ± 0.36**	**0.70 ± 0.29**	0.99 ± 0.26	0.33 ± 0.32	0.70 ± 0.37	0.90 ± 0.23
		RF	0.75 ± 0.33	0.89 ± 0.24	1.00 ± 0.19	0.47 ± 0.33	0.58 ± 0.33	0.69 ± 0.29
		MLP	0.60 ± 0.32	0.96 ± 0.21	1.09 ± 0.16	0.30 ± 0.23	0.53 ± 0.31	0.65 ± 0.32
		SVM	0.74 ± 0.09	0.89 ± 0.07	**0.95 ± 0.04**	**0.28 ± 0.21**	**0.50 ± 0.29**	**0.64 ± 0.32**
1vALL	Body Perf.	XGB	0.26 ± 0.20	0.45 ± 0.16	0.55 ± 0.17	-		
		RF	0.28 ± 0.20	0.46 ± 0.20	0.53 ± 0.20			
		MLP	**0.26 ± 0.17**	**0.38 ± 0.08**	**0.46 ± 0.08**			
		SVM	0.37 ± 0.19	0.45 ± 0.12	0.47 ± 0.09			
	Thyroid	**XGB**	**0.17 ± 0.13**	**0.32 ± 0.34**	0.47 ± 0.35			
		RF	0.42 ± 0.41	0.36 ± 0.43	**0.34 ± 0.38**			
		MLP	0.56 ± 0.40	0.65 ± 0.42	0.79 ± 0.46			
		SVM	0.79 ± 0.33	0.88 ± 0.45	0.87 ± 0.35			
	Cars	XGB	0.34 ± 0.10	0.29 ± 0.03	**0.31 ± 0.06**			
		RF	0.29 ± 0.00	0.30 ± 0.06	0.43 ± 0.11			
		MLP	**0.29 ± 0.00**	**0.29 ± 0.00**	0.33 ± 0.09			
		SVM	0.31 ± 0.07	0.37 ± 0.11	0.38 ± 0.12			
	Car2	**XGB**	**0.29 ± 0.00**	0.31 ± 0.07	0.32 ± 0.08			
		RF	0.31 ± 0.06	0.31 ± 0.07	**0.29 ± 0.04**			
		MLP	0.29 ± 0.00	**0.29 ± 0.04**	0.34 ± 0.10			
		SVM	0.30 ± 0.05	0.37 ± 0.11	0.33 ± 0.09			
	Wine	XGB	0.38 ± 0.31	0.65 ± 0.27	0.71 ± 0.25			
		RF	**0.29 ± 0.23**	**0.50 ± 0.28**	**0.57 ± 0.25**			
		MLP	0.44 ± 0.22	0.71 ± 0.21	0.75 ± 0.15			
		SVM	0.49 ± 0.23	0.75 ± 0.17	0.83 ± 0.12			

Table 12: Report 2 overview: analysis of minimality in 1v1 and 1vAll modes on Benchmark datasets. The values in bold represent the minimum values within each group with the same conditions. A BB classifier is highlighted in bold if it has the highest number of bold values in a row. In case of a tie, specific counterfactuals are prioritized.

Modalities 1v1 and 1vAll: execution time and anomaly score

Dataset	BB Classifier	Counterfactual generation Time Elapsed [hh:mm:ss] (# Counterfactuals)		Avg. Isolation Forest Score on ACCEPTABLE cf.	
		1V1	1VALL	1V1	1VALL
Body Performance				Train set avg. Score: **0.04 ± 0.04**	
	XGB	10:09 (872)	**05:58** (354)	-0.03 ± 0.04	-0.03 ± 0.03
	RF	20:09 (780)	08:41 **(477)**	-0.02 ± 0.04	-0.03 ± 0.03
	MLP	**04:27 (1244)**	10:44 (362)	-0.02 ± 0.04	**-0.01 ± 0.03**
	SVM	07:18 (1208)	19:56 (312)	-0.02 ± 0.04	-0.01 ± 0.04
Thyroid				Train set avg. Score: **0.12 ± 0.06**	
	XGB	**01:51:15** (933)	**01:09:15** (234)	**0.03 ± 0.07**	**0.07 ± 0.05**
	RF	03:39:25 (1345)	02:05:23 (473)	0.00 ± 0.07	0.05 ± 0.08
	MLP	06:10:33 **(2353)**	01:49:35 **(2088)**	0.01 ± 0.06	-0.01 ± 0.09
	SVM	05:49:36 (1205)	03:07:41 (1159)	-0.05 ± 0.06	-0.08 ± 0.06
Cars				Train set avg. Score: **-0.06 ± 0.01**	
	XGB	26:06 **(1023)**	07:32 (352)	-0.06 ± 0.01	-0.06 ± 0.01
	RF	28:02 (853)	08:38 (283)	-0.06 ± 0.01	-0.06 ± 0.01
	MLP	**22:30** (964)	**05:56 (471)**	-0.06 ± 0.01	-0.06 ± 0.01
	SVM	22:52 (756)	06:28 (270)	-0.06 ± 0.01	-0.06 ± 0.01
Cars2				Train set avg. Score: **-0.06 ± 0.01**	
	XGB	15:22 (721)	06:50 (409)	-0.06 ± 0.01	-0.06 ± 0.01
	RF	22:35 (771)	09:56 **(469)**	-0.06 ± 0.01	-0.06 ± 0.01
	MLP	**12:44 (904)**	**05:35** (397)	-0.06 ± 0.01	-0.06 ± 0.01
	SVM	13:20 (793)	06:14 (437)	-0.06 ± 0.01	-0.06 ± 0.01
Wine				Train set avg. Score: **0.05 ± 0.04**	
	XGB	03:01 (462)	01:04 **(320)**	-0.03 ± 0.06	-0.03 ± 0.06
	RF	07:33 (366)	03:20 (246)	**-0.02 ± 0.05**	**0.00 ± 0.04**
	MLP	**01:36 (483)**	**00:34 (233)**	-0.02 ± 0.06	-0.04 ± 0.06
	SVM	14:12 (263)	00:39 (218)	-0.02 ± 0.07	-0.04 ± 0.05

Table 13: Report 2 overview: analysis of Execution Time and Anomaly Score in 1v1 and 1vAll modes on Benchmark datasets. The execution time refers to the amount of time elapsed during the generation of counterfactuals for the 100 sampled instances. The BB classifiers in bold are those that generated more counterfactuals in less time, measured by the ratio of counterfactuals produced to time elapsed. For the Isolation Forest score, in bold are highlighted those that have the most similar score to the score computed on the training set.

Considerations

In all the cases, the generated counterfactual results are consistently good for both modalities; no single black-box model significantly outperforms the others, as their performance remains comparable. Next are reported specific considerations from each single dataset used.

Body Performance

For this dataset, the results are highly impactful, with an impressive **97.27%** of acceptable counterfactuals achieved in the **1v1** mode using an **SVM** black box. Among these, **55.71%** are specific counterfactuals. In the **1vAll** simpler modality, the best performance is obtained using an **XGB** black box, achieving about **88%** acceptable counterfactuals, which is similar to what is obtained with an **RF** black box.

From a minimality perspective, like in the previous report, the **specific counterfactuals** confirm to be **closer** to their test instance with **respect** to the **plausible** counterfactuals.

Thyroid

In this case, the method used with the **XGB** black-box model achieves the highest percentage of acceptable counterfactuals in the **1v1** modality, reaching approximately **97%**, with an impressive **91%** being **specific** counterfactuals a significant result. The same happens also in the **1vAll** mode with **XGB** and **Random Forest (RF)** models that follow closely. XGB is also the method that comes closest to the Isolation Forest score calculated on the train set.

Given that the **1v1** modality primarily focuses on specific counterfactuals, **XGB** is considered the most suitable choice. However, this trend does not hold for the **1vAll** modality. While **XGB** still achieves the highest percentage of acceptable counterfactuals, it fails to satisfy **15 out of 96** instances, making **RF** the preferred model for this modality.

In terms of minimality, both **XGB** and **RF** yield similar results, but **RF** performs better in the **1v1** modality, whereas **XGB** is superior in the **1vAll** modality. It is important to note that the **execution time** employing this dataset is **quite large**. The motivation is due to the fact that all attributes are accountable (no constraints are present), so a larger set of operations is performed on the available features, especially in the **1v1** mode, the most computationally expensive one.

Cars and Cars2

In these datasets, all the black boxes have very comparable and high performances, quite similar to each other. In the **1v1** modality, **Random Forest** is the model that performs best, achieving a **99% acceptability rate**, with **69%** being specific counter-

factuals. In contrast, the **1vAll** modality presents a slightly different scenario. While performance remains comparable, a notable number of instances **fail to generate counterfactuals**. However, for instances where counterfactuals can be produced, the acceptability rate remains very high.
From a minimality perspective, all models perform similarly, with **XGB** performing best in the **1v1** modality and **MLP** performing best in the **1vAll** modality. It is important to remember that all the attributes are categorical and just **3 out of 6** possible ones are kept fixed. The presence of only categorical features has influenced minimality, particularly the *Jaccard Measure*, highlighting a peculiar increment/decrement pattern among the three closest counterfactuals. This differs from other datasets, where increasing the number of points leads to a steady increase in the average minimal distance. The same reasoning also applies to the a**verage anomaly score**, which turns out to be exactly the **same** as the one calculated on the **train set**.
In **Cars2**, the scenario is similar to what is observed in **Cars**, but with the key difference that here, the number of classes to handle is just **3**, leading to **even better performances** than in the previous case. This dataset represents the **best scenario** obtained in absolute among all datasets, reaching an **acceptability** level close to **100%**, with a high level of specificity at **75%**. Even from the perspective of minimality, the situation is improved.

Wine
This dataset has the **lowest testing accuracy** (so a reduced number of instances are correctly classified) and, as a consequence, a reduced number of counterfactuals to be examined. However, it presents nice acceptability and specificity values with higher values in 1v1 Mode with respect to 1vAll. It is important to highlight that this is the dataset that **take less computational time** among the ones used as Benchmark.

Final generic observations

- In case of less populated dataset, it is difficult to be able to generate always some counterfactual, especially if constraints (fixed features) are present.
- Although MLP is not always the black-box model with the highest accuracy, using it as the black box generally results in the generation of more counterfactuals compared to other models.
- Using this method, have evidence that the **percentage** of **discarded counterfactuals is** quite **low** and does not overcome the 40% in the worst case on these Benchmark datasets. The level of specificity is quite high, so it encourages to use the 1V1 modality to obtain a specific counterfactuals explanation.

- Except for some specific case, the **1vAll modality** is **significantly faster** than the 1v1 mode, because the number of latent spaces to be investigated is exactly 1 respect to the (num_classes – 1) spaces in the other 1v1 mode, so in most of the cases the **time proportion is respected**. As consequence even the number of counterfactuals generated is proportional to the latent spaces used.

- The Isolation Forest score gas highlighted that in most of the cases, although it is **slightly lower** respect to the one computed on the training set, it represent an **acceptable value** for the counterfactuals produced.

- The **absence** of a **clear winning black-box** model encourages the use of the proposed method, as it remains **independent** of any specific black-box model while delivering consistently **similar performance** across different datasets and problems.

5.4.3 Report 3: Multi-Class CPILS - KEmoCon Datasets

In this experimentation, the same operations executed for Report 2 are repeated for KEmoCon datasets. The same metrics are analysed from a different point of view. For each considered Subject, a direct comparison between the 1v1 and 1vAll modalities is done in order to highlight better the differences between the two modes.

Subject 28: capability, acceptability and specificity

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC			
1v1	XGB	1/100 (*5)	1581	677 (42.82%)	293 (18.53%)			
	RF	**0/100 (*18)**	2481	1470 (59.25%)	1146 (46.19%)			
	MLP	0/91	**4168**	**3381** (81.12%)	**2117** (50.79%)			
	SVM	0/81	2551	2350 **(92.12%)**	1759 **(68.95%)**			
					Class 0	**Class 1**	**Class 2**	**Class 3**
1vALL	XGB	0/100	832	469 (56.37%)	**278** (1.16 ± 0.30)	146 **(1.03 ± 0.41)**	45 (1.04 ± 0.38)	-
	RF	**0/100**	**2195**	938 (42.73%)	**415 (0.94 ± 0.36)**	393 (0.99 ± 0.30)	130 (1.25 ± 0.28)	-
	MLP	0/91	1528	1168 (76.44%)	323 (0.94 ± 0.25)	**725 (0.87 ± 0.24)**	120 (0.88 ± 0.30)	-
	SVM	0/81	1953	**1624 (83.15%)**	625 (1.03 ± 0.28)	**769 (0.94 ± 0.22)**	230 (1.01 ± 0.16)	-

Table 14: Report 3 overview of Subject 28: analysis of capability, counterfactual quantity, and specificity in 1v1 and 1vAll modalities.

Subject 28: minimality and anomaly score

Method	BB Classifier	PLAUSIBLE Avg. Proximity Distance from the Nth closest cf.			SPECIFIC Avg. Proximity Distance from the Nth closest cf.			Avg. Isolation Forest Score on ACCEPTABLE cf.	
		1st	**2nd**	**3rd**	**1st**	**2nd**	**3rd**	Train set avg. score	**0.07 ± 0.05**
1v1	XGB	0.76 ± 0.36	0.95 ± 0.26	0.94 ± 0.26	0.73 ± 0.44	0.92 ± 0.36	1.15 ± 0.28	**0.01 ± 0.06**	
	RF	0.83 ± 0.48	0.98 ± 0.31	1.12 ± 0.27	0.63 ± 0.50	0.84 ± 0.49	0.96 ± 0.45	-0.02 ± 0.06	
	MLP	**0.44 ± 0.21**	**0.63 ± 0.25**	**0.71 ± 0.24**	**0.28 ± 0.22**	**0.43 ± 0.24**	**0.62 ± 0.24**	0.00 ± 0.05	
	SVM	0.60 ± 0.26	0.82 ± 0.23	0.96 ± 0.14	0.36 ± 0.15	0.57 ± 0.21	0.66 ± 0.20	0.00 ± 0.05	
1vALL	XGB	0.66 ± 0.41	1.01 ± 0.27	1.10 ± 0.24				0.00 ± 0.05	
	RF	0.71 ± 0.42	0.88 ± 0.36	0.96 ± 0.32	-	-	-	0.01 ± 0.05	
	MLP	**0.35 ± 0.17**	**0.60 ± 0.21**	**0.72 ± 0.16**				0.01 ± 0.04	
	SVM	0.58 ± 0.19	0.71 ± 0.18	0.82 ± 0.18				0.01 ± 0.04	

Table 15: Report 3 overview of Subject 28: analysis of minimality and anomaly score in 1v1 and 1vAll modalities.

Subject 5: capability, acceptability and specificity

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC			
1v1	XGB	**0/100 (*1)**	1080	600 (55.56%)	328 (30.37%)			
	RF	0/100 (*5)	2947	1639 (55.62%)	1069 (36.27%)			
	MLP	0/96 (*13)	**3633**	**2765 (76.11%)**	1636 (45.03%)			
	SVM	0/92 (*6)	3407	2537 (74.46%)	**1803 (52.92%)**			
					Class 0	Class 1	Class 2	Class 3
1vALL	XGB	**0/100**	867	698 (80.51%)	202 (1.04 ± 0.23)	**285 (0.63 ± 0.31)**	152 (0.97 ± 0.36)	59 (0.77 ± 0.32)
	RF	0/100	**2272**	608 (26,76%)	62 (1.05 ± 0.21)	**342 (0.90 ± 0.32)**	197 (0.97 ± 0.41)	7 (0.94 ± 0.25)
	MLP	0/96	1822	1578 (86.61%)	294 (1.02 ± 0.21)	**800** (0.94 ± 0.26)	435 (1.02 ± 0.31)	49 **(0.83 ± 0.24)**
	SVM	0/92	1936	1278 (66.01%)	257 (1.11 ± 0.25)	**781** (1.00 ± 0.36)	204 (1.02 ± 0.22)	36 **(0.86 ± 0.18)**

Table 16: Report 3 overview of Subject 5: analysis of capability, counterfactual quantity, and specificity in 1v1 and 1vAll modalities.

Subject 5: minimality and anomaly score

Method	BB Classifier	PLAUSIBLE Avg. Proximity Distance from the Nth closest cf.			SPECIFIC Avg. Proximity Distance from the Nth closest cf.			Avg. Isolation Forest Score on ACCEPTABLE cf.
		1st	2nd	3rd	1st	2nd	3rd	Train set avg. score **0.06 ± 0.05**
1v1	XGB	0.59 ± 0.34	0.81 ± 0.48	0.90 ± 0.51	0.46 ± 0.33	0.53 ± 0.30	0.71 ± 0.35	**0.02 ± 0.06**
	RF	0.66 ± 0.29	0.80 ± 0.27	0.92 ± 0.29	0.54 ± 0.30	0.66 ± 0.30	0.76 ± 0.30	0.00 ± 0.06
	MLP	0.66 ± 0.38	0.73 ± 0.39	0.79 ± 0.38	**0.38 ± 0.26**	0.53 ± 0.25	0.63 ± 0.20	-0.01 ± 0.05
	SVM	**0.59 ± 0.31**	**0.73 ± 0.31**	**0.71 ± 0.18**	0.44 ± 0.24	**0.52 ± 0.21**	**0.59 ± 0.19**	0.00 ± 0.05
1vALL	**XGB**	**0.54 ± 0.29**	0.68 ± 0.25	**0.73 ± 0.27**	-	-	-	0.02 ± 0.06
	RF	0.62 ± 0.31	0.82 ± 0.38	0.86 ± 0.24				0.00 ± 0.05
	MLP	0.58 ± 0.26	**0.68 ± 0.22**	0.76 ± 0.17				0.00 ± 0.05
	SVM	0.59 ± 0.25	0.76 ± 0.18	0.80 ± 0.16				0.00 ± 0.06

Table 17: Report 3 overview of Subject 5: analysis of minimality, and anomaly score in 1v1 and 1vAll modalities.

Subject 17: capability, acceptability and specificity

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC				
1v1	**XGB**	**0/100 (*4) (***1)**	1797	1264 (70.34%)	682 **(37.95%)**				
	RF	0/100 (*30)	**6582**	3122 (47.43%)	**1442** (21.91%)				
	MLP	0/80 (**2)	5190	**4097 (78.94%)**	1361 (26.22%)				
	SVM	1/69	4927	3417 (69.35%)	1375 (27.91%)				
					Class 0	Class 1	Class 2	Class 3	Class 4
1vALL	XGB	1/100	540	384 (71.11%)	**144** (1.09 ± 0.24)	73 (0.88 ± 0.37)	41 **(0.63 ± 0.36)**	100 (0.91 ± 0.31)	26 (1.00 ± 0.35)
	RF	**0/100**	**2531**	660 (26.08%)	69 (1.04 ± 0.20)	117 (0.95 ± 0.32)	**222 (0.84 ± 0.30)**	120 (0.93 ± 0.28)	132 (1.00 ± 0.31)
	MLP	1/80	1558	**1261 (80.94%)**	54 (0.86 ± 0.20)	191 (0.81 ± 0.31)	248 (0.84 ± 0.26)	363 (0.86 ± 0.24)	**405 (0.73 ± 0.23)**
	SVM	1/69	1368	861 (62.94%)	15 (0.98 ± 0.17)	**359** (1.01 ± 0.22)	41 **(0.60 ± 0.24)**	328 (0.95 ± 0.23)	118 (0.85 ± 0.23)

Table 18: Report 3 overview of Subject 17: analysis of Capability, Counterfactual quantity, and Specificity in 1v1 and 1vAll modalities.

Subject 17: minimality and anomaly score

Method	BB Classifier	PLAUSIBLE Avg. Proximity Distance from the Nth closest cf.			SPECIFIC Avg. Proximity Distance from the Nth closest cf.			Avg. Isolation Forest Score on ACCEPTABLE cf.	
		1st	2nd	3rd	1st	2nd	3rd	Train set avg. score	**0.09 ± 0.05**
1v1	XGB	0.46 ± 0.25	0.72 ± 0.30	0.85 ± 0.33	**0.28 ± 0.31**	0.46 ± 0.30	0.70 ± 0.38	**0.02 ± 0.07**	
	RF	0.54 ± 0.34	0.67 ± 0.34	0.71 ± 0.27	0.46 ± 0.31	0.60 ± 0.40	0.73 ± 0.42	0.01 ± 0.07	
	MLP	**0.31 ± 0.20**	**0.38 ± 0.20**	**0.46 ± 0.21**	0.30 ± 0.21	**0.41 ± 0.27**	**0.48 ± 0.29**	0.02 ± 0.06	
	SVM	0.39 ± 0.21	0.48 ± 0.17	0.54 ± 0.16	0.39 ± 0.21	0.46 ± 0.19	0.53 ± 0.19	0.01 ± 0.06	
1vALL	XGB	0.59 ± 0.30	1.00 ± 0.26	0.99 ± 0.26				0.01 ± 0.07	
	RF	0.58 ± 0.32	0.80 ± 0.29	0.87 ± 0.24	-	-	-	0.03 ± 0.06	
	MLP	**0.39 ± 0.22**	**0.55 ± 0.23**	**0.60 ± 0.23**				0.01 ± 0.06	
	SVM	0.59 ± 0.23	0.72 ± 0.22	0.80 ± 0.22				0.02 ± 0.05	

Table 19: Report 3 overview of Subject 17: analysis of minimality and anomaly score in 1v1 and 1vAll modalities.

Subject 17: capability, acceptability and specificity

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC			
1v1	XGB	0/100 (*3)	447	425 **(95.08%)**	361 (80.76%)			
	RF	**0/100**	2624	1439 (54.84%)	1005 (38.30%)			
	MLP	0/99	**3072**	**2385** (77.64%)	1401 (45.61%)			
	SVM	0/97	2309	1977 (85.62%)	**1586** (68.69%)			
					Class 0	Class 1	Class 2	Class 3
1vALL	**XGB**	**0/100**	649	586 **(90.29%)**	102 **(0.89 ± 0.62)**	**370** (1.03 ± 0.20)	114 (1.26 ± 0.35)	-
	RF	0/100	**2679**	852 (31.80%)	253 **(0.85 ± 0.44)**	**507** (1.16 ± 0.28)	92 (1.30 ± 0.37)	-
	MLP	0/99	1629	1087 (66.73%)	267 **(0.89 ± 0.23)**	**619** (0.97 ± 0.29)	201 (1.04 ± 0.24)	-
	SVM	0/97	1785	947 (53.05%)	84 **(0.60 ± 0.25)**	**794** (0.98 ± 0.22)	69 (0.85 ± 0.22)	-

Table 20: Report 3 overview of Subject 27: analysis of capability, counterfactual quantity, and specificity in 1v1 and 1vAll modalities.

Subject 27: minimality and anomaly score

Method	BB Classifier	PLAUSIBLE Avg. Proximity Distance from the Nth closest cf.			SPECIFIC Avg. Proximity Distance from the Nth closest cf.			Avg. Isolation Forest Score on ACCEPTABLE cf.	
		1st	2nd	3rd	1st	2nd	3rd	Train set avg. score	**0.07 ± 0.05**
1v1	XGB	0.97 ± 0.29	-	-	0.54 ± 0.38	0.61 ± 0.44	0.85 ± 0.33	**0.02 ± 0.06**	
	RF	1.00 ± 0.37	1.33 ± 0.41	1.20 ± 0.36	0.69 ± 0.32	0.87 ± 0.34	1.03 ± 0.36	-0.03 ± 0.07	
	MLP	**0.66 ± 0.25**	**0.80 ± 0.30**	**0.88 ± 0.33**	0.53 ± 0.25	**0.56 ± 0.24**	0.72 ± 0.23	0.00 ± 0.05	
	SVM	0.82 ± 0.21	0.88 ± 0.15	0.95 ± 0.17	**0.45 ± 0.13**	0.60 ± 0.17	**0.71 ± 0.17**	0.00 ± 0.05	
1vALL	**XGB**	**0.43 ± 0.39**	0.94 ± 0.21	1.15 ± 0.25				0.00 ± 0.06	
	RF	0.64 ± 0.35	0.81 ± 0.28	0.94 ± 0.29	-	-	-	-0.01 ± 0.06	
	MLP	0.56 ± 0.23	0.69 ± 0.22	**0.81 ± 0.18**				0.01 ± 0.05	
	SVM	0.55 ± 0.17	**0.69 ± 0.20**	0.83 ± 0.18				0.01 ± 0.05	

Table 21: Report 3 overview of Subject 27: analysis of minimality and anomaly score in 1v1 and 1vAll modalities.

Subject 9: capability, acceptability and specificity

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC			
1v1	XGB	11/99 (*58) (**30)	846	521 (61.58%)	501 (59.22%)			
	RF	18/100 (*27) (**55)	2428	1570 (64.66%)	1217 (50.12%)			
	MLP	**0/89**	**3856**	**3078 (79.82%)**	**1423** (36.90%)			
	SVM	3/85(*77) (**5)	2177	1659 (76.21%)	1338 **(61.46%)**			
					Class 0	Class 1	Class 2	Class 3
1vALL	XGB	33/99	492	342 (69.51%)	69 (0.83 ± 0.24)	**197** (0.80 ± 0.32)	74 (0.92 ± 0.14)	2 **(0.78 ± 0.58)**
	RF	14/100	1299	1052 (80.99%)	0 (-)	**1026** (0.85 ± 0.25)	25 (0.97 ± 0.15)	1 **(0.37 ± 0.00)**
	MLP	**0/89**	**1330**	**1125** (84.59%)	182 **(0.78 ± 0.25)**	**660** (0.78 ± 0.28)	179 (0.97 ± 0.18)	104 (0.97 ± 0.18)
	SVM	5/85	1144	1008 **(88.11%)**	84 **(0.80 ± 0.21)**	867 (0.93 ± 0.20)	57 (0.91 ± 0.13)	0 (-)

Table 22: Report 3 overview of Subject 9: analysis of capability, counterfactual quantity, and specificity in 1v1 and 1vAll modalities.

Subject 9: minimality and anomaly score

Method	BB Classifier	PLAUSIBLE Avg. Proximity Distance from the Nth closest cf.			SPECIFIC Avg. Proximity Distance from the Nth closest cf.			Avg. Isolation Forest Score on ACCEPTABLE cf.	
		1st	2nd	3rd	1st	2nd	3rd	Train set avg. score	**0.07 ± 0.05**
1v1	XGB	0.62 ± 0.26	-	-	0.58 ± 0.22	0.74 ± 0.24	0.90 ± 0.22	-0.01 ± 0.06	
	RF	0.63 ± 0.02	0.66 ± 0.03	0.68 ± 0.03	**0.37 ± 0.16**	0.63 ± 0.29	0.80 ± 0.23	-0.02 ± 0.05	
	MLP	**0.56 ± 0.23**	**0.62 ± 0.24**	**0.64 ± 0.24**	0.37 ± 0.22	**0.50 ± 0.23**	**0.57 ± 0.23**	-0.01 ± 0.06	
	SVM	0.64 ± 0.27	0.98 ± 0.19	1.01 ± 0.18	0.45 ± 0.22	0.65 ± 0.22	0.72 ± 0.21	-0.02 ± 0.07	
1vALL	XGB	0.52 ± 0.25	0.80 ± 0.23	0.76 ± 0.23	-	-	-	**0.00 ± 0.06**	
	RF	0.54 ± 0.22	0.78 ± 0.27	0.80 ± 0.26				-0.02 ± 0.05	
	MLP	**0.48 ± 0.23**	**0.57 ± 0.23**	**0.61 ± 0.21**				0.00 ± 0.06	
	SVM	0.77 ± 0.20	0.82 ± 0.19	0.85 ± 0.22				-0.02 ± 0.05	

Table 23: Report 3 overview of Subject 9: analysis of minimality and anomaly score in 1v1 and 1vAll modalities.

Execution time

Dataset	BB Classifier	Counterfactuals generation	
		Time Elapsed [hh:mm:ss] (# Counterfactuals)	
(# Classes)		1V1	1VALL
Subj_28	XGB	34:24 (1581)	17:18 (832)
	RF	59:11 (2481)	19:40 (2195)
(3)	**MLP**	**21:45 (4168)**	**06:42** (1528)
	SVM	33:58 (2551)	10:26 (1953)
Subj_5	XGB	**01:00:45** (1080)	18:01 (867)
	RF	01:33:02 (2947)	21:55 **(2272)**
(4)	MLP	03:19:45 **(3633)**	28:46 (1822)
	SVM	02:53:22 (3407)	**15:52** (1936)
Subj_17	XGB	**01:35:43** (1797)	22:05 (540)
	RF	02:10:20 **(6582)**	19:13 **(2531)**
(5)	MLP	02:52:39 (5190)	**16:26** (1558)
	SVM	02:42:13 (4927)	29:32 (1368)
Subj_27	XGB	55:42 (447)	26:29 (649)
	RF	01:03:16 (2624)	23:33 **(2679)**
(3)	**MLP**	**36:32 (3072)**	**07:09** (1629)
	SVM	01:09:34 (2309)	14:49 (1785)
Subj_9	XGB	**01:01:05** (846)	**21:50** (492)
	RF	02:20:08 (2428)	29:20 (1299)
(4)	MLP	03:42:32 **(3856)**	38:36 **(1330)**
	SVM	06:01:33 (2177)	27:34 (1144)

Table 24: Report 3 overview of all the Subjects: analysis of execution time in relation to how many counterfactuals are produced in 1v1 and 1vAll modalities.

Considerations

These results highlight that in all the cases, also for KEmoCon datasets, it is possible to obtain a valid number of counterfactuals in terms of the metrics analysed. Of course, depending on the available dataset and the black box classifiers, some performances are better than others.

- In most of the cases, the multi-class CPILS method using **MLP black box** generates the **highest number of counterfactuals** with higher percentages of acceptability. Secondly, **SVM** has **similar performances**. For example, for subject 28 in 1v1 Mode, 3381 acceptable counterfactuals, which are 81.12% of the total with about 50% specific. Similarly, for subject 27 in 1v1 Mode, with 2385 acceptable counterfactuals that are 77.64% of the total with about 45% specific.

Also, in terms of **minimality**, MLP is one of those that **presents the lowest values** in most cases.

- For these specific datasets, which share the same knowledge domain, it seems that **XGB and RF** have **lower performances** compared to MLP and SVM. It is more evident for **RF**, where for at least 3/5 subjects in 1vAll Mode, it achieves a very **poor acceptability level**. The same happens in 1v1 Mode but with a higher and more tolerable level of acceptability.

- It is important to highlight the presence and the performance of **Subject 17**, which contains a classification among **5 classes**. Looking at 1v1 Mode, the percentage of **acceptability** is **quite good**. The percentage of **specificity** is quite **limited** but similar for all cases. This is explainable because increasing the number of classes to be recognized makes it **harder to keep specific counterfactuals**. This is also reflected in **execution time**, where the values are **on average greater** compared to other cases. For the 1vAll mode, it is observed that all the classes are recognized, with, of course, some preference but without evident gaps.

- Beyond the percentage of miss-classification of original Black box models, from **these experiments emerges** the reason **why** some **instances are not capable** of generating counterfactuals. This is evident for **Subject 9** (the last one) and less evident in **Subject 17**. These two ones are respectively trained with 1960 and 3500 instances, and they must distinguish between 4 and 5 classes. So, for each one, each class has 490 and 700 instances, **numbers that are incredibly lower** with respect to other considered subjects. If it is considered that also **3 of the 18** attributes should remain **fixed**, it is explained how, in some cases, some instances may fail. This is also reflected in execution time, where the values are the highest among all the other cases.

- The **execution time** in general reflects **how many latent spaces** must be used to search for counterfactuals. **1vAll** mode have in all the cases **lower values** because there is only one latent space to be used. The **quality** of a trained **latent space** may also influence the execution time because, in an approach like this one, a binary combination of classes could be trained better than others. This difference is observed during the counterfactual generation process. Some **classes of counterfactuals** can be generated **more rapidly** than others.

5.4.4 Report 4: Multi-Class CPILS - impact of latent space dimension

This experiment evaluates how the latent space dimension affects the previously discussed metrics, aiming to determine whether increasing this dimension can have any benefits.

In this report, to maintain a focused analysis on this task only a datasets per group is selected: **Subject 5** from KEmoCon datasets which has 4 classes to be recognized and also a larger training instances and the **Wine** dataset from the Benchmark, which has 3 classes to be recognized but fewer training instances and it is also slightly imbalanced. The number of instances is deliberately mentioned because a higher instance count generally leads to more accurate latent space generation, so it is possible to have a different perspective on how latent space dimensionality impacts performance.
To cover as much and more representative latent spaces dimension, while considering the fixed number of attributes and accountable in both datasets, four latent dimensions are analysed: **2, 3, 5 and 7**. The tested methodologies are still 1v1 and 1vAll as the black box models previously employed. For each test dataset, a sample of 100 instances is taken and submitted on the generated counterfactuals method of the multi-class CP_ILS. The samples are taken trying to uniform consider all the classes involved in order to try to generate counterfactuals evenly for all the classes.

To present the results effectively, the metrics are structured according to variations in latent space size. The primary focus is on specificity and acceptability, followed by the anomaly score and execution time. Additionally, a separate table reports the generation capability and how it is influenced by the latent space dimension.

Subject 5

		1v1				1vAll		
BB Classifier	**Dim (k)**	**Cf. ACCEPTABLE**	**Cf. SPECIFIC**	**Isolation Forest Score**	**Time [hh:mm:ss]**	**Cf. ACCEPTABLE**	**Isolation Forest Score**	**Time [hh:mm:ss]**
XGB	2	600/1080 (55.56%)	328 (30.37%)	0.02 ± 0.06	**01:00:45**	698/867 (80.51%)	**0.02 ± 0.06**	18:01
	3	1441/2060 (69.95%)	1015 (49.27%)	**0.02 ± 0.05**	01:03:31	741/933 (79.42%)	0.00 ± 0.06	18:13
	5	1917/2640 **(72.61%)**	1323 **(50.11%)**	0.01 ± 0.05	01:00:50	**1121/1389 (80.71%)**	0.00 ± 0.05	**17:01**
	7	1581/2575 (61.40%)	1128(43.81%)	0.01 ± 0.06	01:02:50	1175/1516 (77.51%)	0.00 ± 0.06	17:08
RF	2	1639/2947 (55.62%)	1069 (36.27%)	0.00 ± 0.06	01:33:02	608/2272 (26.76%)	0.00 ± 0.05	21:55
	3	1996/3391 (58.86%)	1223 (36.07%)	**0.01 ± 0.06**	01:33:42	1084/2488 **(43.57%)**	**0.01 ± 0.05**	**19:11**
	5	2918/4764 (61.25%)	1547 (32.47%)	-0.01 ± 0.06	01:30:58	1176/3059 (38.44%)	0.00 ± 0.06	19:19
	7	3051/4949 **(61.65%)**	1971 **(39.83%)**	-0.02 ± 0.06	**01:30:31**	1243/3364 (36.95%)	-0.01 ± 0.06	19:25
MLP	2	2765/3633 (76.11%)	1636 (45.03%)	-0.01 ± 0.05	03:19:45	1578/1822 (86.61%)	**0.00 ± 0.05**	28:46
	3	4019/4853 **(82.81%)**	2377 **(48.98%)**	**-0.01 ± 0.06**	01:49:24	1822/2071 **(87.98%)**	-0.01 ± 0.05	**10:27**
	5	3834/5071 (75.61%)	2059 (40.60%)	-0.02 ± 0.06	01:20:45	2601/2964 (87.75%)	-0.01 ± 0.06	12:38
	7	4861/6448 (75.39%)	2706 (41.97%)	-0.02 ± 0.06	**01:20:23**	2338/2693 (86.82%)	-0.01 ± 0.05	12:26
SVM	**2**	2537/3407 **(74.46%)**	1803 **(52.92%)**	**0.00 ± 0.05**	02:53:22	1278/1936 **(66.01%)**	**0.00 ± 0.06**	15:25
	3	2919/4356 (67.01%)	1914 (43.94%)	-0.01 ± 0.05	02:31:08	1614/2531 (63.77%)	-0.01 ± 0.05	14:20
	5	3808/5826 (65.36%)	2535 (43.51%)	-0.01 ± 0.06	02:52:57	1614/2835 (56.93%)	-0.01 ± 0.05	**09:51**
	7	3371/5928 (56.87%)	2191 (36.96%)	-0.01 ± 0.06	**02:20:46**	1598/2982 (53.59%)	-0.01 ± 0.06	12:45
			Train Set	**0.06 ± 0.05**				

Table 25: Report 4 overview of Subject 5: analysis of acceptability, specificity, anomaly score, and execution time in 1v1 and 1vAll modalities for different latent space dimensions.

Wine

		1v1				1vAll		
BB Classifier	**Dim (k)**	**Cf. ACCEPTABLE**	**Cf. SPECIFIC**	**Isolation Forest Score**	**Time [hh:mm:ss]**	**Cf. ACCEPTABLE**	**Isolation Forest Score**	**Time [hh:mm:ss]**
XGB	**2**	358/462 **(77.49%)**	253 (54.76%)	-0.03 ± 0.06	**03:01**	241/320 **(75.31%)**	-0.03 ± 0.06	**01:04**
	3	362/495 (73.13%)	272 **(54.95%)**	**-0.02 ± 0.05**	03:28	208/299 (69.57%)	**-0.01 ± 0.05**	01:17
	5	381/523 (72.85%)	286 (54.68%)	-0.02 ± 0.05	03:16	211/306 (68.95%)	-0.03 ± 0.06	01:15
	7	427/565 (75.58%)	302 (53.45%)	-0.03 ± 0.06	03:32	237/362 (65.47%)	-0.01 ± 0.06	01:19
RF	2	239/366 (65.30%)	239 (45.08%)	-0.02 ± 0.05	**07:33**	153/246 (62.20%)	**0.00 ± 0.04**	03:20
	3	322/487 (66.12%)	242 (49.69%)	**-0.02 ± 0.05**	07:43	156/286 (54.55%)	-0.02 ± 0.05	03:24
	5	371/514 **(72.18%)**	234 (45.53%)	-0.03 ± 0.06	07:37	210/331 (63.44%)	-0.04 ± 0.05	03:23
	7	407/576 (70.66%)	297 **(51.56%)**	-0.03 ± 0.06	08:18	257/388 **(66.24%)**	-0.03 ± 0.05	**02:57**
MLP	**2**	377/483 **(78.05%)**	281 **(58.18%)**	-0.02 ± 0.06	**01:36**	171/233 (73.39%)	-0.04 ± 0.06	**00:34**
	3	290/375 (77.33%)	212 (56.53%)	-0.03 ± 0.05	03:00	162/218 **(74.31%)**	-0.01 ± 0.06	01:11
	5	366/509 (71.91%)	276 (54.22%)	**-0.01 ± 0.05**	03:43	148/200 (74.00%)	**0.00 ± 0.05**	00:59
	7	324/428 (75.70%)	238 (55.61%)	-0.01 ± 0.06	03:10	181/252 (71.83%)	-0.01 ± 0.05	01:26
SVM	2	259/263 (98.48%)	231 (87.83%)	-0.02 ± 0.07	14:12	147/218 (67.43%)	-0.04 ± 0.05	**00:39**
	3	245/248 **(98.79%)**	220 **(88.71%)**	**-0.01 ± 0.05**	08:00	143/210 (68.10%)	-0.04 ± 0.05	02:44
	5	354/360 (98.33%)	285 (79.17%)	-0.03 ± 0.05	07:48	165/210 **(78.57%)**	-0.03 ± 0.05	02:52
	7	342/358 (95.53%)	295 (82.40%)	-0.02 ± 0.05	**05:20**	141/201 (70.15%)	**-0.03 ± 0.04**	01:59
			Train Set	**0.05 ± 0.04**				

Table 26: Report 4 overview of Wine: analysis of acceptability, specificity, anomaly score, and execution time in 1v1 and 1vAll modalities for different latent space dimensions.

BB Classifier	Dim (k)	Instances NOT ABLE (without cfs.)			
Dataset		Subject 5		Wine	
		1v1	1vAll	1v1	1vAll
XGB	2	0/100 (*1)	0/100	0/69 (*10)	0/69
	3	0/100	0/100	0/69 (*19)	1/69
	5	0/100 (*1)	0/100	0/69 (*10)	0/69
	7	0/100	0/100	0/69 (*9)	0/69
RF	2	**0/100 (*5)**	0/100	0/71 (*18)	5/71
	3	**0/100 (*5)**	0/100	0/71 (*24)	1/71
	5	**0/100**	0/100	0/71 (*21)	3/71
	7	**0/100**	0/100	0/71 (*22)	0/71
MLP	2	**0/96 (*13)**	0/96	**0/53 (*3)**	0/53
	3	**0/96 (*4)**	0/96	**0/53**	0/53
	5	**0/96 (*1)**	0/96	0/53	0/53
	7	**0/96**	0/96	0/53	0/53
SVM	2	**0/92 (*6)**	0/92	**4/51 (*33)**	0/51
	3	**0/92 (*2)**	0/92	**2/51 (*34)**	1/51
	5	**0/92 (*1)**	0/92	**0/51 (*31)**	5/51
	7	**0/92**	0/92	**0/51 (*16)**	2/51

Table 27: Report 4 overview of Subject 5 and Wine: analysis of generation capability in 1v1 and 1vAll modalities for different latent space dimensions. In subject 5: * means total instances unable to generate a counterfactual for exactly 1 out of 3 counter-classes In Wine: * means total instances unable to generate a counterfactual for exactly 1 out of 2 counter-classes. In bold are highlighted the significant improvements achieved by increasing the latent space dimension.

Considerations

The reports show the impact of varying the latent space dimension across different black-box classifiers, presenting interesting results.

- In most cases, **increasing** the **latent space dimension** results in a **higher number** of **generated counterfactuals**.
- The correct choice of the value of latent space dimension can have a nice impact on the percentage of acceptable and specific counterfactuals. There is **no universally optimal latent space dimension**; it varies depending on the dataset and the black-box model. For instance, the results indicate that **SVM performs** better with **lower latent dimensions**. In the case of **Subject 5**, performance begins to decline beyond a latent dimension of **2**. A similar trend is observed in the **Wine** dataset, where the model behaves consistently for **latent dimensions 2 and 3** but deteriorates beyond that point.
- The number of dimensions have a **small impact** on the **Isolation Forest Score** that result in all the cases quite near to the value computed on the trained dataset with small variations among the different classifiers. The best score is considered the one that is nearer to the training score.
- The **time required** to generate counterfactuals for the 100 test instances **remains roughly the same** for **XGB** and **RF** across all tested latent dimensions. However, a slight difference in processing time is observed for MLP and SVM from dimension 2 onward. This **variation** could be attributed to the latent space generation process, where **early stopping** may conditionally occur in lower dimensions. Having a less precise latent space has the subsequent consequence of impacting the generation process time.
- Table 27 presents the impact on the total **instances capable** of **generating counterfactuals**, independently if they are considered acceptable or not. It illustrates that in most cases **increasing the number of latent space dimension**, the **instances** able to generate at least a counterfactual **increase**. E.g., for **Subject 5** considering **MLP classifier**, in 2 latent dimensions, 13 instances present a problem in generating at least a counterfactual for 1 of 3 counter classes. In 3 dimensions, this number is reduced to 4, and in 5 dimensions, the same number is reduced again to 1. Using 7 dimensions, all the correctly classified instances can generate a counterfactual for all the available classes.

5.4.5 Report 5: Multi-Class CPILS - comparison with DiCE

The goal of this experimentation is to compare the multi-class CPILS with at least one state-of-the-art counterfactual explainer, such as DiCE (2.2), by analyzing the differences across various black-box models (4.4) and datasets (4.1).

In this section, the evaluation metrics introduced in (5.3) are used to provide a multi-perspective analysis of the advantages offered by the new approach with respect to the existing one. The datasets used in these experiments include Body Performance, Thyroid, and Wine from the Benchmark set, excluding Cars, as it contains only categorical features and would require additional preprocessing to be compatible with DiCE. For the emotion recognition task, three K-EMOTIONS datasets are utilized: Subject 28, Subject 17, and Subject 5.

The tested algorithms are:

- Multi-Class CPILS (1v1 mode) (CPILS for short) using **a static latent space dimension equal to 2**.
- DiCE in random version (DiCE R) - generates counterfactuals by randomly sampling feature values while ensuring they meet the desired outcome)
- DiCE genetic version (DiCE G) - uses an evolutionary approach to generate counterfactuals, optimizing diversity and validity

For each tested dataset, a sample of 100 instances are taken and submitted with the 3 methods cited above. The samples are taken trying to uniform consider all the classes involved in order to try to generate counterfactuals evenly for all the classes. To make an equal comparison, all the methodologies are set to produce **exactly 3 counterfactuals** for each counter-class. In addition, this type of analysis focues only on **SPECIFIC counterfactuals** (related to correctly classified test instances), as the last version DiCE is built to satisfy this property.

The following tables present the results of the metrics across different columns, indicating with the symbol *↑* when a higher value is preferable and *↓* when a lower value is better.

Body Performance

		SPEC [↑]	PROXIMITY [↓]		NON – PLAUSIBILITY [↓]		NON-ROBUSTNESS [↓]		DIVERSITY [↑]		TIME [↓]
BBOX	Method	Rate	Ddist	Dcount	IF	Dimpl	Drob1	Drob2	Ddiv1	Ddiv2	[hh:mm:ss]
XGB	CPILS	226/462 (48.92%)	0.48 ± 0.00	0.17 ± 0.06	-0.02 ± 0.04	0.34 ± 0.22	**0.16 ± 0.29**	**0.46 ± 0.33**	**0.98 ± 0.31**	**0.89 ± 0.12**	00:13:07
	DiCE (R)	**621/621 (100%)**	0.42 ± 0.22	**0.15 ± 0.03**	0.00 ± 0.04	0.27 ± 0.15	0.26 ± 0.34	0.54 ± 0.27	0.81 ± 0.26	0.89 ± 0.12	00:17:42
	DiCE (G)	599/613 (97.72%)	**0.18 ± 0.11**	0.81 ± 0.03	**0.05 ± 0.04**	**0.10 ± 0.06**	0.33 ± 0.33	0.61 ± 0.24	0.62 ± 0.28	0.88 ± 0.10	**00:01:05**
RF	CPILS	224/473 (47.36%)	0.48 ± 0.30	0.18 ± 0.09	-0.02 ± 0.04	0.34 ± 0.24	**0.11 ± 0.21**	**0.42 ± 0.29**	**0.98 ± 0.33**	0.88 ± 0.12	00:20:52
	DiCE (R)	**648/648 (100%)**	0.45 ± 0.22	**0.16 ± 0.04**	-0.01 ± 0.04	0.29 ± 0.15	0.25 ± 0.30	0.53 ± 0.26	0.84 ± 0.26	0.89 ± 0.11	00:21:05
	DiCE (G)	609/630 (98.57%)	**0.17 ± 0.10**	0.82 ± 0.03	**0.05 ± 0.04**	**0.10 ± 0.06**	0.42 ± 0.48	0.63 ± 0.22	0.63 ± 0.28	**0.89 ± 0.10**	**00:02:12**
MLP	CPILS	240/558 (43.01%)	0.35 ± 0.21	0.13 ± 0.04	-0.01 ± 0.04	0.25 ± 0.14	**0.12 ± 0.20**	**0.52 ± 0.31**	**0.87 ± 0.27**	0.89 ± 0.12	00:11:59
	DiCE (R)	**576/576 (100%)**	0.38 ± 0.19	**0.13 ± 0.02**	-0.01 ± 0.04	0.27 ± 0.14	0.15 ± 0.21	0.61 ± 0.29	0.83 ± 0.26	**0.90 ± 0.12**	**00:01:34**
	DiCE (G)	552/557 (99.10%)	**0.19 ± 0.11**	0.83 ± 0.03	**0.04 ± 0.04**	**0.10 ± 0.06**	0.19 ± 0.18	0.56 ± 0.24	0.65 ± 0.29	0.89 ± 0.10	00:25:38
SVM	CPILS	322/539 (59.74%)	0.38 ± 0.19	0.14 ± 0.03	-0.01 ± 0.04	0.26 ± 0.13	**0.06 ± 0.12**	**0.49 ± 0.32**	**0.88 ± 0.27**	0.89 ± 0.12	00:29:14
	DiCE (R)	**543/585 (92.82%)**	0.40 ± 0.19	**0.13 ± 0.02**	-0.01 ± 0.04	0.27 ± 0.14	0.06 ± 0.14	0.55 ± 0.30	0.83 ± 0.25	**0.90 ± 0.12**	**00:11:24**
	DiCE (G)	491/561 (87.52%)	**0.21 ± 0.12**	0.82 ± 0.02	**0.04 ± 0.04**	**0.10 ± 0.07**	0.09 ± 0.13	0.53 ± 0.24	0.64 ± 0.28	0.89 ± 0.10	00:13:30

Table 28: Report 5 overview Multi-Class CPILS (1v1 Mode) vs DiCE on Body Performance Dataset. The isolation forest score computed on the train set: **0.04 ± 0.04**

Thyroid

		SPEC [↑]	PROXIMITY [↓]		NON – PLAUSIBILITY [↓]		NON-ROBUSTNESS [↓]		DIVERSITY [↑]		TIME [↓]
BBOX	Method	Rate	Ddist	Dcount	IF	Dimpl	Drob1	Drob2	Ddiv1	Ddiv2	[hh:mm:ss]
XGB	CPILS	374/400 (93.50%)	0.52 ± 0.46	**0.09 ± 0.04**	0.04 ± 0.07	0.33 ± 0.33	**0.15 ± 0.19**	**0.51 ± 0.39**	1.32 ± 0.47	0.34 ± 0.07	01:20:47
	DiCE (R)	**582/582 (100%)**	0.69 ± 0.38	0.10 ± 0.03	0.02 ± 0.06	0.46 ± 0.28	0.20 ± 0.32	0.60 ± 0.43	**1.43 ± 0.40**	**0.40 ± 0.06**	**00:08:56**
	DiCE (G)	582/582 (100%)	0.43 ± 0.38	0.29 ± 0.02	**0.11 ± 0.05**	**0.04 ± 0.07**	0.37 ± 0.28	0.57 ± 0.36	0.90 ± 0.46	0.32 ± 0.05	00:13:12
RF	CPILS	313/463 (67.60%)	0.58 ± 0.44	**0.09 ± 0.03**	0.02 ± 0.08	0.35 ± 0.36	**0.17 ± 0.27**	**0.48 ± 0.40**	1.35 ± 0.43	0.36 ± 0.07	02:41:01
	DiCE (R)	**600/600 (100%)**	0.73 ± 0.36	0.10 ± 0.03	0.02 ± 0.06	0.47 ± 0.28	0.19 ± 0.31	0.60 ± 0.42	**1.43 ± 0.38**	**0.41 ± 0.07**	**00:10:23**
	DiCE (G)	600/600 (100%)	0.44 ± 0.39	0.29 ± 0.02	**0.11 ± 0.05**	**0.03 ± 0.07**	0.36 ± 0.29	0.56 ± 0.36	0.89 ± 0.45	0.32 ± 0.05	00:14:51
MLP	CPILS	206/426 (48.36%)	0.58 ± 0.50	**0.09 ± 0.01**	0.04 ± 0.07	0.34 ± 0.33	0.15 ± 0.21	**0.53 ± 0.42**	1.38 ± 0.49	0.36 ± 0.07	02:35:13
	DiCE (R)	**414/414 (100%)**	0.82 ± 0.33	0.09 ± 0.04	0.00 ± 0.06	0.56 ± 0.27	**0.09 ± 0.17**	0.59 ± 0.42	**1.47 ± 0.40**	**0.40 ± 0.07**	**00:07:21**
	DiCE (G)	414/414 (100%)	0.49 ± 0.39	0.30 ± 0.02	**0.10 ± 0.05**	**0.04 ± 0.08**	0.25 ± 0.22	0.57 ± 0.35	0.85 ± 0.45	0.31 ± 0.05	00:11:49
SVM	CPILS	74/272 (27.21%)	0.85 ± 0.26	**0.07 ± 0.03**	0.01 ± 0.05	0.57 ± 0.30	**0.02 ± 0.08**	**0.20 ± 0.36**	**1.42 ± 0.36**	0.33 ± 0.06	02:28:27
	DiCE (R)	**162/288 (40.62%)**	0.92 ± 0.31	0.09 ± 0.02	-0.01 ± 0.06	0.57 ± 0.23	0.03 ± 0.16	0.28 ± 0.37	1.37 ± 0.45	**0.39 ± 0.07**	**00:06:32**
	DiCE (G)	44/288 (15.28%)	**0.41 ± 0.27**	0.29 ± 0.01	**0.11 ± 0.04**	**0.04 ± 0.08**	0.02 ± 0.14	0.27 ± 0.14	0.92 ± 0.45	0.32 ± 0.05	00:07:50

Table 29: Report 5 overview Multi-Class CPILS (1v1 Mode) vs DiCE on Thyroid Dataset. The isolation forest score computed on the train set: **0.12 ± 0.06**

Wine

		SPEC [↑]	PROXIMITY [↓]		NON – PLAUSIBILITY [↓]		NON-ROBUSTNESS [↓]		DIVERSITY [↑]		TIME [↓]
BBOX	Method	Rate	Ddist	Dcount	IF	Dimpl	Drob1	Drob2	Ddiv1	Ddiv2	[hh:mm:ss]
XGB	CPILS	238/321 (52.34%)	0.61 ± 0.40	0.18 ± 0.06	-0.01 ± 0.06	0.40 ± 0.30	**0.17 ± 0.22**	**0.50 ± 0.26**	**1.07 ± 0.44**	0.90 ± 0.10	**02:10**
	DiCE (R)	**395/395 (100%)**	0.59 ± 0.27	**0.15 ± 0.03**	-0.03 ± 0.05	0.41 ± 0.20	0.24 ± 0.28	0.54 ± 0.26	1.00 ± 0.32	**0.92 ± 0.09**	11:04
	DiCE (G)	388/409 (94,87%)	**0.22 ± 0.21**	0.91 ± 0.02	**0.03 ± 0.06**	**0.15 ± 0.15**	0.41 ± 0.36	0.65 ± 0.25	0.69 ± 0.38	0.91 ± 0.08	49:15
RF	CPILS	116/292 (39.73%)	**0.55 ± 0.33**	**0.17 ± 0.07**	**-0.02 ± 0.05**	**0.35 ± 0.21**	**0.14 ± 0.21**	**0.41 ± 0.24**	1.02 ± 0.37	0.91 ± 0.10	**06:21**
	DiCE (R)	**356/356 (100%)**	0.63 ± 0.30	0.18 ± 0.05	-0.03 ± 0.06	0.42 ± 0.23	0.25 ± 0.27	0.48 ± 0.25	**1.02 ± 0.34**	**0.92 ± 0.09**	21:19
	DiCE (G)	-	-	-	-	-	-	-	-	-	More then hour
MLP	CPILS	164/287 (57.14%)	0.53 ± 0.32	0.15 ± 0.05	0.00 ± 0.05	0.35 ± 0.21	**0.05 ± 0.11**	**0.54 ± 0.32**	1.05 ± 0.39	**0.92 ± 0.11**	**00:58**
	DiCE (R)	**318/318 (100%)**	0.64 ± 0.25	**0.13 ± 0.03**	-0.02 ± 0.05	0.43 ± 0.21	0.06 ± 0.13	0.57 ± 0.35	**1.06 ± 0.35**	0.91 ± 0.11	05:28
	DiCE (G)	306/306 (100%)	**0.35 ± 0.33**	0.91 ± 0.02	**0.01 ± 0.08**	**0.25 ± 0.25**	0.11 ± 0.15	0.70 ± 0.28	0.86 ± 0.42	0.91 ± 0.09	01:20
SVM	CPILS	**149/171 (97.66%)**	**0.53 ± 0.33**	**0.13 ± 0.04**	**0.00 ± 0.06**	**0.37 ± 0.24**	0.04 ± 0.09	0.44 ± 0.32	0.97 ± 0.37	0.90 ± 0.11	**06:46**
	DiCE (R)	207/234 (88.46%)	0.62 ± 0.27	0.14 ± 0.03	-0.03 ± 0.05	0.43 ± 0.22	**0.03 ± 0.08**	**0.43 ± 0.30**	**1.02 ± 0.32**	**0.91 ± 0.11**	07:04
	DiCE (G)	-	-	-	-	-	-	-	-	-	More then hour

Table 30: Report 5 overview Multi-Class CPILS (1v1 Mode) vs DiCE on Wine Dataset. The isolation forest score computed on train set: **0.05 ± 0.04**. The DiCE (Genetic) algorithm gets stuck in two cases (RF and SVM), leading to excessive computation time. As a result, the outcomes for these specific cases are not reported. This issue has been observed occasionally, as noted by users in the DiCE repository forum.

Subject 28

		SPEC [↑]	PROXIMITY [↓]		NON – PLAUSIBILITY [↓]		NON-ROBUSTNESS [↓]		DIVERSITY [↑]		TIME [↓]
BBOX	Method	Rate	Ddist	Dcount	IF	Dimpl	Drob1	Drob2	Ddiv1	Ddiv2	[hh:mm:ss]
XGB	CPILS	137/518 (26.45%)	0.78 ± 0.39	0.12 ± 0.05	0.04 ± 0.04	0.49 ± 0.28	**0.19 ± 0.24**	**0.44 ± 0.46**	**1.14 ± 0.37**	0.92 ± 0.10	00:22:50
	DiCE (R)	**600/600 (100%)**	0.72 ± 0.31	**0.10 ± 0.02**	0.02 ± 0.05	0.53 ± 0.25	0.21 ± 0.24	0.65 ± 0.45	1.06 ± 0.31	**0.93 ± 0.10**	00:17:26
	DiCE (G)	532/600 (88.67%)	**0.18 ± 0.13**	0.89 ± 0.09	**0.08 ± 0.03**	**0.08 ± 0.07**	0.44 ± 0.48	0.48 ± 0.41	0.83 ± 0.33	0.67 ± 0.08	**00:01:08**
RF	CPILS	279/526 (53.04%)	0.79 ± 0.38	0.19 ± 0.06	0.03 ± 0.04	0.52 ± 0.29	0.28 ± 0.34	0.45 ± 0.47	**1.22 ± 0.38**	0.92 ± 0.10	00:45:39
	DiCE (R)	**600/600 (100%)**	0.75 ± 0.34	**0.12 ± 0.03**	0.01 ± 0.06	0.56 ± 0.29	**0.22 ± 0.29**	0.61 ± 0.46	1.16 ± 0.34	**0.93 ± 0.10**	00:26:09
	DiCE (G)	503/598 (100%)	**0.17 ± 0.14**	0.89 ± 0.09	**0.08 ± 0.03**	**0.07 ± 0.05**	0.47 ± 0.44	**0.42 ± 0.45**	0.82 ± 0.34	0.65 ± 0.08	**00:01:25**
MLP	CPILS	253/546 (46.34%)	0.47 ± 0.29	0.13 ± 0.05	0.05 ± 0.04	0.33 ± 0.24	**0.09 ± 0.15**	**0.24 ± 0.36**	1.07 ± 0.37	0.91 ± 0.11	00:10:07
	DiCE (R)	**546/546 (100%)**	0.68 ± 0.28	**0.10 ± 0.01**	0.01 ± 0.04	0.55 ± 0.24	0.09 ± 0.18	0.66 ± 0.45	**1.20 ± 0.30**	**0.94 ± 0.10**	00:04:26
	DiCE (G)	379/459 (82.57%)	**0.35 ± 0.28**	0.89 ± 0.09	**0.07 ± 0.04**	**0.16 ± 0.18**	0.20 ± 0.24	0.58 ± 0.44	0.87 ± 0.34	0.67 ± 0.08	**00:02:32**
SVM	CPILS	308/481 (64.03%)	0.63 ± 0.27	0.11 ± 0.04	0.04 ± 0.04	0.46 ± 0.25	0.08 ± 0.19	0.55 ± 0.47	1.17 ± 0.33	0.91 ± 0.11	00:16:50
	DiCE (R)	**451/486 (92.80%)**	0.73 ± 0.29	**0.10 ± 0.01**	0.02 ± 0.04	0.55 ± 0.25	**0.04 ± 0.14**	0.48 ± 0.47	**1.20 ± 0.30**	**0.93 ± 0.10**	**00:11:33**
	DiCE (G)	326/360 (90.56%)	**0.38 ± 0.28**	0.91 ± 0.08	**0.07 ± 0.03**	**0.17 ± 0.18**	0.08 ± 0.16	**0.24 ± 0.36**	0.85 ± 0.34	0.66 ± 0.08	01:14:00

Table 31: Report 5 overview Multi-Class CPILS (1v1 Mode) vs DiCE on KEmoCon Subject 28 Dataset. The isolation forest score computed on the train set: **0.05 ± 0.04**.

Subject 17

		SPEC [↑]	PROXIMITY [↓]		NON – PLAUSIBILITY [↓]		NON-ROBUSTNESS [↓]		DIVERSITY [↑]		TIME [↓]
BBOX	Method	Rate	Ddist	Dcount	IF	Dimpl	Drob1	Drob2	Ddiv1	Ddiv2	[hh:mm:ss]
XGB	CPILS	372/880 (42.27%)	0.58 ± 0.37	0.15 ± 0.05	0.04 ± 0.07	0.40 ± 0.26	0.53 ± 0.46	0.66 ± 0.39	**1.05 ± 0.33**	**0.91 ± 0.09**	00:58:37
	DiCE (R)	**1054/1054 (100%)**	0.60 ± 0.33	**0.11 ± 0.03**	0.04 ± 0.06	0.42 ± 0.26	**0.46 ± 0.50**	0.69 ± 0.38	1.01 ± 0.33	0.90 ± 0.10	00:22:16
	DiCE (G)	1080/1199 (90.08%)	**0.38 ± 0.22**	0.88 ± 0.14	**0.09 ± 0.04**	**0.11 ± 0.18**	0.61 ± 0.49	**0.30 ± 0.33**	0.68 ± 0.34	0.72 ± 0.11	**00:01:36**
RF	CPILS	314/1067 (29.43%)	0.59 ± 0.33	0.23 ± 0.10	0.05 ± 0.06	0.38 ± 0.25	0.61 ± 0.57	0.55 ± 0.40	0.97 ± 0.34	0.90 ± 0.09	01:33:05
	DiCE (R)	**1082/1082 (100%)**	0.77 ± 0.37	**0.16 ± 0.05**	0.02 ± 0.08	0.53 ± 0.29	0.63 ± 0.63	0.68 ± 0.37	**1.15 ± 0.36**	**0.91 ± 0.09**	00:24:38
	DiCE (G)	503/598 (100%)	**0.38 ± 0.22**	0.88 ± 0.14	**0.09 ± 0.04**	**0.11 ± 0.16**	**0.61 ± 0.54**	**0.30 ± 0.34**	0.68 ± 0.34	0.72 ± 0.11	**00:02:03**
MLP	CPILS	341/945 (36.08%)	0.54 ± 0.28	0.12 ± 0.05	0.06 ± 0.06	0.36 ± 0.23	**0.10 ± 0.21**	**0.24 ± 0.40**	0.98 ± 0.35	0.90 ± 0.11	01:09:22
	DiCE (R)	**960/960 (100%)**	0.72 ± 0.26	**0.10 ± 0.02**	0.02 ± 0.06	0.56 ± 0.25	0.11 ± 0.25	0.73 ± 0.37	**1.16 ± 0.31**	**0.91 ± 0.10**	00:06:39
	DiCE (G)	787/953 (82.58%)	**0.45 ± 0.25**	0.88 ± 0.14	**0.09 ± 0.04**	**0.14 ± 0.24**	0.17 ± 0.30	0.54 ± 0.31	0.71 ± 0.35	0.72 ± 0.11	**00:01:43**
SVM	CPILS	296/766 (38.64%)	0.60 ± 0.25	0.12 ± 0.04	0.07 ± 0.05	0.34 ± 0.20	0.10 ± 0.27	0.44 ± 0.42	0.95 ± 0.31	0.89 ± 0.12	01:08:42
	DiCE (R)	**658/815 (80.74%)**	0.72 ± 0.26	**0.11 ± 0.03**	0.04 ± 0.05	0.49 ± 0.26	**0.06 ± 0.21**	0.51 ± 0.43	**1.07 ± 0.30**	**0.90 ± 0.11**	00:06:34
	DiCE (G)	656/818 (80.20%)	**0.43 ± 0.20**	0.87 ± 0.14	**0.10 ± 0.03**	**0.09 ± 0.12**	0.13 ± 0.29	**0.18 ± 0.27**	0.61 ± 0.29	0.70 ± 0.11	**00:03:38**

Table 32: Report 5 overview Multi-Class CPILS (1v1 Mode) vs DiCE on KEmoCon Subject 17 Dataset. The isolation forest score computed on the train set: **0.09 ± 0.05**

Subject 5

		SPEC [↑]	PROXIMITY [↓]		NON – PLAUSIBILITY [↓]		NON-ROBUSTNESS [↓]		DIVERSITY [↑]		TIME [↓]
BBOX	Method	Rate	Ddist	Dcount	IF	Dimpl	Drob1	Drob2	Ddiv1	Ddiv2	[hh:mm:ss]
XGB	CPILS	219/522 (41.95%)	0.54 ± 0.32	**0.12 ± 0.05**	0.05 ± 0.05	0.35 ± 0.25	**0.25 ± 0.29**	0.59 ± 0.45	1.02 ± 0.37	0.98 ± 0.08	00:43:46
	DiCE (R)	**886/ 886 (100%)**	0.74 ± 0.37	0.14 ± 0.04	0.01 ± 0.07	0.52 ± 0.30	0.28 ± 0.36	0.56 ± 0.45	**1.15 ± 0.36**	**0.99 ± 0.08**	00:09:37
	DiCE (G)	777/ 778 (99.87%)	0.51 ± 0.29	0.98 ± 0.04	**0.06 ± 0.04**	**0.19 ± 0.19**	0.46 ± 0.36	**0.37 ± 0.39**	0.72 ± 0.31	0.96 ± 0.06	**00:06:24**
RF	CPILS	329/846 (38.89%)	0.73 ± 0.34	0.19 ± 0.06	0.03 ± 0.05	0.44 ± 0.26	**0.16 ± 0.25**	0.48 ± 0.44	1.11 ± 0.36	0.97 ± 0.08	01:16:22
	DiCE (R)	**829/829 (100%)**	0.84 ± 0.41	**0.18 ± 0.07**	0.00 ± 0.07	0.53 ± 0.31	0.25 ± 0.32	0.43 ± 0.45	**1.16 ± 0.36**	**0.99 ± 0.07**	00:13:14
	DiCE (G)	813/827 (98.31%)	**0.51 ± 0.29**	0.98 ± 0.03	**0.06 ± 0.04**	**0.20 ± 0.20**	0.43 ± 0.35	**0.38 ± 0.39**	0.73 ± 0.30	0.96 ± 0.06	**00:03:30**
MLP	CPILS	328/793 (41.36%)	0.62 ± 0.30	0.14 ± 0.04	0.03 ± 0.05	0.40 ± 0.22	0.09 ± 0.19	0.52 ± 0.47	1.11 ± 0.34	0.98 ± 0.08	01:28:17
	DiCE (R)	**826/826 (100%)**	0.77 ± 0.29	**0.11 ± 0.03**	0.01 ± 0.05	0.59 ± 0.26	**0.08 ± 0.19**	0.62 ± 0.45	**1.18 ± 0.32**	**0.99 ± 0.08**	00:05:15
	DiCE (G)	723/749 (96.53%)	**0.52 ± 0.32**	0.98 ± 0.04	**0.07 ± 0.04**	**0.18 ± 0.16**	0.18 ± 0.23	**0.35 ± 0.38**	0.72 ± 0.33	0.96 ± 0.06	**00:03:07**
SVM	CPILS	352/688 (51.16%)	0.71 ± 0.30	0.14 ± 0.05	0.04 ± 0.04	0.44 ± 0.22	**0.03 ± 0.11**	0.44 ± 0.46	1.12 ± 0.30	0.97 ± 0.08	01:15:26
	DiCE (R)	**656/788 (83.25%)**	0.77 ± 0.22	**0.11 ± 0.03**	0.01 ± 0.05	0.55 ± 0.23	0.04 ± 0.12	0.48 ± 0.46	**1.12 ± 0.34**	**0.98 ± 0.08**	**00:07:41**
	DiCE (G)	580/728 (80.20%)	**0.58 ± 0.27**	0.99 ± 0.03	**0.06 ± 0.04**	**0.22 ± 0.19**	0.05 ± 0.12	**0.28 ± 0.37**	0.75 ± 0.29	0.96 ± 0.06	00:14:14

Table 33: Report 5 overview Multi-Class CPILS (1v1 Mode) vs DiCE on KEmoCon Subject 5 Dataset. The isolation forest score computed on the train set: **0.06 ± 0.05**

Considerations

From these experimental results emerge what are the advantages and drawbacks to use this new approach.

- In terms of **specificity DiCE clearly wins** in both absolute numbers and percentages. The reason for this outcome is that it does not rely on any binary approximation; instead, it generates counterfactuals through random or genetic manipulation to match the desired target class. In contrast, CPILS may produce counterfactuals that, while different from the original class and still acceptable, have less specificity since they do not always satisfy the prediction criteria of a multi-class black-box model (defined previously as plausible counterfactuals).

- From the perspective of **proximity** and **plausibility**, **CPILS** emerges as a **balanced middle ground** between DiCE (random) and DiCE (genetic). In general, **DiCE (genetic)** performs **better** in terms of **Ddist** and, consequently, **IF and Dimpl**, as its optimization-driven approach generates counterfactuals that are very close to the original instance. However, this often comes at the **cost** of modifying a large number of features to satisfy the criteria, as reflected in its **high Dcount metric**. This trade-off can **negatively impact user interpretability**, as changes affecting many features are harder to comprehend. Notably, IF and Dimpl, while representing different concepts, tend to correlate closely. On the other hand, **DiCE (random)** produces counterfactuals that are **less minimal** due to its stochastic nature, often resulting in **less plausible** values but **minimizing the number of features changed**. CPILS, in most cases, positions itself between these two approaches, striving to balance proximity and plausibility. In certain cases, such as with the **Thyroid** dataset, where the number of features is very high, CPILS performs better than DiCE (random), achieving counterfactuals with greater proximity or plausibility.

- The **robustness** metric is where **CPILS excels the most** across the majority of datasets and all black-box classifiers. The use of this particular technique, which relies on the binarization of multi-class problems through an ensemble of latent spaces, ensures a higher level of robustness compared to the DiCE methods. The fact that CPILS generates fewer counterfactuals, which are also less minimal than those produced by DiCE (genetic) and, in some cases, less plausible, is compensated by its strong robustness. These counterfactuals are **less susceptible to perturbations** and **adversarial attacks**, as indicated by the Drob1 and Drob2 metrics. There are a few exceptions, such as the Wine dataset with the SVM classifier, where CPILS does not maintain the same level of robustness. However, in these cases, it compensates with higher minimality

and plausibility. In general, the challenge in generating counterfactuals lies in balancing the distance from the original sample and the proximity to samples of different classes. Often, vectors closer to the original sample tend to be more adversarial, while those further away are less so. This property is **not** always **remarked** for **KEmoCon datasets**, the next report shows its impact **vary the latent space dimension**.

- The **diversity** metric indicates that the most varied set of generated counterfactuals, both in terms of distance and the number of different features modified, is achieved using **CPILS** or **DiCE (random)**. In contrast, **DiCE (genetic)**, due to its optimization-driven nature, consistently **produces fewer diverse results**, generating counterfactuals that are highly similar to each other or that involve fewer feature changes. **Higher diversity values** in CPILS and DiCE (random) suggest a **broader variation in the generated counterfactuals**. In some cases, CPILS significantly outperforms DiCE (random), which is particularly notable given that one of DiCE's primary objectives is to maximize diversity. This highlights another **key advantage** of using **CPILS**.

- The **time** is the most **variable metric**, as seen in previous reports, and it depends on each specific case. In most scenarios, **CPILS** is the **most time-consuming** method compared to both versions of DiCE. This is primarily because generating counterfactuals with CPILS requires **more mathematical operations in the latent space**, along with the need to map the discovered minimal changes back to the original space by solving the Lagrange Multiplier method. Additionally, performance is influenced by the quality of the dataset and the structure of the created latent space. In contrast, **DiCE operates directly in the input space**, without involving a latent space. There are instances where the random mode is slower than the genetic approach, likely due to the dataset's characteristics. When a dataset has many features, random sampling struggles to find valid counterfactuals. On the other hand, genetic algorithms can refine solutions by reducing unnecessary trials and learning which mutations work, leading to faster convergence. The only case where **CPILS outperforms** all black-box classifiers in terms of time is with the **Wine dataset**.

5.4.6 Report 6: Multi-Class CPILS - multi latent dimensions vs DiCE

In the previous report, the robustness metrics in the multi-class CPILS method appeared to outperform DiCE on the Benchmark dataset, even when using a latent space dimension of just 2. However, this trend was not as pronounced for the KEmoCon dataset. The goal of the current experimentation is to assess how varying the latent space dimension influences the previously introduced metrics, particularly robustness. To achieve this, the same experimental setup is repeated for two specific subjects in the KEmoCon dataset (Subject 27 and Subject 5), with latent space dimensions of 2, 3, and 5 being tested for the multi-class CPILS method.

Subject 27

		SPEC [↑]	PROXIMITY [↓]		NON – PLAUSIBILITY [↓]		NON-ROBUSTNESS [↓]		DIVERSITY [↑]		TIME [↓]
BBOX	Method	Rate	Ddist	Dcount	IF	Dimpl	Drob1	Drob2	Ddiv1	Ddiv2	[mm:ss]
XGB	CPILS_2	261/343 (76.09%)	0.68 ± 0.41	**0.12 ± 0.04**	0.04 ± 0.06	0.46 ± 0.30	0.20 ± 0.24	0.57 ± 0.48	1.19 ± 0.41	0.88 ± 0.18	38:02
	CPILS_3	393/460 (85.43%)	0.75 ± 0.44	0.15 ± 0.03	0.02 ± 0.06	0.53 ± 0.35	0.15 ± 0.23	**0.50 ± 0.48**	1.25 ± 0.39	0.88 ± 0.17	35:23
	CPILS_5	300/365 (82.19%)	0.79 ± 0.40	0.12 ± 0.05	0.03 ± 0.06	0.54 ± 0.32	**0.14 ± 0.22**	0.57 ± 0.48	1.25 ± 0.38	0.86 ± 0.19	35:12
	DiCE (R)	**199/199 (100%)**	0.73 ± 0.38	0.15 ± 0.12	-0.01 ± 0.07	0.56 ± 0.32	0.15 ± 0.21	**0.65 ± 0.46**	**1.30 ± 0.41**	**0.90 ± 0.18**	**10:07**
	DiCE (G)	336/342 (98.25%)	**0.50 ± 0.34**	0.82 ± 0.19	**0.05 ± 0.04**	**0.27 ± 0.25**	**0.43 ± 0.19**	0.54 ± 0.47	0.98 ± 0.38	0.66 ± 0.09	10:30
RF	CPILS_2	257/556 (46.22%)	0.89 ± 0.39	**0.17 ± 0.05**	0.01 ± 0.06	0.61 ± 0.35	0.21 ± 0.29	**0.27 ± 0.43**	1.32 ± 0.41	0.90 ± 0.15	52:07
	CPILS_3	239/580 (41.21%)	0.90 ± 0.35	0.17 ± 0.08	0.01 ± 0.06	0.68 ± 0.33	0.27 ± 0.33	0.42 ± 0.48	1.40 ± 0.41	0.86 ± 0.17	49:31
	CPILS_5	233/561 (41.53%)	0.68 ± 0.36	0.19 ± 0.08	0.02 ± 0.06	0.48 ± 0.28	0.27 ± 0.28	0.41 ± 0.48	1.20 ± 0.39	0.88 ± 0.15	53:01
	DiCE (R)	**240/240 (100%)**	0.91 ± 0.37	0.17 ± 0.09	**-0.04 ± 0.07**	**0.75 ± 0.34**	**0.12 ± 0.20**	0.52 ± 0.46	**1.45 ± 0.40**	**0.92 ± 0.15**	**04:13**
	DiCE (G)	375/398 (94.22%)	**0.56 ± 0.36**	0.84 ± 0.19	**0.05 ± 0.04**	**0.35 ± 0.30**	**0.65 ± 0.40**	**0.61 ± 0.46**	1.01 ± 0.36	0.64 ± 0.10	07:15
MLP	CPILS_2	228/594 (38.38%)	0.64 ± 0.27	0.12 ± 0.04	0.04 ± 0.05	0.43 ± 0.20	0.05 ± 0.14	**0.52 ± 0.47**	1.15 ± 0.34	0.89 ± 0.17	17:09
	CPILS_3	282/594 (47.47%)	0.65 ± 0.26	0.12 ± 0.04	0.04 ± 0.04	0.47 ± 0.22	0.06 ± 0.15	0.53 ± 0.47	1.16 ± 0.32	0.87 ± 0.17	14:29
	CPILS_5	236/594 (39.73%)	0.60 ± 0.25	0.12 ± 0.05	0.04 ± 0.04	0.43 ± 0.21	**0.05 ± 0.11**	0.54 ± 0.47	1.13 ± 0.32	0.86 ± 0.18	14:39
	DiCE (R)	**594/594 (100%)**	0.79 ± 0.26	**0.10 ± 0.01**	-0.01 ± 0.05	0.69 ± 0.26	0.05 ± 0.15	**0.82 ± 0.38**	**1.36 ± 0.34**	**0.93 ± 0.14**	**02:05**
	DiCE (G)	412/471 (87.47%)	**0.51 ± 0.28**	0.85 ± 0.18	**0.06 ± 0.04**	**0.30 ± 0.21**	**0.09 ± 0.18**	0.55 ± 0.46	0.91 ± 0.31	0.68 ± 0.08	03:07
SVM	CPILS_2	344/559 (61.54%)	0.69 ± 0.30	**0.11 ± 0.05**	0.04 ± 0.05	0.43 ± 0.18	0.03 ± 0.09	**0.45 ± 0.48**	1.15 ± 0.34	0.90 ± 0.16	32:09
	CPILS_3	331/569 (58.17%)	0.71 ± 0.29	0.14 ± 0.06	0.04 ± 0.05	0.49 ± 0.20	0.03 ± 0.11	0.51 ± 0.48	1.18 ± 0.32	0.90 ± 0.15	30:56
	CPILS_5	333/568 (58.63%)	0.69 ± 0.31	0.12 ± 0.05	0.04 ± 0.05	0.46 ± 0.18	**0.02 ± 0.07**	0.48 ± 0.48	1.14 ± 0.32	0.91 ± 0.15	31:01
	DiCE (R)	**508/537 (94.60%)**	0.84 ± 0.34	0.15 ± 0.05	0.00 ± 0.06	0.58 ± 0.25	0.02 ± 0.08	**0.71 ± 0.43**	**1.24 ± 0.33**	**0.93 ± 0.13**	**03:19**
	DiCE (G)	477/534 (89.33%)	**0.62 ± 0.33**	0.85 ± 0.17	**0.04 ± 0.04**	**0.38 ± 0.25**	0.03 ± 0.11	0.63 ± 0.46	0.99 ± 0.32	0.65 ± 0.09	10:27

Table 34: Report 6 overview: multi-class CPILS using latent space dimension (2,3 and 5) vs DiCE (random and genetic) for KEmoCon Subject 27 dataset. In bold are remarked the best result obtained according to the metric, in red are highlighted the worse values that justify CPILS as the better solution for that specific metric. The underlined value is considered the second-best choice.

Subject 5

		SPEC [↑]	PROXIMITY [↓]		NON – PLAUSIBILITY [↓]		NON-ROBUSTNESS [↓]		DIVERSITY [↑]		TIME [↓]
BBOX	Method	Rate	Ddist	Dcount	IF	Dimpl	Drob1	Drob2	Ddiv1	Ddiv2	[hh:mm:ss]
XGB	CPILS_2	219/522 (41.95%)	0.54 ± 0.32	**0.12 ± 0.05**	0.05 ± 0.05	0.35 ± 0.25	0.25 ± 0.29	**0.59 ± 0.45**	1.02 ± 0.37	0.98 ± 0.08	00:43:46
	CPILS_3	292/593 (49.24%)	0.62 ± 0.33	0.15 ± 0.07	0.03 ± 0.05	0.42 ± 0.27	**0.18 ± 0.24**	0.55 ± 0.45	1.11 ± 0.37	0.97 ± 0.08	00:42:43
	CPILS_5	308/650 (47.38%)	0.66 ± 0.31	0.16 ± 0.07	0.03 ± 0.05	0.38 ± 0.23	0.21 ± 0.26	0.54 ± 0.45	1.09 ± 0.36	0.98 ± 0.08	00:43:11
	DiCE (R)	**886/ 886 (100%)**	0.74 ± 0.37	0.14 ± 0.04	0.01 ± 0.07	0.52 ± 0.30	0.28 ± 0.36	0.56 ± 0.45	**1.15 ± 0.36**	**0.99 ± 0.08**	00:09:37
	DiCE (G)	777/ 778 (99.87%)	**0.51 ± 0.29**	0.98 ± 0.04	**0.06 ± 0.04**	**0.19 ± 0.19**	**0.46 ± 0.36**	**0.37 ± 0.39**	0.72 ± 0.31	0.96 ± 0.06	**00:06:24**
RF	CPILS_2	329/846 (38.89%)	0.73 ± 0.34	0.19 ± 0.06	0.03 ± 0.05	0.44 ± 0.26	0.16 ± 0.25	0.48 ± 0.44	1.11 ± 0.36	0.97 ± 0.08	01:16:22
	CPILS_3	338/816 (41.42%)	0.83 ± 0.48	0.20 ± 0.09	0.02 ± 0.06	0.49 ± 0.33	0.16 ± 0.26	0.41 ± 0.44	1.20 ± 0.41	0.97 ± 0.08	01:15:27
	CPILS_5	347/847 (40.97%)	0.84 ± 0.46	0.20 ± 0.08	0.02 ± 0.06	0.53 ± 0.35	**0.15 ± 0.24**	**0.48 ± 0.45**	**1.26 ± 0.43**	0.97 ± 0.08	01:14:11
	DiCE (R)	**829/829 (100%)**	0.84 ± 0.41	**0.18 ± 0.07**	0.00 ± 0.07	0.53 ± 0.31	0.25 ± 0.32	0.43 ± 0.45	1.16 ± 0.36	**0.99 ± 0.07**	00:13:14
	DiCE (G)	813/827 (98.31%)	**0.51 ± 0.29**	0.98 ± 0.03	**0.06 ± 0.04**	**0.20 ± 0.20**	**0.43 ± 0.35**	**0.38 ± 0.39**	0.73 ± 0.30	0.96 ± 0.06	**00:03:30**
MLP	CPILS_2	328/793 (41.36%)	0.62 ± 0.30	0.14 ± 0.04	0.03 ± 0.05	0.40 ± 0.22	0.09 ± 0.19	0.52 ± 0.47	1.11 ± 0.34	0.98 ± 0.08	01:28:17
	CPILS_3	323/815 (39.63%)	0.66 ± 0.29	0.16 ± 0.05	0.02 ± 0.05	0.42 ± 0.23	**0.08 ± 0.17**	0.52 ± 0.47	1.15 ± 0.33	0.98 ± 0.08	00:49:22
	CPILS_5	333/830 (40.12%)	0.62 ± 0.31	0.15 ± 0.05	0.03 ± 0.05	0.41 ± 0.24	0.09 ± 0.18	0.61 ± 0.46	1.11 ± 0.35	0.98 ± 0.08	00:40:36
	DiCE (R)	**826/826 (100%)**	0.77 ± 0.29	**0.11 ± 0.03**	0.01 ± 0.05	0.59 ± 0.26	0.08 ± 0.19	**0.62 ± 0.45**	**1.18 ± 0.32**	**0.99 ± 0.08**	00:05:15
	DiCE (G)	723/749 (96.53%)	**0.52 ± 0.32**	0.98 ± 0.04	**0.07 ± 0.04**	**0.18 ± 0.16**	**0.18 ± 0.23**	**0.35 ± 0.38**	0.72 ± 0.33	0.96 ± 0.06	**00:03:07**
SVM	CPILS_2	352/688 (51.16%)	0.71 ± 0.30	0.14 ± 0.05	0.04 ± 0.04	0.44 ± 0.22	**0.03 ± 0.11**	0.44 ± 0.46	1.12 ± 0.30	0.97 ± 0.08	01:15:26
	CPILS_3	301/655 (45.95%)	0.77 ± 0.28	0.15 ± 0.04	0.03 ± 0.04	0.47 ± 0.24	0.03 ± 0.11	0.38 ± 0.45	**1.21 ± 0.34**	0.97 ± 0.09	01:10:08
	CPILS_5	366/758 (48.28%)	0.77 ± 0.34	0.17 ± 0.05	0.02 ± 0.05	0.47 ± 0.27	0.03 ± 0.11	0.38 ± 0.45	1.20 ± 0.35	0.97 ± 0.09	01:14:34
	DiCE (R)	656/788 (83.25%)	0.77 ± 0.22	**0.11 ± 0.03**	0.01 ± 0.05	0.55 ± 0.23	0.04 ± 0.12	**0.48 ± 0.46**	1.12 ± 0.34	**0.98 ± 0.08**	**00:07:41**
	DiCE (G)	580/728 (80.20%)	**0.58 ± 0.27**	0.99 ± 0.03	**0.06 ± 0.04**	**0.22 ± 0.19**	**0.05 ± 0.12**	**0.28 ± 0.37**	0.75 ± 0.29	0.96 ± 0.06	00:14:14

Table 35: Report 6 overview: multi-class CPILS using latent space dimension (2,3 and 5) vs DiCE (random and genetic) for KEmoCon Subject 5 dataset. In bold are remarked the best result obtained according to the metric, and in red are highlighted the worse values of non-robustness. The underlined value is considered the second-best choice.

Considerations

The analysis of the two subjects highlights the importance of selecting the appropriate latent space dimensions. The most significant insights from this experiment are presented below.

- In terms of specificity rate **DiCE clearly still wins** unless in percentage number. Indeed, for Subject 27, the absolute **number of generated counterfactuals** is greater than in **CPILS** using **latent dimension equal to 3** with respect to DiCE (Genetic) implementation. Instead, it wins in all of the three latent dimensions proposed with respect to DiCE random modality. The number of total counterfactuals is not to be misunderstood; this experiment is focused only on **specificity**, but however CPILS is also capable of producing plausible counterfactuals, not taken into consideration here.

- For **proximity** and **plausibility**, the considerations concluded for Report 5 are still preserved, confirming that **multi-class CPILS** still remains a valid **middle solution** between DiCE random and genetic approaches.

- The impact of the variation of latent space dimensions for **robustness** metric is more evident for **Subject 27**. In this dataset, multi-class CPILS is able to outperform in terms of robustness in **all the presented cases** in **both** the robustness **metrics** proposed with very **high gaps** especially on **Drob2**, the most suitable from these two. Considering Drob1, the only case where this metric is worse is when using the RF classifier; however, the level of implausibility is very high, which leads to the second best result using CPILS with latent dimension equal to 2. It is important to note that, using **XGB**, the only case where CPILS performs better in **Drob2 metric** is only using **latent dimension equal to 3**. This also corresponds to a **higher level of specificity** and remarks on the importance of a proper selection of the latent dimension. In **Subject 5**, the results are **less pronounced** but still emphasize the importance of selecting an appropriate latent dimension. Based on the Drob1 metric, CPILS emerges as the best solution, while for Drob2, it ranks second. However, it still captures a significant variation in value when the latent dimension is chosen correctly.

- The other metrics (diversity and execution time) do not highlight any interesting pattern. The considerations from the previous report remain applicable.

In general, the choice of latent space dimension in CPILS is crucial and has a significant impact. It should be wisely selected, as increasing this value can reduce anti-robustness metrics and enhance the generation of more robust counterfactuals.

6 Conclusions and Future Works

In this work, a critical analysis has been conducted by analyzing the advantages and disadvantages of using this new proposed approach by considering different scenarios from multiple points of view. The working modalities have, of course, an important role in the type of counterfactuals search involving different latent spaces and types of counterfactuals, such as the latent dimension used. The two main modalities considered are One-vs-One and One-vs-Rest because they are the ones that discard fewer counterfactuals as they provide a more efficient computing time and generate closer counterfactuals. However, even they can take other interesting information that can be exploited in future works. In this analysis, to explain a multi-class classification problem, several binary black box models with the same exact characteristics of the original one are trained and employed in a binarized approach exploiting CPILS. This leave opened to new different strategies that may see a model with different characteristics used as a surrogate model.

The strengths and limitations identified in the comprehensive analysis are summarized below in key points.

Strengths

- **Simplicity**: this approach builds upon the original CPILS method, initially designed for binary classification, and extends it to a multi-class scenario. It leverages the foundational principles validated in previous work, preserving the benefits of the original intuitive idea. The method aims to approximate the behavior of the multi-class black-box model through counterfactual explanations, utilizing an ensemble of latent spaces. Each latent space independently applies CPILS to search for counterfactuals, ensuring a more robust and interpretable approximation. In this case, it leverages the interpretability established for CPILS to provide explanations.
- **Flexibility and Customizability**: the method offers a wide range of functionalities to give the users the flexibility to choose depending on their specific situation or problem. These flexible features can be summarized as follows:
  - enables explanation for any black-box model (model-agnostic).
  - ensuring actionability allowing the users to select specific combinations of features to be changed in counterfactual search and what to maintain fixed
  - permit to specify the maximum number of features to change in counterfactual search

  - selection of a specific modality of search
  - choosing a specific latent space dimensionality
  - specifying the target counterfactual classes, limiting the search on specific latent spaces
  - filtering (as post-processing technique) the counterfactuals applying (ranking, grouping and limiting the number)

- **Parallelization**: although not explicitly mentioned in this work, the proposed method inherently supports parallelization. When multiple latent spaces are involved, the research can be parallelized across multi-core architectures to significantly reduce computational time, instead of using one latent space at a time. Both the latent space construction and counterfactual generation processes could be used in a distributed and parallel execution context. Furthermore, more advanced implementations could leverage multiple machines.

- **Performances**: the previous reports have highlighted that in the Benchmark and KEmoCon dataset, the levels of acceptability and specificity are very good in most of the cases for all the black box models to be explained. The comparison with DiCE has underlined that it is a good trade-off between its random and genetic implementation and plays a role in terms of robustness, highlighting outperformance in Benchmark and KEmoCon datasets.

Limitations

- **Computational Complexity**: although its simplicity in usage and working mechanism, whenever a high number of classes is involved, a higher number of latent spaces should be trained, stored, and subsequently consulted in the counterfactual search for the 1v1 and 1vAll modalities. This amper the execution time as highlighted in the reports. To mitigate this effect, in further work, a parallelized version can be developed as discussed in the previous point. Alternatively, a more lightweight and faster solution is represented by the 1vAll approach without any guarantee in the search of specific counterfactuals.

- **The choice of latent space dimension**: determining the optimal latent space dimension a priori is not feasible without conducting specific trials and evaluating performance on a case-by-case basis. Lower values of the latent space dimension ensure faster, simpler, and more visualizable results, though often at the cost of reduced accuracy. Conversely, higher values do not necessarily lead to improved accuracy, but they do increase complexity, requiring more resources and computation time.

- **Counterfactual not acceptable**: in most of the cases, the number of discardable counterfactuals varies and should be minimized. This limitation is due to the fact that the method works as an approximation explainer; in several cases, there is still a negligible number of counterfactuals to be discarded.

A further interesting work could involve still the usage of the CPILS method but applying it directly to a multi-class latent space, where multiple prediction directions are considered, trying to maintain the same level of interpretability as its standard proposal. This would likely reduce computational time, as the number of operations required would be lower, and it would operate directly on the more complex decision boundary of the original multi-class problem.

A Appendix A: User Manual and Software Implementation

As discussed in the previous sections, the method builds upon the binary CPILS approach. The multi-class CPILS leverages the same module and dependencies as the original version to ensure that its core foundations remain intact. It is structured within a new main class that provides various methods for:

- Initializing with specific configurations.
- Splitting and populating sub-datasets according to configuration.
- Generating new ensembles of latent spaces based on the original black-box structure.
- Producing counterfactuals with specific filtering and aggregation criteria.
- Plotting training losses curves in latent space generation.
- Visualizing the generated ensemble of latent spaces (limited to 2D cases where the latent dimension is 2), selecting specific pairs of classes, or comparing with the original prediction of the multi-class model.
- Visualization of the prediction direction into the latent space among specific classes.
- Saving and reloading externally trained latent samples to avoid redundant computations.
- Running tests on a selected set of test instances.

The next section explains some of these functionalities to guide users in effectively integrating this method into their own code effectively. Finally, key function implementations are presented. The complete code and the full documentation can be found at the following GitHub link [37].

A.1 User Manual

The software is developed using Python language as it is one of the high-level programming languages most employed in the machine learning field. It provides a wide range of libraries to implement complex functionalities in a very easy way. To enhance result visualization and enable interactive exploration, the Jupyter Notebook environment is adopted here to guide users through the main functionalities. The following explanations refer to an ad-hoc pre-configured Jupyter Notebook, **MultiClass_CPILS_Tutorial.ipynb**, located in the same directory as the source file. This tutorial uses the Body Performance dataset as a data source to demonstrate the usage of various functions.

Quick Start

The first thing to do is to import the required library to enable the usage of the class where the multi-class CPILS solution is implemented. Using this command, all the required imports to work with the original CPILS and utilities are included.

```
1 import MultiClass_CPILS
```

Next, load a (multi-class) dataset and split it into train and test sets. Then, select a black box classifier and fit it with the data previously prepared. In this example, an XGB classifier is employed. Remember to copy the original black-box model before fitting it.

```
1  from xgboost import XGBClassifier
2  from copy import deepcopy
3  train_set=pd.read_csv(DS_PATH+"/BodyPerformance/
       bodyPerformance_Train.csv")
4  test_set=pd.read_csv(DS_PATH+"/BodyPerformance/
       bodyPerformance_Test.csv")
5  X_train=train_set.iloc[:,:-1].astype(np.float64)
6  X_test=test_set.iloc[:,:-1].astype(np.float64)
7  y_train=train_set.iloc[:,-1]
8  y_test=test_set.iloc[:,-1]
9
10 bbox_chosen = XGBClassifier(n_estimators=60, reg_lambda=3,
       eval_metric='logloss', random_state=42)
11 # Copy the original model before training it
12 bbox = deepcopy(bbox_chosen)
13 bbox.fit(X_train, y_train)
```

Initialization

Decide the working modality from the available choices: `"1v1"`, `"1vAll"`, `"1vAll_Spec"`, `"Allv1"`, `"All_Specv1"`. Next, create an instance of the class by calling its constructor and passing the previously defined variable as an argument. The first argument is the (multi-class) trained black box, the second and third are the training and test sets, then the copied untrained black box model, next the target class name, default `"class"`, and finally the modality chosen. This class allows for future studies to approximate binary models using a different black-box model, distinct from the multi-class model being explained. For this reason, a deep copy is required before training.

```
1 ens_latent_spaces=MultiClass_CPILS.MultiClass_CP_ILS(bbox,
    train_set,test_set,bbox_chosen,"class",mode="1v1")
```

Here is the output obtained with the previous commands.

```
Target class: class
Class values are: [0 1 2 3]
Numerical indexes: [0, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Categorical indexes: [1, 2]
[[0], [1, 2], [3], [4], [5], [6], [7], [8], [9], [10], [11]]

Modality:  1v1
Preparation of datasets for combination (0, 1)
Preparation of datasets for combination (0, 2)
Preparation of datasets for combination (0, 3)
Preparation of datasets for combination (1, 2)
Preparation of datasets for combination (1, 3)
Preparation of datasets for combination (2, 3)
```

Figure 20: Initialization output for a 1v1 mode.

The method correctly identifies the target class name, determines the number of classes to be recognized, and distinguishes between categorical and numerical features. It then generates all possible combinations of latent spaces to be learned, preparing the dataset by splitting it according to the selected modality.

```
Target class: class
Class values are: [0 1 2 3]
Numerical indexes: [0, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Categorical indexes: [1, 2]
[[0], [1, 2], [3], [4], [5], [6], [7], [8], [9], [10], [11]]

Modality:  1vAll
Preparation of datasets for combination (0, 'All')
Preparation of datasets for combination (1, 'All')
Preparation of datasets for combination (2, 'All')
Preparation of datasets for combination (3, 'All')
```

Figure 21: Another example of initialization output for a 1vAll mode.

Latent Space Generation

To start the generation of latent spaces process, first determine the number of latent dimensions to use and the early stopping criteria to prevent overfitting. Then, call the following function passing these two information as the first two parameters. An additional parameter `"timer"` is present to visualize the number of seconds at the end of the entire training process.

```
1 ens_latent_spaces.generate_ls(latent_dim_k=2,early_stop_param
    =10,timer=True)
```

In this example, the selected latent space dimension is set to 2 and the early stop parameter to 10. An output like this is shown, highlighting the training and test losses and the overall training time elapsed in seconds.

```
Training latent space for combination: (0, 1)
Epoch: 229 Train Loss 0.00255 Test Loss 0.00188 Early Stopping 10
Training latent space for combination: (0, 2)
Epoch: 321 Train Loss 0.00296 Test Loss 0.00157 Early Stopping 10
Training latent space for combination: (0, 3)
Epoch: 106 Train Loss 0.03369 Test Loss 0.00326 Early Stopping 10
Training latent space for combination: (1, 2)
Epoch: 165 Train Loss 0.00240 Test Loss 0.00182 Early Stopping 10
Training latent space for combination: (1, 3)
Epoch: 259 Train Loss 0.00233 Test Loss 0.00102 Early Stopping 10
Training latent space for combination: (2, 3)
Epoch: 850 Train Loss 0.00248 Test Loss 0.00134 Early Stopping 10
Time needed to train all the latent spaces 540.22 sec
```

Figure 22: Another example of initialization output for a 1vAll mode.

To store all the generated latent information in a dictionary and load it later for further experimentation, it is possible to use these two functions to save and load the data from the file system, respectively.

```
1 ens_latent_spaces.load_latent_spaces(LS_PATH+"
    Body_Perf_XGB_gen_lat_spaces1V1.pkl")
2 ens_latent_spaces.save_latent_spaces(LS_PATH+"
    Body_Perf_XGB_gen_lat_spaces1V1.pkl")
```

Plotting Functionalities

To visualize the generated latent spaces and assess their quality, several plotting functions are provided, available exclusively for two-dimensional latent spaces. The following function is designed to plot a specific combination of classes. It accepts three parameters:

- **target** (which can be `classes_MBB` or `classes_BBB`) specifies whether the contribution of the original (multi-class) black box model should be visualized or just the binary classification.
- **class_name** determines if a particular class involved in the combination should be plotted.
- **comb** indicates a specific tuple, allowing the visualization of a particular combination.

Here are reported some examples of usage:

```
ens_latent_spaces.plot_ls_comb(target="classes_MBB")
ens_latent_spaces.plot_ls_comb(target="classes_BBB")
ens_latent_spaces.plot_ls_comb(target="classes_BBB",comb=(0,1))
ens_latent_spaces.plot_ls_comb(target="classes_MBB",class_name
    =3)
```

The target attribute also allows to specify an additional value `"directions"` to visualize the prediction directions within the binary latent space.

```
ens_latent_spaces.plot_ls_comb(target="directions")
```

A more general comparison can be done, considering the contribution of the original multi-class black-box classifier and all the possible class combinations using the following function. It visualizes how the multi-class black-box classifier assigns labels to points within the latent space and how the binarized classifier classifies them for each specific combination.

```
ens_latent_spaces.plot_ls_comb_compare()
```

To visualize the training and testing loss curves, this function can be used.

```
ens_latent_spaces.plot_ls_training_losses()
```

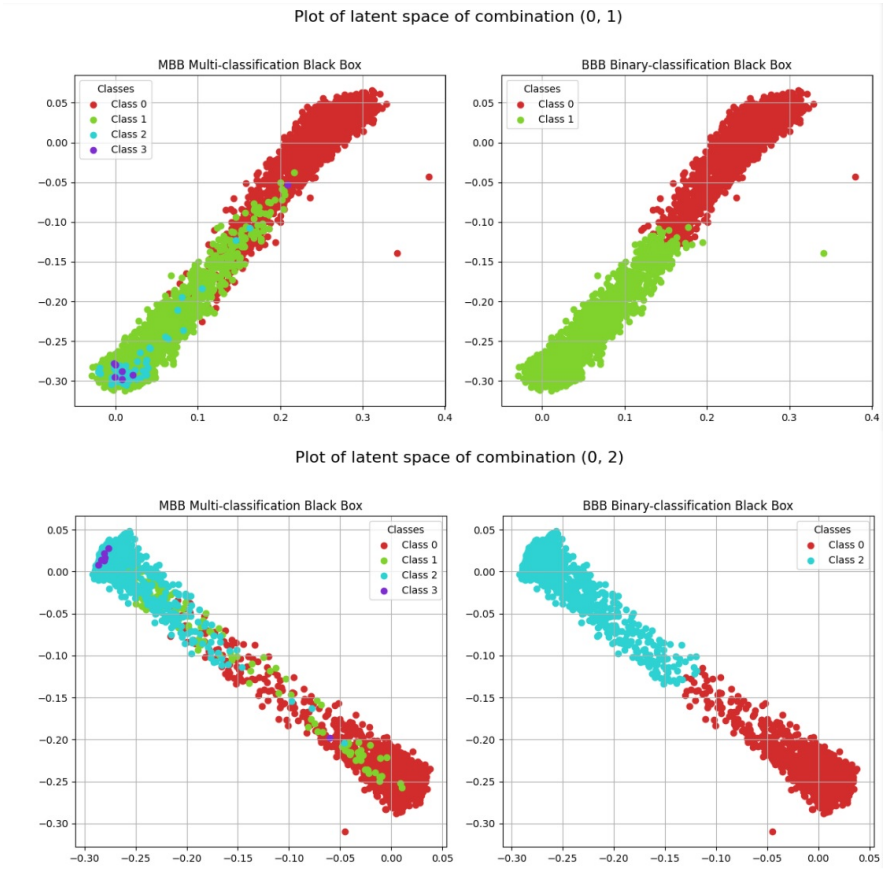

Figure 23: An example of the output generated by a plotting function that visualizes two specific latent spaces comparing how MBB and BBB classify the latent points.

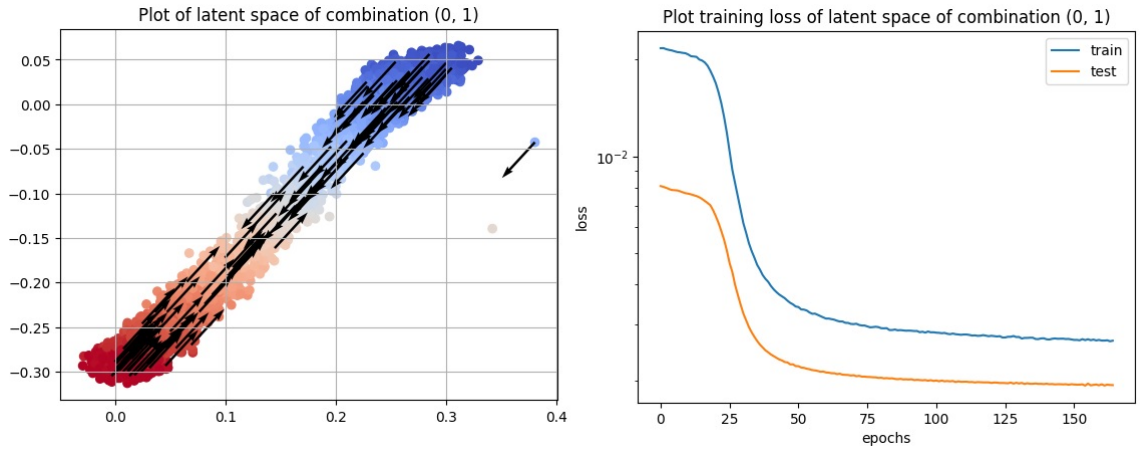

Figure 24: Two examples of plots illustrate the visualization of prediction directions and the loss curve for the specific class combination (0,1).

Counterfactuals Generation

The generation of counterfactuals requires the following parameters:

- **test_instance** : the instance to find the counterfactuals
- **change_f**: the list of features to be actionable, specifying in an array the indexes of the involved columns.
- **max_f**: the maximum number of features to be changed during the research process as explained in (3.1.2).
- **counter_classes**: allows to specify in an array the classes to be searched during the generation phase, obviously, different from the class target predicted from the original MBB. If this parameter is left to -1, all the possible counter classes are taken into account.
- **filtering_mode**: this can be only `"acceptable"` (default) or `"specific"`.

In the following example, the specific function is called keeping the first five features fixed, allowing for the maximum level of changeability among the remaining variables. In this counterfactual search, only latent spaces where the resulting class is either 1 or 3 are considered, specifically searching for specific counterfactuals. The counterfactuals are saved in a dict format, in this case in the ris variable, and are accessed using as key the counterfactual label.

```
1  test_instance=test_set.iloc[index:index+1,:-1]
2  #The set of changable feature
3  change_f=list(range(5,test_instance.shape[1]))
4
5  # set fixed "age, gender_F, gender_M, height_cm, weight_kg" in
       Body Performance
6
7  #The number of features that can be changed in the search
8  max_f=len(change_f)
9
10 ris=ens_latent_spaces.generate_counterfactuals(test_instance,
       change_f,max_f,counter_classes=[1,3],filtering_mode="specific
       ")
```

```
Modality: 1v1
Starting generation of counterfactuals using filtering_mode: specific
'specific' considers only counterfactuals belonging to a specific class, different from the predicted outcome of the Multi

The predicted class from MBB is [2]

Generation of counterfactual class 1
100%|██████████| 1/1 [00:01<00:00,  1.48s/it]
BBB Encoded y_pred: [0 0 0 0 0]
BBB Decoded y_pred: [1 1 1 1 1]
MBB Original y_pred: [1 1 1 0 2]
acceptable counterfactuals: 4
Discarded counterfactuals: 1

Generation of counterfactual class 3
100%|██████████| 1/1 [00:00<00:00,  9.37it/s]
BBB Encoded y_pred: [1 1 1 1 1]
BBB Decoded y_pred: [3 3 3 3 3]
MBB Original y_pred: [2 2 3 3 2]
acceptable counterfactuals: 2
```

Figure 25: An example of the output obtained executing the previous script.

The class recognized by the MBB classifier is 2. The counterfactuals are generated correctly for classes 1 and 3, as previously indicated. Selecting the SPECIFIC filtering mode, in the first latent space, 5 counterfactuals are generated, however, only 4 can be considered acceptable because the last one has the label equal to 2, that is the same label as the starting class. For this reason, the method discards it, but before returning the set, it also discards the 4th counterfactual because it has a label (0) differently from the SPECIFIC class combination for which the search was conducted (1,2). In the second latent space, three of the five counterfactuals have a label class equal to 2, so they are discarded. Only two counterfactuals with class 3 are acceptable (and specific).

	age	gender_F	gender_M	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm
1	54.0	1.0	0.0	160.3	52.7	27.8	85.0	159.0	30.0	13.4	19.0	151.0

```
ris[1]["counterfactuals"]
```

	age	gender_F	gender_M	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm
0	54.0	1.0	0.0	160.3	52.7	27.8	85.0	159.0	0.0	213.0	19.0	151.0
1	54.0	1.0	0.0	160.3	52.7	27.8	118.0	159.0	30.0	213.0	19.0	151.0
2	54.0	1.0	0.0	160.3	52.7	27.8	85.0	43.9	30.0	213.0	19.0	151.0

Figure 26: On the top, the test instance used to search its counterfactuals; on the bottom, the three counterfacutals of class 1. The first five features remain unchanged, ensuring actionability, while the others are modified as necessary to alter the original class label.

Ranking Counterfactuals

This functionality has been added as post-processing elaboration on the final set of acceptable counterfactuals. It has been intentionally developed separately from the counterfactual generation methods to provide users with the flexibility to manage the final set as they see fit. The function designed for this task requires the following parameters:

- **test_instance**: the test instance used to generate counterfactuals, it is used here to compute proximity distance, if that metric is selected.
- **counterfactuals**: the list of counterfactuals generated by the previous function and to be post-processed.
- **ranking** : only value allowed are `"proximity"` (default) or `"accuracy"`. The first ranks the counterfactuals per proximity distance (12), while the latter sorts them in descending order according to the MBB prediction probability.
- **group_by_class** : a boolean indicating whether the ranking should be performed separately for each counterfactual class or combined across all classes, regardless of their labels.
- **top_k**: a number that specifies the maximum number of counterfactuals returned, default value is -1 meaning no restrictions.

Here is reported an example where the specific ranking function is used. The previously generated counterfactuals of class 1 and 3 are joined together using a ranking based on proximity, without any grouping criteria, limiting the amount of counterfactuals to 4.

```
1 #merge the 2 set of counterfactuals in the ris variable
2 counterfactuals=pd.concat([ris[1]["counterfactuals"],ris[3 ["
    counterfactuals"]]).reset_index(drop=True)
3
4 results=ens_latent_spaces.execute_ranking_of_counterfactuals(
    test_instance,counterfactuals, ranking="proximity",
    group_by_class=False, top_k=4)
```

	age	gender_F	gender_M	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm
1	54.0	1.0	0.0	160.3	52.7	27.8	85.0	159.0	30.0	13.4	19.0	151.0

	age	gender_F	gender_M	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm	class	dist	prob
0	54.0	1.0	0.0	160.3	52.7	27.8	85.0	159.0	30.0	13.4	0.0	151.0	3	15.833333	0.955909
1	54.0	1.0	0.0	160.3	52.7	27.8	85.0	159.0	30.0	-25.0	19.0	151.0	3	32.000000	0.998215
2	54.0	1.0	0.0	160.3	52.7	27.8	85.0	159.0	0.0	213.0	19.0	151.0	1	168.201599	0.706606
3	54.0	1.0	0.0	160.3	52.7	27.8	118.0	159.0	30.0	213.0	19.0	151.0	1	168.591304	0.407182

Figure 27: On the top, the test instance used to search its counterfactuals; on the bottom, the four counterfacutals of class 1 and 3 sorted by proximity distance. The last three columns are manually added to verify the correct sorting and which counterfactual classes are involved.

Here is reported another example of usage for a different test sample where the instance is labelled form MBB with class 0. In this case, the ranking mode is set to `"accuracy"`, and no limitation on the maximum number of counterfactuals is set.

```
1 index=3050
2 test_instance=test_set.iloc[index:index+1,:-1]
3 ris=ens_latent_spaces.generate_counterfactuals(test_instance,
    change_f,max_f,counter_classes=[1,2,3],filtering_mode="
    specific")
4 #merge the 3 set of counterfactuals in the ris variable
5 counterfactuals=pd.concat([ris[1]["counterfactuals"],ris[2]["
    counterfactuals"],ris[3]["counterfactuals"]]).reset_index(
    drop=True)
6 results=ens_latent_spaces.execute_ranking_of_counterfactuals(
    test_instance,counterfactuals, ranking="accuracy",
    group_by_class=False, top_k=-1)
```

	age	gender_F	gender_M	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm
3050	61.0	0.0	1.0	172.6	73.0	20.4	72.0	114.0	40.9	15.0	37.0	188.0

	age	gender_F	gender_M	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm	class	dist	prob
0	61.0	0.0	1.0	172.6	73.0	20.4	72.0	114.0	40.9	-25.0	37.0	188.0	3	33.333333	0.934322
1	61.0	0.0	1.0	172.6	73.0	20.4	72.0	114.0	40.9	15.0	0.0	188.0	3	30.833333	0.915930
2	61.0	0.0	1.0	172.6	73.0	54.9	72.0	114.0	0.0	15.0	37.0	188.0	3	44.589641	0.631982
3	61.0	0.0	1.0	172.6	73.0	20.4	72.0	114.0	40.9	15.0	7.0	303.0	2	99.040536	0.598788
4	61.0	0.0	1.0	172.6	73.0	20.4	72.0	114.0	40.9	213.0	37.0	188.0	1	165.000000	0.436933

Figure 28: On the top, the test instance used to search its counterfactuals; on the bottom, the counterfacutals of class 1,2 and 3 sorted by prediction probability. The last three columns are manually added to verify the correct sorting and which counterfactual classes are involved.

A.2 Source Code

This section highlights the most important and relevant parts of the source code. For better readability, some commands are split across multiple lines. Full documentation has been generated using Sphinx, allowing users to easily understand the purpose of each function.

```
class MultiClass_CP_ILS():

    """
    A multi-class implementation of the CP_ILS method designed for binary classification.

    This class handles multi-class classification scenarios by
    transforming them into binary classification problems in different modes.
    The supported modes include 1v1, 1vAll, 1vAll_Spec, Allv1, and All_Specv1,
    which specifies how the multi-class problem is decomposed into binary ones.


    Constants:
    ----------

    MODE_1V1 : str
        Constant representing the 1-vs-1 mode.
    MODE_1VALL : str
        Constant representing the 1-vs-All mode.
    MODE_1VALL_SPEC : str
        Constant representing the 1-vs-All (Specific) mode.
    MODE_ALLV1 : str
        Constant representing the All-vs-1 mode.
    MODE_ALL_SPECV1 : str
        Constant representing the All-vs-1 (Specific) mode.

    Attributes:
    ----------

    flatten : function
        A lambda function to flatten multi-dimensional arrays.
    __mbbox_model : model
        The multi-class black-box classifier model used for predictions, also referred as MBB
    __train_set : pd.DataFrame
        The training dataset used to generate latent spaces.
    __test_set : pd.DataFrame
        The test dataset used for validation and counterfactual generation.
    __mode : str
        The mode of classification, which defines how the multi-class problem is treated.
    __class_combinations : list
        A list of class combinations generated according to the mode.

    __dict_train_datasets : dict
        A dictionary storing the training datasets for each class combination.
    __dict_test_datasets : dict
        A dictionary storing the test datasets for each class combination.
    __dict_latent_spaces : dict
        A dictionary storing the latent spaces for each class combination.
    __hex_colors_for_classes : dict
        A dictionary storing the hexadecimal colors for each class, used for
        plotting points with same class color in all exp.
```

```
Public Methods:
--------

__init__(self, bbox_model, train_set, test_set, target_class="class", mode=MODE_1V1):
    Initializes the MultiClass_CP_ILS object with the given dataset,
    model, and classification mode.

generate_ls(self, latent_dim_k=2, early_stop_param=5, timer=False):
    Generates latent spaces for each class combination using CP-ILS method,
    leveraging a black-box model.

generate_counterfactuals(self, test_instance, change_f, max_f, counter_classes=-1,
                         filtering_mode="", debug=True):
    Generates counterfactual explanations for a given test instance by
    modifying the input features for a specific mode

execute_ranking_of_counterfactuals(self, test_instance, df_counterfactuals,
                                   ranking="proximity", group_by_class=True, top_k=-1):
    Function to rank with a certain ranking criterion a list of
    counterfactuals, obtained respect to a test_instance

plot_ls_comb(self,target,comb=None,class_name=None):
    Function to plot a specific combination
    of latent space callable from external.

plot_ls_comb_compare(self,comb=None,class_name=None):
   Function to plot and compare specific combination
   of latent space callable from external.

plot_ls_training_losses(self,comb=None,class_name=None):
    Function to plot specific combination
    of train_test losses callable from external.

execute_testing_on_data(self,test_name,test_instances,change_f, max_f,
    filtering_mode="acceptable",debug=False):
    Automatic procedure to print on external files
    the counterfactuals main information and treat them separately

save_latent_spaces(self, filename):
    Saves the latent spaces to a file using pickle.

load_latent_spaces(self, filename):
    Loads latent spaces from a file using pickle.

get_ls_of_class(self, class_name):
    Returns the latent spaces specific to a given class.

get_ls_of_not_class(self, class_name):
    Returns the latent spaces for all classes except the specified one.
```

```
    Private Methods:
    --------

    __adjust_target_class(self, target_name, train_set,test_set):
        Adjusts the target class column to be the last column in the dataset and
        renames it to 'class'.

    __build_dataset_idx_num_cat(self):
        Builds index lists for numerical and categorical attributes in the dataset.

    __assign_color_to_classes(self):
        Assigns a unique color to each class using HLS color space.

    __prepare_datasets(self):
        Prepares datasets for the specified classification mode by splitting them
        into appropriate combinations of classes.

    __generate_counterfactuals_1v1(self,test_instance,change_f,max_f,y_pred,counter_classes=-1,
                                   filtering_mode="",debug=True):
        Generates counterfactual explanations for a given test instance using the modality 1v1

    __generate_counterfactuals_1vAll_AllSpec(self,test_instance,change_f,max_f,y_pred,
                        counter_classes=-1,filtering_mode="",debug=True):
        Generates counterfactual explanations for a given test instance using
        the modality 1vALL or 1vALLSpec

    __generate_counterfactuals_Allv1_Allv1Spec(self,test_instance,change_f,max_f,y_pred,
                                       counter_classes=-1,filtering_mode="",debug=True):
        Generates counterfactual explanations for a given test instance using
        the modality Allv1 or AllSpecV1

    __plot_ls_directions(self,latent,X_instances,title=""):
        Function to plot the binary classifier counterfactuals direction in a latent space

    __plot_ls_class_scatter_plot(self,Z_train,Y_labels,title="",target="classes_MBB",
                                 comb=("None","All")):
        Function to plot the 2D scatter plot of a latent space, optionally
        targeting a specific classes combination

    __prepare_ls_class_scatter_plot(self,ax,Z_train,Y_labels,title="",comb=("None","All")):
        Function to plot the 2D scatter plot of a latent space comparing the MBB with the BBB

    __plot_ls_losses(self,losses,title=""):
        Function to plot the losses of training and testing during the latent spaces generations

    """
```

```
    #Constants
    MODE_1V1="1v1"
    MODE_1VALL="1vAll"
    MODE_1VALL_SPEC="1vAll_Spec"
    MODE_ALLV1="Allv1"
    MODE_ALL_SPECV1="All_Specv1"

    def __init__(self,bbox_model: BaseEstimator,train_set: pd.DataFrame,test_set: pd.DataFrame,
                 bbox_structure:BaseEstimator,target_class:str="class",mode:str=MODE_1V1):
        """
            Constructor method for initialization of attributes, preparing datasets in
            proper format and for class combinations.
            Setting the target class with name class if different, locate what indexes
            are categorical and numerical, build the needed combination used for generation
            of latent spaces and further to generate counterfactuals depending on the mode
            passed.

            Args:
                bbox_model (BaseEstimator): A scikit-learn trained model used for
                                            multi-class classification (black box model).
                train_set (pd.DataFrame): DataFrame containing the training dataset.
                test_set (pd.DataFrame): DataFrame containing the test dataset.
                target_class (str): The target class to predict. Default is "class".
                bbox_structure (BaseEstimator): The black box model to train inside the
                                                specific combination
                mode (str): The mode for generating counterfactuals. Default is MODE_1V1.
        """

        MODE_ALLOWED=[self.MODE_1V1,self.MODE_1VALL,self.MODE_1VALL_SPEC,
                      self.MODE_ALLV1,self.MODE_ALL_SPECV1]

        #Global utility function for flatteing, used also in original CP_ILS
        self.flatten = lambda m: [item for row in m for item in row]

        #Preliminary: check if target and mode passed are correct
        if (target_class not in train_set.columns.values):
            print(f"{target_class} is not a valid target class. \n"
                  f"Please insert one of {train_set.columns.values}")
            return

        if (mode not in MODE_ALLOWED):
            print(f" {mode} is not one of the allowed modality.\n"
                  f"Please insert one of  {MODE_ALLOWED}")
            return

        self.__adjust_target_class(target_class,train_set,test_set)

        #Group of dataset per combination of classes
        self.__class_combinations=[]
        self.__dict_train_datasets={}
        self.__dict_test_datasets={}
        self.__dict_latent_spaces={}

        self.__prepare_datasets()
```

```
    def load_latent_spaces(self,filename:str):
        """
            Load the generated latent spaces from a file.

            Args:
                filename (str): The string containing the path to load the file
        """
        with open(filename, "rb") as file:
            self.__dict_latent_spaces=pickle.load(file)
            print(f"Latent space loaded from {filename}")


    def save_latent_spaces(self, filename: str):
        """
            Save the generated latent spaces into a file.

            Args:
                filename (str): The path where the latent spaces will be saved.
        """
        with open(filename, "wb") as file:
            pickle.dump(self.__dict_latent_spaces, file)
        print(f"Latent space saved in {filename}")
    def get_ls_of_class(self, class_name:str):
        """
            Method to return the latent space where the class with class_name is involved.

            Args:
                class_name (str): The name of the class to be reserched in
                                  latents spaces combinations
            Returns:
                    list: A list of tuples, where each tuple contains a latent space
                          key and its corresponding value, filtered to include only
                          those where class_name is present in the key.
        """
        return [(key,value) for key,value in zip(self.__dict_latent_spaces.keys(),
                                                 self.__dict_latent_spaces.values())
                if class_name in key]

    def get_ls_of_not_class(self, class_name: str):
        """
        Method to return the latent space where the class with class_name is NOT involved.

        Args:
            class_name (str): The name of the class to be researched in
                              latent space combinations.

        Returns:
            list: A list of tuples, where each tuple contains a latent
                  space key and its corresponding value,
                filtered to include only those where
                class_name is NOT present in the key.
        """
        return [(key,value) for key,value in zip(self.__dict_latent_spaces.keys(),
                                                 self.__dict_latent_spaces.values())
                if class_name not in key]
```

```
def __prepare_datasets(self):
    """
        This function aims to preparing the training and testing dataset for each possible
        class combinations depending on the mode selected at the begin.
        In 1v1 mode: for each possible pair of classes "combinations" filter the samples
        that are involved for that binary classification
        In 1vAll or Allv1 mode: for each possible class select its samples entierly
        and then selecting randomly the remaining (n_classes-1) in equal proportion
        to obtain the same number of samples of the target class.
        In 1vAllSpec or Allv1Spec: the procedure is the same of 1vAll-Allv1 but not chosen
        the sample randomly but the ones most near to the target class.

    """

    print(f"\nModality:  {self.__mode}")

    if (self.__mode==self.MODE_1V1):

        for comb in combinations(self.__all_classes,2):
            self.__class_combinations.append(comb)

            class_i=comb[0] #e.g 1
            class_j=comb[1] #e.g 3

            print(f"Preparation of datasets for combination {comb}")
            self.__dict_train_datasets[comb]=pd.concat([self.__train_set[
                self.__train_set["class"]==class_i],
                self.__train_set[self.__train_set["class"]==class_j]]).astype(np.float64)

            self.__dict_test_datasets[comb]=pd.concat([self.__test_set[
                self.__test_set["class"]==class_i],
                self.__test_set[self.__test_set["class"]==class_j]]).astype(np.float64)

    if (self.__mode==self.MODE_1VALL or self.__mode==self.MODE_ALLV1):

        for class_value in self.__all_classes:
            comb=(class_value,"All")
            self.__class_combinations.append(comb)

            class_i=comb[0] #e.g 1
            class_j=comb[1] #e.g ALL

            print(f"Preparation of datasets for combination {comb}")
            self.__dict_train_datasets[comb]=pd.concat([self.__train_set[
                self.__train_set["class"]==class_i],
                self.__train_set[self.__train_set["class"]==class_j]]).astype(np.float64)

            self.__dict_test_datasets[comb]=pd.concat([self.__test_set[
                self.__test_set["class"]==class_i],
                self.__test_set[self.__test_set["class"]==class_j]]).astype(np.float64)

    if (self.__mode==self.MODE_1VALL or self.__mode==self.MODE_ALLV1):

        for class_value in self.__all_classes:
            comb=(class_value,"All")
            self.__class_combinations.append(comb)

            class_i=comb[0] #e.g 1
            class_j=comb[1] #e.g ALL
```

```
            print(f"Preparation of datasets for combination {comb}")

            #Get the number of instances of the target class and obtain the
            # number of sample to use for the other classes
            train_class_i=self.__train_set[self.__train_set["class"]==class_i]
            no_train_samples_per_class=int(train_class_i.shape[0]/(len(self.__all_classes)-1))

            test_class_i=self.__test_set[self.__test_set["class"]==class_i]
            no_test_samples_per_class=int(test_class_i.shape[0]/(len(self.__all_classes)-1))

            train_set_copy=self.__train_set.copy()
            test_set_copy=self.__test_set.copy()

            #Create new dataframes sampling accordingly to the number obtained,
            # in order to have a balanced datasets.
            final_train_set=pd.DataFrame()
            final_test_set=pd.DataFrame()

            for cls_value in self.__all_classes:
                if (cls_value!=class_i):
                    no_eff_min_train_samples=min(
                        train_set_copy[train_set_copy["class"]==cls_value].shape[0],
                                                    no_train_samples_per_class)

                    no_eff_min_test_samples=min(
                        test_set_copy[test_set_copy["class"]==cls_value].shape[0],
                        no_test_samples_per_class)

                    final_train_set=pd.concat([final_train_set,
                        train_set_copy[
                            train_set_copy["class"]==cls_value].sample(n=no_eff_min_train_samples,
                                                        random_state=42)]).reset_index(drop=True)

                    final_test_set=pd.concat([final_test_set,
                            test_set_copy[
                                test_set_copy["class"]==cls_value].sample(n=no_eff_min_test_samples,
                                    random_state=42)]).reset_index(drop=True)
                else:
                    final_train_set=pd.concat([final_train_set,
                        train_set_copy[
                            train_set_copy["class"]==cls_value].sample(
                                n=no_train_samples_per_class*(len(self.__all_classes)-1),
                                random_state=42)]).reset_index(drop=True)

                    final_test_set=pd.concat([final_test_set,
                        test_set_copy[
                            test_set_copy["class"]==cls_value].sample(
                                n=no_test_samples_per_class*(len(self.__all_classes)-1),
                                random_state=42)]).reset_index(drop=True)

            #**CONVENTION**:
            #Lonley class "1" is encoded with class 0
            #Grouped class "All" is encoded with class 1
            #This is useful for the testing part in the generation of counterfactual to determine
            # if they are acceptable or not
```

```
            #Binarization of the dataset impose as class=0 the class_i, the first in
            # the combination and 1 the others
            final_train_set.loc[final_train_set["class"]==class_i,"class"]=-1
            final_train_set.loc[(final_train_set["class"]!=class_i) &
                                (final_train_set["class"]!=-1),"class"]=1
            final_train_set.loc[(final_train_set["class"]==-1),"class"]=0

            #Binarization of the dataset impose as class=0 the class_i, the first
            # in the combination and 1 the others
            final_test_set.loc[final_test_set["class"]==class_i,"class"]=-1
            final_test_set.loc[(final_test_set["class"]!=class_i) &
                               (final_test_set["class"]!=-1),"class"]=1
            final_test_set.loc[(final_test_set["class"]==-1),"class"]=0

            self.__dict_train_datasets[comb]=final_train_set
            self.__dict_test_datasets[comb]=final_test_set

        if (self.__mode==self.MODE_1VALL_SPEC or self.__mode==self.MODE_ALL_SPECV1):

            for class_value in self.__all_classes:
                comb=(class_value,"All_Spec")
                self.__class_combinations.append(comb)

                class_i=comb[0] #e.g 1
                class_j=comb[1] #e.g ALL

                print(f"Preparation of datasets for combination {comb}")

                #Get the number of instances of the target class and obtain the number
                # of sample to use for the other classes
                train_class_i=self.__train_set[self.__train_set["class"]==class_i]
                no_train_samples_per_class=int(
                    train_class_i.shape[0]/(len(self.__all_classes)-1))

                test_class_i=self.__test_set[self.__test_set["class"]==class_i]
                no_test_samples_per_class=int(
                    test_class_i.shape[0]/(len(self.__all_classes)-1))

                final_train_set=pd.DataFrame()
                final_train_set=pd.concat([final_train_set,
                    train_class_i.sample(n=no_train_samples_per_class*(len(self.__all_classes)-1),
                            random_state=42)],axis=0).reset_index(drop=True)

                #For each remaining dataset, the points are chosen
                # using the shortest proximity distance to the class_i cluster in even proportion
                for class_value_rim in [x for x in self.__all_classes if x != class_value]:

                    #Compute a matrix of distances among multidimentional
                    # points and selecting the lowest values
                    train_class_j=self.__train_set[self.__train_set["class"]==class_value_rim]
                    dists=self.cdist_sample_distance(train_class_i,train_class_j)
                    flat_indices = np.argsort(dists, axis=None)
                    row_indices, col_indices = np.unravel_index(flat_indices, dists.shape)

                    unique_col_indices = []
                    for c in col_indices:
                        if c not in unique_col_indices:
                            unique_col_indices.append(c)
                        if len(unique_col_indices) == no_train_samples_per_class:
                            break
                    final_train_set=pd.concat([
                        final_train_set,
                        train_class_j.iloc[unique_col_indices,:]],axis=0).reset_index(drop=True)
```

```
            #The test dataset instead is chosen randomly because it is not possibile
            # to know in advance what data could be presented
            test_set_copy=self.__test_set.copy()
            final_test_set=pd.DataFrame()

            for cls_value in self.__all_classes:
                if (cls_value!=class_i):
                    final_test_set=pd.concat([final_test_set,
                        test_set_copy[
                            test_set_copy["class"]==cls_value].sample(
                                n=no_test_samples_per_class,random_state=42)]).reset_index(drop=True)
                else:
                    final_test_set=pd.concat([final_test_set,
                        test_set_copy[
                            test_set_copy["class"]==cls_value].sample(
                                n=no_test_samples_per_class*(len(self.__all_classes)-1),
                                    random_state=42)]).reset_index(drop=True)


            #**CONVENTION**:
            #Lonley class "1" is encoded with class 0
            #Grouped class "All" is encoded with class 1
            #This is useful for the testing part in the generation of counterfactual to determine
            # if they are acceptable or not

            #Binarization of the dataset impose as class=0 the class_i,
            # the first in the combination and 1 the others
            final_train_set.loc[final_train_set["class"]==class_i,"class"]=-1
            final_train_set.loc[(final_train_set["class"]!=class_i) &
                                (final_train_set["class"]!=-1),"class"]=1
            final_train_set.loc[(final_train_set["class"]==-1),"class"]=0

            #Binarization of the dataset impose as class=0 the class_i,
            # the first in the combination and 1 the others
            final_test_set.loc[final_test_set["class"]==class_i,"class"]=-1
            final_test_set.loc[(final_test_set["class"]!=class_i) &
                               (final_test_set["class"]!=-1),"class"]=1
            final_test_set.loc[(final_test_set["class"]==-1),"class"]=0

            self.__dict_train_datasets[comb]=final_train_set
            self.__dict_test_datasets[comb]=final_test_set
```

```python
def generate_ls(self,latent_dim_k:int=2,early_stop_param:int=5,timer:bool=False):
    """
        This method compute the latent spaces for all the combinations of classes according
        to the initial selected modality and using as training and testing dataset the
        specific prepared for each combination.
        For each combination, a new black box classifier is trained with the sub-selected
        training samples, then the CP_ILS classical method is called in order to build this
        specific binary latent space.

        The results of binary black box, latent space and training values are
        saved into self.__dict_latent_spaces dict variable.

        Args:
            latent_dim_k (int): the dimension K of each generated latent space
            by default it is equal to 2

            early_stop_param (int): the condition to avoid
            to generate too many iterations in case of no improvments, default 5 epochs.

            timer (bool): boolean variable to take note of the time elapsed for training
    """

    if timer:
        start_time = time.time()

    for comb in self.__class_combinations:
        print(f"Training latent space for combination: {comb}")

        class_i=comb[0]
        class_j=comb[1]

        label_encoder = LabelEncoder()
        sub_ij_training_dataset=self.__dict_train_datasets[comb].copy()
        sub_ij_test_dataset=self.__dict_test_datasets[comb].copy()

        sub_ij_training_dataset["class"]=label_encoder.fit_transform(
            sub_ij_training_dataset["class"].astype(np.int64))

        sub_ij_test_dataset["class"]=label_encoder.transform(
            sub_ij_test_dataset["class"].astype(np.int64))

        binary_bb = deepcopy(self.__bbox_structure)
        binary_bb.fit(sub_ij_training_dataset.iloc[:,:-1],
                      sub_ij_training_dataset.iloc[:,-1])

        latent = cpils.CP_ILS(binary_bb.predict, binary_bb.predict_proba,
                              latent_dim=latent_dim_k, early_stopping=early_stop_param)
        losses = latent.fit((sub_ij_training_dataset.iloc[:,:-1].values,
                             sub_ij_test_dataset.iloc[:,:-1].values),
                            self.__idx_num_cat, seed=42)

        self.__dict_latent_spaces[comb]={"binary_bbox":binary_bb,
                                         "label_encoder": label_encoder,
                                         "latent":latent, "losses": losses}
    if timer:
        end_time = time.time()
        print(f"Time needed to train all the latent spaces {(end_time-start_time):.2f} sec")
```

```
def generate_counterfactuals(self,test_instance:pd.DataFrame,change_f:List[int],max_f:int,
                             counter_classes:Any=-1,filtering_mode:str="",debug:bool=True):
    """
        Main function to generate counterfactuals for a specific instance. This function
        should be called after having trained all the latent spaces.
        Depending on the modality initially chosen, it applies the default generate
        counterfacutal method on each combination.

        Args:
            test_instance (pd.DataFrame): the instance interested in counterfactuals
                                          explanations
            change_f (List[int]): the list of feature that can be modifiable during
                                  the procedure of generation
            max_f (int): it is the maximum number of attribute that can be changed
                         togheter from the change_f list
            counter_classes (Any): it specify a list of possible classes values interested
                                   in generation of counterfactuals or -1 if all available
            filtering_mode (str): (optionally) it indicate the modality of possible
                                  filtering of counterfactuals 'acceptable','specific'
            debug (bool): boolean value to print on console debug informations

        Returns:
            list: a list of counterfactuals grouped by class, cotaining information about
                  all the generated, the predictions of the multi-class and binary
                  classifiers and the filtered ones.

    """
    if (debug): print(f"Modality: {self.__mode}")

    if (self.__mode==self.MODE_1V1 or self.__mode==self.MODE_ALLV1 or
        self.__mode==self.MODE_ALL_SPECV1):

        filtering_acceptable = ''''acceptable' considers any counterfactuals
        belonging to classes different from the predicted outcome of the
        Multi-classification black-box model'''

        filtering_specific = ''''specific' considers only counterfactuals
        belonging to a specific class, different from the predicted
        outcome of the Multi-classification black-box model'''

        if filtering_mode!="acceptable" and filtering_mode!="specific":
            print(f"filtering_mode can be only 'acceptable' or 'specific'.\n"
                  f"{filtering_acceptable}\n{filtering_specific}")
            return
    else:

        if filtering_mode!="":
            print(f"In this mode {self.__mode} you cannot specify any filtering method."
                  f"All the counterfactuals are acceptable.")
            return

        if (debug): print(f"Starting generation of counterfactuals ")


    statistics={}
    y_pred=self.__mbbox_model.predict(test_instance)
    if (debug): print(f"\nThe predicted class from MBB is {y_pred}\n")

    if (counter_classes!=-1 and y_pred in counter_classes):
        print("The counter class you want is the same as the predicted class")
        return


    if (self.__mode==self.MODE_1V1):
        return self.__generate_counterfactuals_1v1(test_instance,change_f,max_f,
                                                   y_pred,counter_classes,
                                                   filtering_mode,debug)
```

```
    if (self.__mode==self.MODE_1VALL or self.__mode==self.MODE_1VALL_SPEC):
        return self.__generate_counterfactuals_1vAll_AllSpec(test_instance,change_f,max_f,
                                                              y_pred,counter_classes,
                                                              filtering_mode,debug)

    if (self.__mode==self.MODE_ALLV1 or self.__mode==self.MODE_ALL_SPECV1):
        return self.__generate_counterfactuals_Allv1_Allv1Spec(test_instance,
                                                               change_f,max_f,y_pred,
                                                               counter_classes,
                                                               filtering_mode,debug)

    return None
```

```
    def __generate_counterfactuals_1v1(self,test_instance:pd.DataFrame,change_f:List[int],
                                       max_f:int,y_pred:int,counter_classes:Any=-1,
                                       filtering_mode:str="",debug:bool=True):

        """
            This function should be called after having trained all the latent spaces.
            It applies the default generate counterfacutal method on each combination
            and take the counterfacutals that are acceptable and optionally filtering
            the specifics.

            Args:
                test_instance (pd.DataFrame): the instance interested in
                                              counterfactuals explanations
                change_f (List[int]): the list of feature that can be modifiable during
                                      the procedure of generation
                max_f (int): it is the maximum number of attribute that can be changed
                             togheter from the change_f list
                y_pred (int): the predicted class of the multi-class classifier
                counter_classes (Any): it specify a list of possible classes values interested
                                       in generation of counterfactuals or -1 if all available
                filtering_mode (str): (optionally) it indicate the modality of possible filtering
                                      of counterfactuals 'acceptable','specific', default 'acceptable'
                debug (bool): boolean value to print on console debug informations

            Returns:
                list: a list of counterfactuals grouped by class, cotaining information about
                    all the generated, the predictions of the multi-class
                    and binary classifiers and the filtered ones.
        """

        #Looking in all latent spaces where the class predicted is involved in
        # order to follow the direct direction,
        # from 1 the specific class to (1, an other specific class).

        list_of_latent_spaces_to_be_considered=self.get_ls_of_class(y_pred)

        statistics={}
        for elem in list_of_latent_spaces_to_be_considered:
            comb=elem[0] # here is expected to find a tuple like (1,3)
            latent=elem[1]["latent"]

            class_of_counterfactual= comb[1] if comb[0]==y_pred else comb[0]

            if (counter_classes==-1 or class_of_counterfactual in counter_classes):

                statistics[class_of_counterfactual]={}
                if (debug): print(f"Generation of counterfactual "
                                  f"class {class_of_counterfactual}")

                counterfactuals=latent.get_counterfactuals(test_instance.astype(np.float64),
                                           change_f, max_f, max_steps=50,
                                           n_cfs=-1, n_feats_sampled=10,
                                           topn_to_check=10, seed=42).reset_index(drop=True)

                if (not counterfactuals.empty):

                    bbox=elem[1]["binary_bbox"]

                    y_counter_prediction=bbox.predict(counterfactuals)
                    label_encoder=elem[1]["label_encoder"]
                    if (debug): print(f"BBB Encoded y_pred: {y_counter_prediction}")

                    y_counter_prediction_decoded=label_encoder.inverse_transform(
                        y_counter_prediction)
                    if (debug): print(f"BBB Decoded y_pred: {y_counter_prediction_decoded}")

                    y_counter_prediction_mbbox=self.__mbbox_model.predict(counterfactuals).copy()
```

```
                #Save all the generated counterfactuals and their
                # predictions indipendently they are correct or not
                statistics[class_of_counterfactual]
                ["all_counterfactuals"]=counterfactuals.copy()

                statistics[class_of_counterfactual]
                ["bbbox_predictions"]=y_counter_prediction_decoded.copy()

                statistics[class_of_counterfactual]
                ["mbbox_predictions"]=y_counter_prediction_mbbox.copy()
                #Discard the countefactuals that fails under binary-classifier:
                #this means the methods it self has failed in discovering
                # an acceptable counterfactual
                indexes_with_diff_label=np.where(
                    y_counter_prediction_decoded!=class_of_counterfactual)
                counterfactuals.drop(counterfactuals.index[indexes_with_diff_label[0]],
                                     inplace=True)

                tot_counter_produced_binary=len(counterfactuals)

                y_counter_prediction_mbbox=np.delete(y_counter_prediction_mbbox,
                                                     indexes_with_diff_label)
                if (debug): print(f"MBB Original y_pred: {y_counter_prediction_mbbox}")

                #Discard the counterfactuals that fails under multi-class classifier:
                # the class is the same as starting class
                indexes_with_same_starting_label=np.where(y_counter_prediction_mbbox==y_pred)

                counterfactuals.drop(counterfactuals.index[indexes_with_same_starting_label[0]],
                                     inplace=True)

                y_counter_prediction_mbbox=np.delete(y_counter_prediction_mbbox,
                                                     indexes_with_same_starting_label)

                tot_counter_acceptable=len(counterfactuals)
                if (debug): print(f"acceptable counterfactuals: {tot_counter_acceptable}")

                #Discard all the counterfactuals different from the ORIGINAL
                # class_of_counterfactual designed
                if (filtering_mode=="specific"):
                    indexes_with_diff_label=np.where(
                        y_counter_prediction_mbbox!=class_of_counterfactual)

                    counterfactuals.drop(counterfactuals.index[indexes_with_diff_label[0]],
                                         inplace=True)
                    tot_counter_correct=len(counterfactuals)
                    if (tot_counter_correct!=tot_counter_acceptable):
                        if (debug): print(f"Discarded counterfactuals: "
                                          f"{tot_counter_acceptable-tot_counter_correct}\n")

                statistics[class_of_counterfactual]["counterfactuals"]=counterfactuals
            else:
                statistics[class_of_counterfactual]["all_counterfactuals"]=None
                statistics[class_of_counterfactual]["counterfactuals"]=None

    return statistics
```

B Appendix B: Tables and Additional Details

B.1 Hardware Architecture

The experimental results are obtained using the following hardware devices. The GPU is primarily utilized for training latent spaces, while the CPU is mainly responsible for running various experiments, with counterfactual generation as the main actor.

Device	CPU
Model	Intel® Core i7-8700 Processor
Cores	6
Threads	12
Clock	Up to 3.20 GHz
Max Turbo Frequency	4.60 GHz
Cache	Level 1: 384 KB Level 2: 1.5 MB Level 3: 12 MB
RAM	32 GB DDR4 2133 MHz

Table 36: CPU Specifications

Device	GPU
Model	Nvidia GeForce GTX 1060
CUDA Driver Version	12.5
Runtime Version	12.4
Total Global Memory	6144 MB
Multiprocessors	10
Total CUDA Cores	1280
Max Clock Rate	1759 MHz (1.76 GHz)
L2 Cache Size	1572864 bytes
Max No. Thread per Block	1024
Max No. Thread per Multiprocessor	2048

Table 37: GPU Specifications

B.2 Black-box testing accuracy and Normalization effect

This section presents the original classification performance of the various black-box models and examines the impact of the min-max normalization pre-processing step on some of the tested datasets. The black box models that most benefit from this operation are MLP and SVM, with a very significant incrementation in test accuracy, expecially on KEmoCon datasets. The following pages also analyze the impact of this on the specificity and acceptability levels of the multi-class CPILS approach, focusing on the most relevant results. For these reasons, all the conducted experimentations in the Section (5) are executed with this preliminary operation.

Dataset	BB Classifier	Train Accuracy	Test Accuracy	Test Accuracy	Total Miss Classifications	Class Most Miss-Classified (count)
		Original Dataset (Train-Test)		(100 samples of test set – all classes taken equally)		
Body Performance	XGB	91%	75%	72%	28	1 (12)
	XGB & Norm	92%	74%	69%	31	1 (13)
	RF	100%	74%	71%	29	1 (14)
	RF & Norm	100%	73%	72%	28	1 (13)
	MLP	**61%**	**61%**	63%	37	2 (22)
	MLP & Norm	**69%**	**67%**	64%	36	1 (15)
	SVM	**61%**	**61%**	66%	34	1 (11)
	SVM & Norm	**64%**	**63%**	65%	35	1 (14)
Thyroid	XGB	100%	99%	96%	4	0 (2) 1 (2)
	XGB & Norm	100%	99%	96%	4	0 (2) 1 (2)
	RF	100%	99%	100%	-	-
	RF & Norm	100%	99%	100%	0	-
	MLP	**96%**	**95%**	70%	30	1 (23)
	MLP & Norm	**97%**	**96%**	76%	24	1 (17)
	SVM	93%	93%	**42%**	58	1 (33)
	SVM & Norm	94%	94%	**50%**	50	1 (33)
Wine	XGB	93%	70%	69%	31	0 (13)
	XGB & Norm	93%	70%	69%	31	0 (13)
	RF	100%	73%	72%	28	0 (14)
	RF & Norm	100%	72%	71%	29	0 (14)
	MLP	**57%**	**55%**	53%	47	0 (20)
	MLP & Norm	**62%**	**60%**	53%	47	0 (19)
	SVM	**49%**	**48%**	**46%**	54	2 (34)
	SVM & Norm	**60%**	**60%**	**51%**	49	2 (22)

Table 38: The training and testing performance on the Benchmark dataset, both with and without normalization, is analyzed. A detailed examination is provided of the most misclassified classes among the 100 tested samples.

Dataset	BB Classifier	Train Accuracy	Test Accuracy	Test Accuracy	Total Miss Classifications	Class Most Miss-Classified (count)
		Original Dataset (Train-Test)		100 samples of test set – all classes taken equally)		
Sub_28	XGB	100%	100%	100%	0	-
	XGB & Norm	100%	100%	100%	0	-
	RF	100%	100%	100%	0	-
	RF & Norm	100%	100%	100%	0	-
	MLP	**78%**	**77%**	**71%**	29	1 (18)
	MLP & Norm	**93%**	**94%**	**91%**	9	2 (5)
	SVM	**49%**	**49%**	**50%**	50	2 (19)
	SVM & Norm	**85%**	**84%**	**81%**	19	2 (9)
Sub_5	XGB	100%	100%	100%	0	-
	XGB & Norm	100%	100%	100%	0	-
	RF	100%	100%	100%	0	-
	RF & Norm	100%	100%	100%	0	-
	MLP	**70%**	69%	**66%**	34	3(16)
	MLP & Norm	**96%**	96%	**96%**	4	1(3)
	SVM	**45%**	**44%**	**39%**	61	2 (25)
	SVM & Norm	**90%**	**91%**	**92%**	8	1 (4) 2 (4)

Table 39: The training and testing performance on the KEmoCon dataset Subject 28 and Subject 5, both with and without normalization, is analyzed. A detailed examination is provided of the most misclassified classes among the 100 tested samples.

Body Performance

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC
1v1	XGB	0/72 (*15) (**3)	784	677 (86.35%)	**268 (34.18%)**
	XGB & Norm	0/69 (*6)	872	723 (82.91%)	**399 (45.76%)**
	RF	0/71 (*6) (**3)	925	824 (89.08%)	375 (40.54%)
	RF & Norm	0/74 (*4)	780	631 (80.90%)	343 (43.97%)
	MLP	0/63	1121	**713 (63.60%)**	**296 (26.40%)**
	MLP & Norm	0/64	1244	**911 (73.23%)**	**448 (36.01%)**
	SVM	0/66 (*2)	957	**922 (96.34%)**	525 (54.86%)
	SVM & Norm	0/65 (*2)	1208	**1175 (97.27%)**	673 (55.71%)

Table 40: The normalization impact on the Body Performance dataset, highlighting acceptability and specificity.

Method	BB Classifier	PLAUSIBLE Avg. Proximity Distance from the Nth closest cf.			SPECIFIC Avg. Proximity Distance from the Nth closest cf.			Avg. Isolation Forest Score on ACCEPTABLE cf.	
		1st	2nd	3rd	1st	2nd	3rd	Train set avg. score	0.04 ± 0.04
								Train set avg. score NORM	0.04 ± 0.04
1v1	XGB	59.25 ± 59.04	76.60 ± 70.16	56.77 ± 52.36	42.48 ± 42.84	41.26 ± 23.71	59.89 ± 36.95	-0.02 ± 0.03	
	XGB & Norm	0.41 ± 0.30	0.47 ± 0.35	0.56 ± 0.27	0.23 ± 0.20	0.44 ± 0.20	0.59 ± 0.20	-0.03 ± 0.04	
	RF	63.01 ± 67.27	76.79 ± 71.88	78.90 ± 72.17	49.96 ± 55.14	59.87 ± 49.48	71.57 ± 50.19	-0.03 ± 0.03	
	RF & Norm	0.40 ± 0.31	0.53 ± 0.30	0.54 ± 0.30	0.26 ± 0.26	0.48 ± 0.26	0.53 ± 0.21	-0.02 ± 0.04	
	MLP	36.55 ± 29.51	49.24 ± 38.07	61.39 ± 46.26	43.51 ± 31.37	72.43 ± 55.26	63.72 ± 42.87	-0.02 ± 0.03	
	MLP & Norm	0.33 ± 0.21	0.48 ± 0.20	0.52 ± 0.18	0.22 ± 0.19	0.32 ± 0.19	0.37 ± 0.16	-0.02 ± 0.04	
	SVM	24.43 ± 12.11	42.73 ± 40.43	56.27 ± 48.39	22.76 ± 10.65	41.75 ± 38.07	47.95 ± 34.15	-0.01 ± 0.04	
	SVM & Norm	0.33 ± 0.17	0.44 ± 0.14	0.53 ± 0.16	0.16 ± 0.11	0.28 ± 0.16	0.35 ± 0.18	-0.02 ± 0.04	

Table 41: The normalization impact on the Body Performance dataset highlights the drastic reduction in proximity distances (between 0-1). The impact on the anomaly score is negligible.

Thyroid

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC			
1v1	XGB	0/96 (*4)	**595**	583 (97.98%)	**538 (90.42%)**			
	XGB & Norm	0/96 (*1)	**933**	907 (97.21%)	**849 (91.00%)**			
	RF	0/100	1754	1312 (74.80%)	**855 (48.75%)**			
	RF & Norm	0/100 (*1)	1345	1066 (79.26%)	**829 (61.64%)**			
	MLP	0/70	2727	**1837 (67.36%)**	1233 (45.21%)			
	MLP & Norm	0/76	2353	**1827 (77.65%)**	1121 (47.64%)			
	SVM	0/42 (*17)	376	240 (63.83%)	85 (22.61%)			
	SVM & Norm	0/50	1205	737 (61.16%)	196 (16.27%)			
					Class 0	Class 1	Class 2	-
1vALL	XGB	15/96	270	269 (99.63%)	135 (0.44 ± 0.44)	45 (0.16 ± 0.19)	89 (0.72 ± 0.44)	-
	XGB & Norm	15/96	234	234 (100.00%)	33 (0.15 ± 0.03)	45 (0.23 ± 0.19)	156 (0.76 ± 0.41)	-
	RF	0/100	693	**643 (92.78%)**	88 (0.43 ± 0.30)	89 (0.84 ± 0.25)	466 (1.06 ± 0.34)	-
	RF & Norm	0/100	473	**463 (97.89%)**	93 (0.73 ± 0.46)	63 (0.72 ± 0.38)	307 (0.70 ± 0.44)	-
	MLP	0/70	1792	**819 (45.70%)**	184 (0.76 ± 0.53)	260 (0.80 ± 0.41)	375 (0.76 ± 0.47)	-
	MLP & Norm	0/76	2088	**1029 (49.28%)**	341 (0.83 ± 0.42)	308 (1.19 ± 0.38)	380 (1.10 ± 0.44)	-
	SVM	0/42	880	**290 (32.95%)**	61 (0.58 ± 0.16)	0 (-)	229 (1.15 ± 0.27)	-
	SVM & Norm	0/50	1159	**814 (70.23%)**	35 (1.10 ± 0.19)	8 (0.46 ± 0.25)	771 (1.26 ± 0.26)	-

Table 42: The normalization impact on the Thyroid dataset, highlighting acceptability and specificity in 1v1 and 1vAll modalities. Here, the impact is more significant in the 1vAll mode.

Wine

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC			
1v1	XGB	0/69 (*21)	368	278 (75.54%)	209 (56.79%)			
	XGB & Norm	0/69 (*10)	462	358 (77.49%)	253 (54.76%)			
	RF	0/72 (*28)	432	296 (68.52%)	202 (46.76%)			
	RF & Norm	0/71 (*18)	366	239 (65.30%)	165 (45.08%)			
	MLP	0/53	349	**269 (77.08%)**	**186 (53.30%)**			
	MLP & Norm	0/53 (*3)	483	**377 (78.05%)**	**281 (58.18%)**			
	SVM	0/46 (*3)	191	**118 (61.78%)**	**92 (48.17%)**			
	SVM & Norm	4/51 (*33)	263	**259 (98.48%)**	**231 (87.83%)**			
					Class 0	Class 1	Class 2	Class 3
1vALL	XGB	0/69	291	**204 (70.10%)**	65 (35.26 ± 65.65)	114 (67.29 ± 108.72)	25 (104.68 ± 119.78)	-
	XGB & Norm	0/69	320	**241 (75.31%)**	120 (0.73 ± 0.30)	99 (0.64 ± 0.30)	22 (0.62 ± 0.49)	-
	RF	1/72	313	204 (65.18%)	85 (53.72 ± 83.92)	110 (110.70 ± 107.38)	9 (132.20 ± 128.41)	-
	RF & Norm	5/71	246	153 (62.20%)	68 (0.42 ± 0.29)	78 (0.44 ± 0.24)	7 (0.80 ± 0.29)	-
	MLP	0/53	188	141 (75.00%)	31 (3.68 ± 8.99)	102 (158.08 ± 137.00)	8 (15.31 ± 17.57)	-
	MLP & Norm	0/53	233	171 (73.39%)	65 (0.70 ± 0.25)	95 (0.73 ± 0.27)	11 (0.60 ± 0.36)	-
	SVM	27/46	55	**21 (38.18%)**	2 (3.68 ± 8.99)	19 (158.08 ± 137.00)	0 (0.00 ± 0.00)	-
	SVM & Norm	0/51	218	**147 (67.43%)**	84 (0.82 ± 0.26)	63 (0.66 ± 0.20)	0 (-)	-

Table 43: The normalization impact on the Wine dataset, highlighting acceptability and specificity in 1v1 and 1vAll modalities. Here, the impact is more significant on the SVM black-box model in both modalities, increasing acceptability by approximately 30%.

Method	BB Classifier	PLAUSIBLE Avg. Proximity Distance from the Nth closest cf.			SPECIFIC Avg. Proximity Distance from the Nth closest cf.			Avg. Isolation Forest Score on ACCEPTABLE cf.	
		1st	2nd	3rd	1st	2nd	3rd	Train set avg. score	0.05 ± 0.04
								Train set avg. score NORM	0.05 ± 0.04
1v1	XGB	42.29 ± 79.27	56.91 ± 74.52	162.86 ± 115.97	23.44 ± 57.81	66.87 ± 91.32	162.31 ± 117.42	-0.01 ± 0.06	
	XGB & Norm	0.49 ± 0.36	0.70 ± 0.29	0.99 ± 0.26	0.33 ± 0.32	0.70 ± 0.37	0.90 ± 0.23	-0.03 ± 0.06	
	RF	43.59 ± 58.40	43.61 ± 60.36	41.51 ± 37.03	7.65 ± 17.94	70.67 ± 85.50	127.17 ± 116.88	-0.03 ± 0.05	
	RF & Norm	0.75 ± 0.33	0.89 ± 0.24	1.00 ± 0.19	0.47 ± 0.33	0.58 ± 0.33	0.69 ± 0.29	-0.02 ± 0.05	
	MLP	43.09 ± 89.10	88.46 ± 109.68	165.54 ± 121.40	1.97 ± 8.62	15.30 ± 31.05	49.50 ± 78.37	-0.03 ± 0.05	
	MLP & Norm	0.60 ± 0.32	0.96 ± 0.21	1.09 ± 0.16	0.30 ± 0.23	0.53 ± 0.31	0.65 ± 0.32	-0.02 ± 0.06	
	SVM	118.02 ± 46.09	96.02 ± 5.67	218.18 ± 10.87	149.70 ± 93.89	229.00 ± 129.34	82.67 ± 53.94	-0.03 ± 0.05	
	SVM & Norm	0.74 ± 0.09	0.89 ± 0.07	0.95 ± 0.04	0.28 ± 0.21	0.50 ± 0.29	0.64 ± 0.32	-0.02 ± 0.07	
1vALL	XGB	3.76 ± 6.35	37.98 ± 71.68	97.35 ± 108.49				-0.03 ± 0.05	
	XGB & Norm	0.38 ± 0.31	0.65 ± 0.27	0.71 ± 0.25				-0.03 ± 0.06	
	RF	14.54 ± 28.12	58.62 ± 76.64	118.89 ± 95.61				**-0.02 ± 0.04**	
	RF & Norm	0.29 ± 0.23	0.50 ± 0.28	0.57 ± 0.25	-	-	-	**0.00 ± 0.04**	
	MLP	7.20 ± 33.59	62.54 ± 85.20	181.64 ± 117.95				-0.02 ± 0.05	
	MLP & Norm	0.44 ± 0.22	0.71 ± 0.21	0.75 ± 0.15				-0.04 ± 0.06	
	SVM	181.83 ± 50.98	102.95 ± 4.77	122.32 ± 5.67				-0.02 ± 0.06	
	SVM & Norm	0.49 ± 0.23	0.75 ± 0.17	0.83 ± 0.12				-0.04 ± 0.05	

Table 44: The normalization impact on the Wine dataset, highlighting minimality and anomaly score in 1v1 and 1vAll modalities.

Subject 28

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC			
1v1	XGB	0/100	3637	1613 (44.35%)	1211 (33.30%)			
	XGB & Norm	0/100	3026	1236 (40.85%)	616 (20.36%)			
	RF	0/100 (*14)	4369	**2243 (51.34%)**	**1602 (36.67%)**			
	RF & Norm	0/100 (*14)	4288	**2407 (56.13%)**	**1853 (43.21%)**			
	MLP	0/71	**3881**	**2707 (69.75%)**	**1666 (42.93%)**			
	MLP & Norm	0/91	**5389**	**4286 (79.53%)**	**2681 (49.75%)**			
	SVM	0/50 (*1)	**1591**	1476 (92.77%)	999 (62.79%)			
	SVM & Norm	0/81	**3864**	3414 (88.35%)	2447 (63.33%)			
					Class 0	Class 1	Class 2	Class 3
1vALL	XGB	0/100	1432	752 (52.51%)	204 (394129.34 ± 754628.22)	406 (574808.11 ± 928207.24)	142 (413821.86 ± 658975.10)	-
	XGB & Norm	0/100	1414	744 (52.62%)	436 (1.20 ± 0.29)	233 (1.07 ± 0.36)	75 (1.09 ± 0.40)	-
	RF	0/100	3443	**1146 (33.28%)**	500 (576019.42 ± 940777.02)	458 (370457.47 ± 587790.28)	188 (630699.12 ± 931305.21)	-
	RF & Norm	0/100	3495	**1339 (38.31%)**	644 (1.02 ± 0.36)	516 (1.01 ± 0.30)	179 (1.20 ± 0.27)	-
	MLP	0/71	1626	**1201 (73.86%)**	622 (583218.65 ± 707783.66)	377 (587068.34 ± 588186.42)	202 (291825.47 ± 726496.41)	-
	MLP & Norm	0/91	2132	**1546 (72.51%)**	360 (0.94 ± 0.28)	988 (0.85 ± 0.23)	198 (0.81 ± 0.29)	-
	SVM	0/50	**556**	520 (93.53%)	283 (684762.76 ± 467894.58)	161 (1194308.89 ± 453528.39)	76 (429577.65 ± 311950.75)	-
	SVM & Norm	0/81	**2525**	1917 (75.92%)	724 (1.05 ± 0.29)	853 (0.93 ± 0.23)	340 (0.89 ± 0.24)	-

Table 45: Impact of normalization on the Subject 28 dataset, highlighting acceptability and specificity in 1v1 and 1vAll modalities. The effect is more pronounced on the MLP black-box model, increasing acceptability by approximately 5% in 1v1 mode and 20% in 1vAll mode.

Subject 5

Method	BB Classifier	Instances NOT ABLE (without cfs.)	Tot. cf.	Tot. cf. ACCEPTABLE	Tot. cf. SPECIFIC			
1v1	XGB	0/100	2370	1452 (61.27%)	964 (40.68%)			
	XGB & Norm	0/100	1630	921 (56.50%)	512 (31.41%)			
	RF	0/100 (*5)	**4602**	2638 (57.32%)	1584 (34.42%)			
	RF & Norm	0/100 (*5)	**4895**	2672 (54.59%)	1759 (35.93%)			
	MLP	0/66	**3343**	**2141 (64.04%)**	**958 (28.66%)**			
	MLP & Norm	0/96	**6232**	**4321 (69.34%)**	**2161 (34.68%)**			
	SVM	0/39	**1024**	**695 (67.87%)**	**386 (37.70%)**			
	SVM & Norm	0/92 (*2)	**5298**	**3721 (70.23%)**	**2550 (48.13%)**			
					Class 0	Class 1	Class 2	Class 3
1vALL	XGB	0/100	1229	895 (72.82%)	279 (106013.29 ± 330557.44)	406 (318718.68 ± 747701.40)	175 (378931.90 ± 825392.12)	18 (3696.39 ± 3508.68)
	XGB & Norm	0/100	1146	858 (74.87%)	298 (1.18 ± 0.29)	334 (0.67 ± 0.32)	160 (0.91 ± 0.36)	66 (0.75 ± 0.31)
	RF	0/100	2592	914 (35.26%)	337 (208557.13 ± 339030.79)	363 (747292.51 ± 1164217.43)	205 (149164.77 ± 307129.71)	9 (157.82 ± 102.74)
	RF & Norm	0/100	2712	742 (27.36%)	170 (1.07 ± 0.32)	340 (0.94 ± 0.36)	219 (0.86 ± 0.43)	13 (0.85 ± 0.55)
	MLP	0/66	**1175**	**708 (60.26%)**	193 (91301.39 ± 267771.33)	88 (254085.29 ± 495947.28)	256 (606639.56 ± 663923.26)	171 (396627.72 ± 533577.85)
	MLP & Norm	0/96	**2417**	**1961 (81.13%)**	269 (1.02 ± 0.19)	1011 (0.93 ± 0.28)	611 (0.97 ± 0.31)	70 (0.71 ± 0.24)
	SVM	0/39	**413**	317 (76.76%)	63 (1066396.68 ± 521644.04)	33 (1490111.73 ± 901011.57)	121 (1608613.89 ± 331486.63)	100 (445279.77 ± 229167.09)
	SVM & Norm	0/92	**1936**	1278 (66.01%)	257 (1.11 ± 0.25)	781 (1.00 ± 0.36)	204 (1.02 ± 0.27)	36 (0.86 ± 0.18)

Table 46: The normalization impact on the Subject 5 dataset, highlighting acceptability and specificity in 1v1 and 1vAll modalities. Here, the impact is more significant on the MLP black-box model in 1v1 mode, increasing acceptability by approximately 10%.

Execution Time Prospective

DS (# Classes)	BB Classifier	Counterfactuals generation using 100 samples of test set	
		Time Elapsed [hh:mm:ss] (# Counterfactuals)	
		1V1	1VALL
Body Performance (4)	XGB	11:06 (784)	04:31 (314)
	XGB & Norm	10:09 (872)	05:58 (354)
	RF	19:09 (925)	07:59 (331)
	RF & Norm	20:09 (780)	08:41 (477)
	MLP	05:54 (1121)	13:58 (342)
	MLP & Norm	04:27 (1244)	10:44 (362)
	SVM	06:11 (957)	20:44 (253)
	SVM & Norm	07:18 (1208)	19:56 (312)
Thyroid (3)	XGB	02:03:32 (595)	01:05:06 (270)
	XGB & Norm	01:51:15 (933)	01:09:15 (234)
	RF	**04:07:43 (1754)**	01:47:33 (693)
	RF & Norm	**03:39:25 (1345)**	02:05:23 (473)
	MLP	**04:49:53 (2727)**	**02:34:25 (1792)**
	MLP & Norm	**06:10:33 (2353)**	**01:49:35 (2088)**
	SVM	**08:28:13 (376)**	**01:17:54 (880)**
	SVM & Norm	**05:49:36 (1205)**	**03:07:41 (1159)**
Wine (3)	XGB	03:39 (368)	01:13 (291)
	XGB & Norm	03:01 (462)	01:04 (320)
	RF	08:33 (432)	03:05 (313)
	RF & Norm	07:33 (366)	03:20 (246)
	MLP	**03:00 (349)**	**01:58 (188)**
	MLP & Norm	**01:36 (483)**	**00:34 (233)**
	SVM	**07:09 (191)**	**11:43 (55)**
	SVM & Norm	**14:12 (263)**	**00:39 (218)**
Subj_28 (3)	XGB	41:05 (3637)	26:04 (1432)
	XGB & Norm	43:01 (3026)	22:52 (1414)
	RF	01:12:25 (4369)	24:23 (3443)
	RF & Norm	01:08:25 (4288)	25:05 (3495)
	MLP	13:24 (3881)	08:59 (1626)
	MLP & Norm	15:53 (5389)	07:56 (2132)
	SVM	**05:46:29 (1591)**	**02:46:28 (556)**
	SVM & Norm	**41:44 (3864)**	**11:56 (2525)**
Subj_5 (4)	XGB	01:30:49 (2370)	26:45 (1229)
	XGB & Norm	01:30:37 (1630)	27:44 (1146)
	RF	01:54:59 (4602)	27:07 (2592)
	RF & Norm	01:54:09 (4895)	27:45 (2712)
	MLP	**28:50 (3343)**	09:12 (1792)
	MLP & Norm	**04:01:03 (6232)**	31:05 (2417)
	SVM	**10:01:08 (1024)**	**03:30:37 (413)**
	SVM & Norm	**05:19:15 (5298)**	**32:50 (1936)**

Table 47: The normalization impact on execution time for the tested dataset in both the modalities 1v1 and 1vAll.

References

[1] E. Tjoa and C. Guan. *A Survey on Explainable Artificial Intelligence (XAI): Toward Medical XAI,* in IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 11, pp. 4793-4813, Nov. 2021, doi: 10.1109/TNNLS.2020.3027314. https://ieeexplore.ieee.org/document/9233366.

[2] Hassija, V., Chamola, V., Mahapatra A. et al. *Interpreting Black-Box Models: A Review on Explainable Artificial Intelligence. Cogn Comput 16, 45–74 (2024).* https://doi.org/10.1007/s12559-023-10179-8.

[3] European Commission. *Artificial Intelligence Act.* Available at: https://artificialintelligenceact.eu/the-act/.

[4] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, Francisco Herrera *Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI* Information Fusion, Volume 58, 2020, Pages 82-115, ISSN 1566-2535 https://doi.org/10.1016/j.inffus.2019.12.012

[5] Qian Z, Huang K, Wang QF, Zhang XY. *A survey of robust adversarial training in pattern recognition: Fundamental, theory, and methodologies. Pattern Recogn. 2023;132.* https://doi.org/10.48550/arXiv.2203.14046

[6] Wikipedia contributors. *Brain–computer interface* Available at: https://en.wikipedia.org/wiki/Brain%E2%80%93computer_interface.

[7] Alshehri, H.; Al-Nafjan, A.; Aldayel, M. *Decoding Pain: A Comprehensive Review of Computational Intelligence Methods in Electroencephalography-Based Brain–Computer Interfaces.* https://doi.org/10.3390/diagnostics15030300.

[8] Capgemini Research Institute. *Emotional Intelligence: The Essential Skillset for the Age of AI* Available at: https://www.capgemini.com/dk-en/wp-content/uploads/sites/42/2019/10/Digital-Report-%E2%80%93-Emotional-Intelligence.pdf.

[9] Wikipedia contributors. *Emotional Intelligence* Available at: https://en.wikipedia.org/wiki/Emotional_intelligence.

[10] Team, Scientific Data Curation (2020). *Metadata record for: K-EmoCon, a multimodal sensor dataset for continuous emotion recognition in naturalistic conversations* https://doi.org/10.6084/m9.figshare.12618797.v1

[11] H. Abdi and L. J. Williams, *Principal component analysis* Wiley Interdiscipl. Rev., Comput. Statist., vol. 2, no. 4, pp. 433–459, Jul. 2010. https://doi.org/10.1002/wics.101

[12] L. Van der Maaten and G. Hinton, *Visualizing data using t-SNE* J. Mach. Learn. Res., vol. 9, no. 11, pp. 1–20, 2008.

[13] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin, *Variational autoencoder for deep learning of images, labels and captions,* in Proc. NIPS, 2016, pp. 2352–2360. https://doi.org/10.48550/arXiv.1609.08976

[14] R. K. Mothilal, A. Sharma, and C. Tan, *Explaining machine learning classifiers through diverse counterfactual explanations,* in Proc. FAT, 2020, pp. 607–617. https://doi.org/10.1145/3351095.3372850

[15] T. Laugel, M.-J. Lesot, C. Marsala, X. Renard, and M. Detyniecki, *Comparison-based inverse classification for interpretability in machine learning,* in Communications in Computer and Information Science. Cham, Switzerland: Springer, 2018, pp. 100–111. https://doi.org/10.48550/arXiv.1712.08443

[16] S. Rathi, *Generating counterfactual and contrastive explanations using SHAP* 2019, arXiv:1906.09293 https://doi.org/10.48550/arXiv.1906.09293

[17] Yiwei Shi, Qi Zhang, Kevin McAreavey, Weiru Liu, *Counterfactual Shapley Values for Explaining Reinforcement Learning* https://doi.org/10.48550/arXiv.2408.02529

[18] Guidotti, Riccardo, et al. *Local rule-based explanations of black box decision systems.* https://doi.org/10.48550/arXiv.1805.10820

[19] Guidotti, R., Monreale, A., Ruggieri, S. et al. *Stable and actionable explanations of black-box models through factual and counterfactual rules.* Data Min Knowl Disc 38, 2825–2862 (2024). https://doi.org/10.1007/s10618-022-00878-5

[20] S. Piaggesi, F. Bodria, R. Guidotti, F. Giannotti and D. Pedreschi, *Counterfactual and Prototypical Explanations for Tabular Data via Interpretable Latent Space,* in IEEE Access, vol. 12, pp. 168983-169000, 2024, doi: 10.1109/ACCESS.2024.3496114. https://ieeexplore.ieee.org/document/10753432

[21] Waggener, Bill (1995). *Pulse Code Modulation Techniques. Springer.* p. 206. ISBN 978-0-442-01436-0. Retrieved 13 June 2020. https://books.google.it/books?id=8l_o6kI3760C

[22] N. Ahmed, T. Natarajan, and K. R. Rao, *Discrete cosine transform,* IEEE Trans. Comput., vols. C-23, no. 1, pp. 90–93, May 1974. https://ieeexplore.ieee.org/document/1672377

[23] S. Kullback and R. A. Leibler, *On information and sufficiency,* Ann. Math. Statist., vol. 22, no. 1, pp. 79–86, 1951.http://dx.doi.org/10.1214/aoms/1177729694

[24] Cover, T., & Hart, P. (1967). *Nearest neighbor pattern classification.* IEEE Transactions on Information Theory, 13(1), 21–27. https://doi.org/10.1109/TIT.1967.1053964

[25] R. T. Rockafellar, *Lagrange multipliers and optimality,* SIAM Rev., vol. 35, no. 2, pp. 183–238, Jun. 1993. https://epubs.siam.org/doi/10.1137/1035044

[26] Cheul Young Park et. Al. , *K-EmoCon, a multimodal sensor dataset for continuous emotion recognition in naturalistic conversations* https://doi.org/10.1038/s41597-020-00630-y

[27] Kołodziej, Marcin & Majkowski, Andrzej & Tarnowski, Paweł & Rak, Remigiusz. (2015). *Recognition of visually induced emotions based on electroencephalography.* https://ieeexplore.ieee.org/document/7341394

[28] F. T. Liu, K. M. Ting, and Z. Zhou, *Isolation-based anomaly detection,* ACM Trans. Knowl. Discovery Data, vol. 6, no. 1, pp. 1–39, Mar. 2012. https://doi.org/10.1145/2133360.2133363

[29] https://www.kaggle.com/

[30] https://archive.ics.uci.edu/datasets

[31] https://www.kaggle.com/datasets/kukuroo3/body-performance-data

[32] https://archive.ics.uci.edu/dataset/102/thyroid+disease

[33] https://archive.ics.uci.edu/dataset/19/car+evaluation

[34] https://archive.ics.uci.edu/dataset/186/wine+quality

[35] https://scikit-learn.org/stable/supervised_learning.html

[36] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[37] https://github.com/AlterVigna/Multi_Class_cp-ils