GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

INTERPRETING NEURAL NETWORKS - DEVELOPING A METHODOLOGY FOR BANKING CASE STUDIES USING EXPLAINABLE AI

Autor: Javier Prieto Domínguez

Co-Director: David Alfaya Sanchez

Co-Director: Jaime Pizarroso Gonzalo

Madrid, Junio

UNIVERSIDAD PONTIFICIA COMILLAS
Escuela Técnica Superior de Ingeniería (ICAI)
Grado en Ingeniería Matemática e Inteligencia Artificial

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título **Interpreting Neural Networks - Developing a Methodology for Banking Case Studies Using Explainable AI** en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2024/25 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: **Javier Prieto Domínguez**   Fecha: 10/06/2025

Autorizada la entrega del proyecto

LOS DIRECTORES DEL PROYECTO

Fdo.: **David Alfaya Sánchez**   Fecha: 10/06/2025
Fdo.: **Jaime Pizarroso Gonzalo**   Fecha: 10/06/2025

A mi familia, amigos y a Lucia

UNIVERSIDAD PONTIFICIA COMILLAS

Escuela Técnica Superior de Ingeniería (ICAI)

Grado en Ingeniería Matemática e Inteligencia Artificial

Abstract

Neural networks are increasingly employed in the banking sector for tasks ranging from credit scoring to fraud detection. Despite their powerful predictive capabilities, the opacity of neural network models poses significant challenges for their interpretability. This project aims to bridge the gap between neural network architectures and their practical interpretation by developing a comprehensive methodology utilizing state-of-the-art Explainable AI (XAI) techniques.

Contents

1 Introduction

Currently, the regulations many entities face on the use of AI are very restrictive. They must be able to "explain" the decisions made by their models to justify business decisions based on those AI systems [7]. Today, the vast majority of them use models such as decision trees, random forests, and logistic regressions due to the lack of interpretability of bigger and more powerful models [8, 9]. The aim of this project is to provide a solution to solve two main problems. First, neural networks and larger models are considered "black-box" models, so we cannot fully understand the decision-making processes behind specific outputs, and therefore result useless when facing regulations. Secondly, we wish to give *actionable* solutions for differentiable classification models, such as neural networks or logistic regressions, by identifying changes in input features that could alter the model output

In the context of banking and loan applications, for example, if a loan application is denied, the solution should provide actionable insights, such as "reduce your debt by $10,000 to qualify" or "increase your salary by $5,000 to qualify." These are concrete changes in the individual's profile that can change the classification outcome. These *explanations* or actionable changes are known as *counterfactuals* [1, 2]. A counterfactual reveals what should have been different in an instance to observe a different outcome. These explanations are a clear and direct definition for local interpretability. It provides an insight into why a singular instance has been classified a certain way. For our specific case, our aim is not to solve or provide global interpretability, as the end user, or clients of a bank, do not need to know the reason a model behaves a certain way for the entire dataset

Our objective is to devise a novel counterfactual technique based on an *optimal mathematical solution* to the problem, using optimization methods and the model's derivatives to find the minimal shift needed for a given instance to change its output. We propose a innovative solution in which the end user is able to weigh how easy or difficult is to change each of the original features.

In Section 2 we discuss related works in the field of counterfactual explanations, highlighting the properties a good explanation should fulfill and the current state-of-the-art methods. In Section 3 presents the methodology used in this project, including the mathematical formulation of the problem (Subsections 3.1 and 3.2), the use of weights to shape the distance function (Subsection 3.3), the handling of actionability (Subsections 3.4 and 3.5), and the implementation details(Algorithm 1 and Subsections 3.6 and 3.8). We evaluate the proposed method in Section 4, where we analyse each of the properties we fulfill in depth and we show the website created in Section 5 Finally, we conclude the project in Section 6 and discuss future work. The code is available at `https://github.com/javiprietod/TFG`

2 Related work

Research in counterfactual explanations often highlights a set of properties a good explanation should fulfill [2]. These are:

- **Validity:** The counterfactual truly changes the classification outcome.

- **Minimality:** The counterfactual changes only the smallest set of features.

- **Similarity:** The counterfactual remains close to the original instance by some distance metric.

- **Plausibility:** The counterfactual resembles realistic feature values found in the reference dataset.

- **Actionability:** The counterfactual only changes features that can realistically be altered (e.g., debt reduction but not age).

- **Diversity:** A set of counterfactuals should offer varied options for achieving the desired outcome.

Other desirable properties of the explanation *itself* are efficiency, which speaks to the speed of the method of returning the explanations; stability, if two instances are similar, their counterfactual explanations should be similar as well; and fairness, explanation remains valid even if sensitive attributes (e.g., ethnicity) were altered [2]

Counterfactual methods can be categorized by the strategy used to create the explanations. The main strategies include Optimization-based, Heuristic-based, Instance-based, and Decision-tree-based methods [2]. While optimization-based explainers usually have outstanding results in many of the individual properties highlighted above, they usually lack a good trade-off between all of them. Heuristic, instance and decision tree based methods are usually endogenous, meaning the counterfactual returned comes from the original dataset, which provides good results for plausibility and diversity, but not for similarity. On the other hand, optimization methods are in their majority exogenous, meaning they create a new sample based on the algorithm they follow, which usually performs good on minimality and similarity while possibly leaving other properties like plausibility and actionability unfulfilled. The current state-of-the-art does not yet include an explainer that fulfills all the desirable properties [2]

The task of finding an algorithm that fulfills all of the properties has been attempted in many papers. The most well-known technique in counterfactual space is WACH [1], one of the first papers to publish and propose these explanations. They propose a loss function, adopted by other explainers, consisting in a distance function and a cross-entropy loss, which they try to minimize with a common optimizer such as SGD [17] or Adam [18]. While it is a good starting point, it does not guarantee the optimal solution, and it is not guaranteed to converge to a solution that fulfills all the properties mentioned above.

Among the more rigorous contributions, ACTREC (Actionable Recourse) [4], is one of the first to address which features can be changed and which must remain fixed. ACTREC formulates the counterfactual generation as a discrete optimization problem using integer programming, ensuring feasibility, actionability, and minimal cost over a discretized space.

It guarantees global optimality within its constraints but is limited to linear models and can become computationally intensive for high-dimensional problems

Another notable method is DiCE [22], which uses a differentiable loss function to generate a set of diverse counterfactuals. DiCE ensures validity and similarity while promoting diversity using Determinantal Point Processes [20]. While many methods focus on finding the closest counterfactual with high precision, DiCE offers users a broad range of actionable options not ensuring the closest counterfactual.

Finally, SGNCE (Scaling Guarantees for Nearest Counterfactual Explanations) [23] guarantees the closest actionable counterfactual by formulating the search as a mixed-integer programming (MIP) problem, exploring the full solution space and supporting variables of mixed types. This approach provides formal guarantees on coverage, feasibility and similarity, making it robust but computationally demanding. Unlike SGNCE, our method achieves similar goals through a continuous, second-order optimization strategy that avoids combinatorial search. By doing so, we gain significant efficiency and maintain similarity, while still enabling tailored cost functions and domain-specific adaptations; an advantage in real-world sectors like finance.

Other common XAI methods like LIME [5] and SHAP [6] exist, but they provide explanations by assigning feature importance rather than producing clear, actionable changes in the input that could alter the classification. They both have adapted to the counterfactual world [21] but are mainly adapted to textual data and it has not been compared with many of the algorithms in the literature [2]

Our objective is to devise a novel counterfactual technique based on the *optimal mathematical solution* to the problem, using optimization methods and the model's derivatives to find the minimal shift needed for a given instance to change its output, fitting into the optimization-based category. We not only fulfill this similarity to the original instance by minimizing a distance function, validity and plausibility are guaranteed as well with the intrinsics of the method. Other features like minimality (of features changed), actionability, and diversity are achieved through a set of weights that impact the decisions and inner workings of the method in every step. This gives the ability, to either the entity that provides the service or the end-user, to determine which features are actually changeable and how difficult it is to change each of them. For example, in a certain situation, the user might feel more comfortable increasing their salary than reducing the loan amount. As these weights can be changed, our method has the ability to provide a wide spectrum of explanations, adapting to every situation and the needs and interests of different parties.

3   Methodology

The purpose of this project is to create a innovative XAI technique for differentiable models. We will initially restrict the analysis to differentiable models since the method requires the use of derivatives for optimization matters and for that, differentiability is needed. We will also assume that it is a binary classification model with a threshold to determine the change in the

target variable. This new technique, or method, falls into the Counterfactual Explanations family of XAI. As mentioned earlier in the motivation, a counterfactual reveals what should have been different in an instance to observe a diverse outcome

Following the definition of a counterfactual, and inspired by some of the main properties mentioned earlier, the innovative method aims to fulfill as many properties as possible. With the validity and similarity properties in mind, which refer to counterfactuals that succeed in changing the classification output with the minimum change needed, the mathematical optimization method of Lagrange multipliers [12] comes up with a direct application to the problem. In this optimization method, it is possible to find local minimum (or maximum but it does not apply to the problem) of a function with certain equation constraints. In the case of the counterfactual, we want to find a point that minimizes the distance with respect to the original instance, subject to the condition of changing its classification output. This would fulfill both of the properties mentioned above. The equation with the Lagrangian multiplier becomes:

$$\mathcal{L}(\mathbf{x}, \lambda) = C(\mathbf{x}) + \lambda \left(M(\mathbf{x}) - \theta \right) \tag{3.1}$$

where

- x is the new instance (the counterfactual) with m features,

- $C(\cdot)$ is a weighted cost function (distance) measuring how far x is from its original instance,

- $M(\cdot)$ is a differentiable model,

- θ is the threshold for changing the classification,

- λ is the Lagrange multiplier.

The base of the cost/distance function is the weighted *L2* norm, which can be defined as follows:

$$C(\mathbf{x}) = \sum_{i=1}^{m} \mathbf{w}_i \cdot (\mathbf{x}_i - \mathbf{x}_i^0)^2 \tag{3.2}$$

where x^0 is the original instance and w is the weight vector

We will discuss added regularizers added to this cost function later, but the main idea is to have a distance function that can be weighted by the user, or the entity using the method, to determine how easy or difficult it is to change each feature. This is a key part of the method, as it allows for flexibility and customization in the counterfactual generation process.

3.1 Mathematical solution

The minimum value of $\mathcal{L}$ is found when the partial derivatives w.r.t. $\mathbf{x}$ and λ are equal to 0. This implies:

$$F(\mathbf{x}, \lambda) \;=\; \nabla\mathcal{L}(\mathbf{x}, \lambda) \;=\; 0 : \begin{cases} \nabla C(\mathbf{x}) \;-\; \lambda \cdot \big(\nabla M(\mathbf{x})\big) \;=\; 0, \\ -\,M(\mathbf{x}) \;+\; \varepsilon \;=\; 0. \end{cases} \tag{3.3}$$

To solve this problem we will use the *Newton-Raphson* method to solve this mathematical problem, ensuring we reach the optimal solution. Newton's method is used to find the roots of a function, or the solution to the equation $f(x) = 0$. In optimization, with $f \in C^2$, we are trying to find the roots of f', or the solutions to $f'(x) = 0$, known as critical points. These points can be minima, maxima or saddle points, but in the case of this problem we will try to find the minima of the function f.

For this, Newton-Raphson's method of optimization uses the first and second term of the Taylor expansion of the function to find that point iteratively [11]. For 1 dimension, or 1 variable, the update rule is $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$. As seen in (3.3), we have at least two variables (x and λ), so the we have to use the generalization of the iterative scheme using the hessian as the second derivative. Thus, the new iterative scheme obtained becomes

$$\big(\mathbf{x}_{n+1}, \lambda_{n+1}\big) \;=\; \big(\mathbf{x}_n, \lambda_n\big) \;-\; \mathbf{H}^{-1}\big(\mathcal{L}(\mathbf{x}_n, \lambda_n)\big) \;\cdot\; \nabla\mathcal{L}\big(\mathbf{x}_n, \lambda_n\big).$$

If we set $F(\mathbf{x}, \lambda) \;=\; \nabla\mathcal{L}(\mathbf{x}, \lambda)$, the equation simplifies to

$$\big(\mathbf{x}_{n+1}, \lambda_{n+1}\big) \;=\; \big(\mathbf{x}_n, \lambda_n\big) \;-\; \mathbf{J}\big(F(\mathbf{x}_n, \lambda_n)\big) \;\cdot\; F\big(\mathbf{x}_n, \lambda_n\big),$$

where

$$\mathbf{J}\big(F(\mathbf{x}_n, \lambda_n)\big) \;=\; \begin{pmatrix} H\big(C(\mathbf{x}_n)\big) \;-\; \lambda_n \cdot H\big(M(\mathbf{x}_n)\big) & -\nabla M(\mathbf{x}_n) \\ -\nabla M(\mathbf{x}_n) & 0 \end{pmatrix}.$$

Above, $H(\cdot)$ denotes the Hessian (second derivative) with respect to $\mathbf{x}$, and $\nabla(\cdot)$ denotes the gradient (first derivative)

A similar method of optimizing a loss function has been done in the past (BFGS [13]), but due to the computational complexity of calculating second derivatives, most of the work has been with its approximations. In this case, we will employ *autograd*, PyTorch's automatic differentiation engine to calculate these second derivatives efficiently, eliminating the need for approximations. This could be sub-optimal for very big datasets, but because for now we will only be working with the numerical columns of the datasets, the problem is not very relevant for now.

All of this will be implemented programmatically in `PyTorch`, using neural networks (or logistic regressions if there are no hidden layers), which, using activations like *sigmoid* or *tanh*, are at least twice-differentiable

Notwithstanding, this mathematical approach has some challenges. First, in some iterations of some datapoints, the hessian can become singular or ill-conditioned, meaning that

the inverse does not exist or it is very big and the update step explode. This causes problems to the overall convergence of the method for that certain point. Second, many real-life datasets include variables like integer and categorical columns. The method works very well with continuous variables but it does not have a direct and easy solution for dealing with these column types.

3.2 Singular Matrices

As discussed, one of the main problems when taking this optimization-based generation is the possibility of singular or ill-conditioned Hessians. This can most tipically happen because of two main things. If we have two variables that have a high correlation, the hessian will most likely have a very small eigenvalue associated, which means that the matrix could become ill-conditioned or even singular. This is very easily solved by performing a bit of exploratory data analysis and removing highly correlated variables. This is already a common practice in the data science world, so it is assumed that it should not produce any problems.

The second problem is harder to solve, and it is the one we will be discussing in this section. This problem appears when the gradient of the model is very close to 0, or the model has a flat curvature around a particular input. This leads again to a small eigenvalue, and the matrix becomes ill-conditioned. We explored some techniques like using the pseudo-inverse, damping techniques or switching to first-order updates for the affected iterations, which are common approaches to solving this problem [24], but the experiments were not successful. To mitigate this issue, we implement an alternative update strategy when this type of ill-conditioning in the Hessian is detected.

The alternative update step devised consists of using the normalized gradient of the model's output with respect to the input features to update.

$$\begin{cases} \mathbf{x}_{n+1} = \mathbf{x}_n - \frac{\nabla M(\mathbf{x_n})}{|\nabla M(\mathbf{x_n})|} \\ \lambda_{n+1} = \lambda_n \end{cases} \tag{3.4}$$

This update step is designed to move the input in the direction of the gradient of the model's output with respect to the input features, but more importantly, it works as a nudge in a direction to exit the flat region of the model. We have found that this update step is very effective in practice, as it allows the algorithm to continue making progress towards a valid counterfactual even when the Hessian is ill-conditioned.

3.3 How Weights Shape the Distance

In our counterfactual explanation framework, the distance function (3.2) quantifies how "far" a candidate counterfactual $\mathbf{x}'$ lies from the original instance $\mathbf{x}$, where $w_i \geq 0$ encodes the relative "cost" or difficulty of changing feature i.

- **High weights** ($w_i \gg 1$) make changes in feature i "expensive", discouraging the optimization from selecting that feature unless it is indispensable for flipping the model output.

- **Low weights** ($w_i \approx 0$) make changes in feature i "cheap", biasing counterfactuals toward modifying x_i first.

By tuning the vector **w**, practitioners can encode domain knowledge about which features are easy or hard to change in the real world, and users can tune their explanations by encoding their desirability or actionability to alter some features over others.

In practice, one might begin with all $w_i = 1$ (equal cost), inspect the resulting explanation, and then increase w_i for features the end user finds unrealistic or non-actionable, thereby refining the counterfactuals to be truly *actionable*. We will analyse actionability in Section 4.5.

3.4   Discrete features

To address discrete variables, we found that imposing a regularizer to the distance function worked the best for the task. The regularizer is designed to take value 0 at the integer points. We explored different trigonometric funcions, such as sines and cosines with periods matching the integers, but some features were getting stuck on the maxima of that function, as it has derivative 0 as well and Newton's method is designed to find critical points in general. One of the functions proposed for solving the discrete challenge faced was
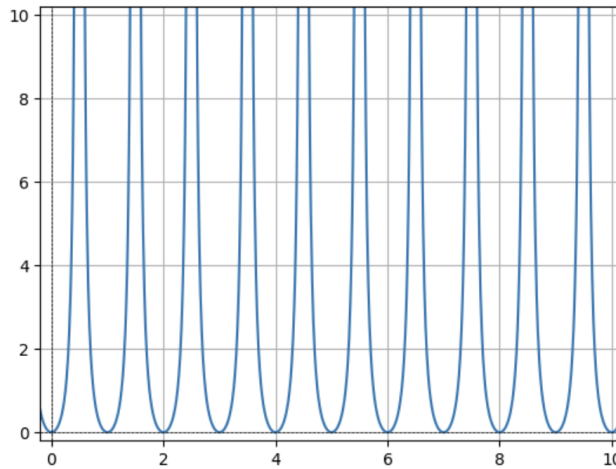
$$tan(\pi * x)^2 \tag{3.5}$$

that looks like

Figure 1: Tangent regularizer function

This function takes the value 0 in the integer points, but more importantly, they are minima, which are critical points in the newton's method. This means that iteratively, the features where the regularization is enforced will end up in one of those points because the Newton's method aims to find the points that make $f'(x) = 0$. This function works well in

practice and with the newton method, but we proposed a better solution that works even better in practice. The function proposed is

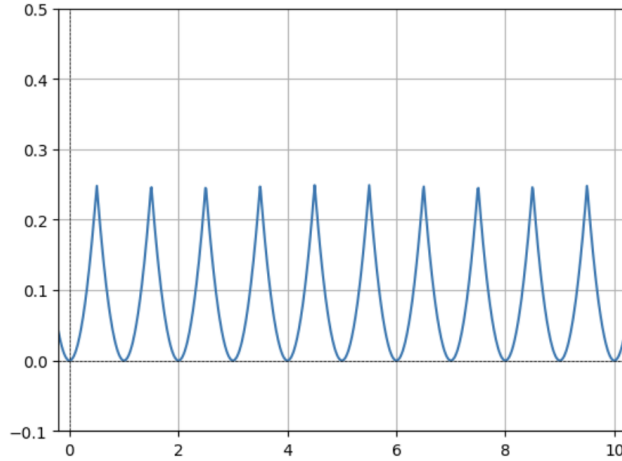$$(x - \lfloor x \rceil)^2 \tag{3.6}$$

that looks like

Figure 2: Round regularizer function

We say it works better beacause it reaches the same solution as the tangent function, but it is much more stable, faster and easier to compute. As the tangent funcion takes the value ∞ in the values exactly between integers, the gradients it takes are very big and the updates are very big as well, which leads to more inestability. The round function, on the other hand, takes the value 0 in the integers and 0.25 as its maximum, so it is much more stable and easier to compute. In theory, any function whose derivative is only 0 in the integers should work.

This regularizer is added to the cost function, so the new cost function becomes

$$C(\mathbf{x}, \mathbf{x}', \mathbf{w}) = \sum_{i=1}^{m} \mathbf{w}_i \cdot (\mathbf{x}_i - \mathbf{x}'_i)^2 + \sum_{i=1}^{m} (x - \lfloor x \rceil)^2 \cdot \mathbf{1}_{\{x_i \in \mathbb{Z}\}} \tag{3.7}$$

A common practice in data-science is to standardize the data, meaning that the features are transformed to have mean 0 and standard deviation 1, helping the model to converge faster and better. In the case of (3.7), the regularizer only works if the features are not standardized, as the periodic function is designed to have these critical points in the integers. If the features are standardized, the regularizer will not work as expected. In this case, we can use a scaled version of the regularizer, which is designed to work with standardized features. The scaled version of the regularizer is defined as

$$C(\mathbf{x}, \mathbf{x}', \mathbf{w}) = \sum_{i=1}^{m} \mathbf{w}_i \cdot (\mathbf{x}_i - \mathbf{x}'_i)^2 + \sum_{i=1}^{m} (x \cdot std_i + mean_i - \lfloor x \cdot std_i + mean_i \rceil)^2 \cdot \mathbf{1}_{\{x_i \in \mathbb{Z}\}} \tag{3.8}$$

where std_i and $mean_i$ are the standard deviation and mean of the feature i respectively, previously calculated with the whole dataset before training the model. Now we can use the regularizer to penalize the distance function for discrete features, ensuring that the counterfactuals generated will have integer values for those features.

Nevertheless, we will not use this regularizer in every iteration. This regularizer is not designed to find the optimal solution with the integer features, it is designed to find *any* integer solution. For that, we want to approach the optimal solution first without taking into account the integer features, so that the solution with integers comes close to the optimal solution. Then, we incude the regularizer in the cost function, so the algorithm converges to the closest solution in which the integer columns are very close to being integers. When it is reaching the optimal solution with the regularizer, we disable the regularizer, round the features to the closest integer and disable them by setting the weights to ∞, so that it is virtually impossible to change that feature. After that, we continue the optimization process with the variables left to reach the optimal solution with the integer features frozen. A detailed algorithm of the process is shown in Algorithm 1.

3.5   Enforcing Plausibility

Another important property of counterfactuals is plausibility, which refers to the fact that the counterfactual should be a realistic and feasible instance. In our case, we want to ensure that the counterfactuals generated are plausible in the context of the problem at hand. To achieve this, we can use a set of constraints that limit the range of values that the features can take. These constraints can be defined based on domain knowledge, expert opinion or the data distribution in general. For example, if we are working with a dataset of bank customers, we can set constraints on the range of values that the features can take, such as the minimum and maximum values for each feature. This way, we can ensure that the counterfactuals generated are plausible and realistic in the context of the problem at hand

In practice, we can implement these constraints by clamping the values of the features to their respective ranges, as we are lacking expert knowledge. This means that if a feature value goes below its minimum or above its maximum, it will be set to the closest valid value.

3.6   Threshold

The threshold is a key part of the method. For starters, it is the value that determines whether the model output changes or not. The threshold can be set to any value, but it is usually set to 0.5 for binary classification problems. Abstracting ourselves from the mathematical problem and counterfactual generations, we want this to be useful in a real world scenario.

Many times, entities change the model they use to classify instances and make business decisions based on the model output. As the counterfactual method is designed to find the minimal change given that the model's output is the threshold, if we match the threshold to the *business decision threshold* (the one the banks use to finally determine whether to give out a loan or not), it creates a risk of the counterfactuals not being useful in the future

because of those model changes. For this reason, we can set the threshold to be $0.5 + 10^{-5}$ or $0.5 + 10^{-7}$, which is a small value that ensures that the counterfactucals generated will be valid and useful in the future.

3.7   Obtaining the Counterfactuals

We can now obtain the counterfactuals by iteratively applying the Newton-Raphson method to the Lagrangian function (3.1) with the cost function (3.8). The algorithm is detailed in Algorithm 1

3.8   Stopping Criteria

The Newton–Raphson loop (Algorithm 1) terminates when *all* of the following safeguards are satisfied:[1]

1. **Feasibility of the classifier constraint.** We stop only after the current point $\mathbf{p}_{\text{new}}$ has crossed the decision boundary, or equivalently, when the $threshold - M(\mathbf{p}_{\text{new}}) \leq 0$. This criterion guarantees that the produced counterfactual is indeed assigned to the desired target class.

2. **Sufficiently small Newton step.** The relative update $\frac{\|\delta\|}{\|[\mathbf{p}_{\text{new}}, \lambda]\|}$ must fall below a tolerance ε (we use $\varepsilon = 10^{-6}$).

3. **Integer–rounding completion.** If the discrete regulariser is active, we allow extra iterations after conditions 1 and 2 have been satisfiesd in order to round every integer feature and re-optimise the continuous ones while the integer ones are frozen.

4. **Fail-safe epoch limit.** Finally, a hard cap of $\texttt{max\_epochs} = 100$ avoids endless cycling in pathological cases, in line with standard practice in large-scale optimisation libraries [25] and Newton solvers used in engineering packages.

4   Evaluating the algorithm

It is important to compare the algorithm with other state-of-the-art methods in the literature to evaluate its performance and effectiveness. We will use several metrics to evaluate the algorithm, including some of the properties highlighted earlier. The datasets used for the evaluation of the algorithm are the following:

- **Loan** [14]: $255\,347$ credit applications with 17 predictive attributes (8 integer, 2 continuous and 7 categorical). Our experiments use two subsets with 8 features (some of the actionable numerical features) and 24 features (with one-hot encoding).

---

[1]Classical discussions of termination for Newton-type methods can be found in [24]

Algorithm 1 Newton's Method for Counterfactual Explanations

1: **function** NEWTON_OPTIMIZATION($\mathbf{p}, M, w$)
2:    $\mathbf{p}_{\text{new}} \leftarrow \mathbf{p}$, $\lambda \sim N(0,1)$, **continue** $\leftarrow true$, $first\_time \leftarrow true$, $thres\_term \leftarrow$ threshold $- M(\mathbf{p}_{\text{new}})$
3:    **while continue and** epochs $<$ max_epochs **do**
4:        **if** $\sum(w \neq 0) = 1$ **then**   ▷ single active feature
5:            $\delta \leftarrow \left[\dfrac{M(\mathbf{p}_{\text{new}}) - \text{threshold}}{\nabla M(\mathbf{p}_{\text{new}})}, 0 \right]$
6:        **else**
7:            **if** $|thres\_term| < 0.1 \wedge first\_time \wedge$ epochs > 1 **then**   ▷ close to threshold
8:                $reg\_int \leftarrow true$
9:                $first\_time \leftarrow False$
10:           **end if**
11:           Reset gradients of $\mathbf{p}_{\text{new}}$ and λ
12:           **function** FPLFUNC($\mathbf{x}, \lambda$)   ▷ first-order derivative of Lagrangian
13:               $g_d \leftarrow \nabla C(\mathbf{x})$, $g_r \leftarrow \nabla M(\mathbf{x})$
14:               **return** $\left[g_d - \lambda\, g_r, \ (\text{threshold} - M(\mathbf{x})) \right]$
15:           **end function**
16:           $\mathbf{fpl} \leftarrow$ FPLFUNC($\mathbf{p}_{\text{new}}, \lambda$)
17:           $J \leftarrow$ Jacobian(FPLFUNC, $(\mathbf{p}_{\text{new}}, \lambda)$)
18:           **if** $\|J_\lambda\|_\infty < \varepsilon$ **then**   ▷ ill-conditioned
19:               $\delta \leftarrow thres\_term \cdot \left[\frac{\nabla M(\mathbf{p}_{\text{new}})}{|\nabla M(\mathbf{p}_{\text{new}})|}, 0 \right]$
20:           **else**
21:               $\delta \leftarrow J^{-1} \cdot \mathbf{fpl}$ ▷ Newton step
22:           **end if**
23:       **end if**
24:       $\mathbf{p}_{\text{new}}[w_{active}] \leftarrow \mathbf{p}_{\text{new}}[w_{active}] - \delta[w_{active}]$
25:       $\lambda \leftarrow \lambda - \delta_{-1}$
26:       **if** $|thres\_term| < 0.1 \wedge$ epochs $> 1 \wedge reg\_clamp$ **then**
27:           Clamp $\mathbf{p}_{\text{new}}$ to valid bounds
28:           Deactivate out-of-bounds feature in w
29:       **end if**
30:       Evaluate $thres\_term \leftarrow$ threshold $- M(\mathbf{p}_{\text{new}})$
31:       $epochs \leftarrow epochs + 1$
32:       **continue** $\leftarrow (thres\_term > 0) \vee \dfrac{\|\delta\|}{\|[\mathbf{p}_{\text{new}}, \lambda]\|} > \varepsilon$
33:       **if not continue and** reg_int **then**
34:           Round integer features; disable reg_int; deactivate integer features
35:           **continue**$\leftarrow$true
36:       **end if**
37:    **end while**
38:    **return** $\mathbf{p}_{\text{new}}$
39: **end function**

- **Spambase** [15]: $4\,601$ e-mail messages described by 57 continuous TF–IDF word-frequency features plus one binary target.

- **Santander Customer Transaction** [16]: $200\,000$ customer records with 200 continous attributes. We evaluate both the full 200-feature set and a trimmed version with the 100 most informative variables.

4.1   Validity

Validity is the property in counterfactual explanations that measures whether the counterfactual changes the classification output of the model. It is the most important property of counterfactual explanations, as it is the one that ensures that the counterfactual is actually a counterfactual. In our method, we ensure validity by using the Lagrangian multiplier in the optimization problem. The Lagrangian multiplier is used to enforce the condition that the classification output changes, which is the condition for a counterfactual to be valid.

4.2   Similarity

Similarity is the property in counterfactual explanations that measures how close the counterfactual is to the original instance. There are many ways in which we can measure similarity. Some methods use the *L1* norm whole others use the *L2* norm, which is the one we are using in our method. We claim that our method obtains the most similar counterfactual to the original instance given that the classification output changes. This is because Newton's optimization methods finds the solution to (3.3), finding the x and lambda in which there is a critical point of the Lagrangian function (3.1). We have to first ensure that the solution given is a local minimum and not a local maximum or a saddle point.

For this we will apply a grid check of the distances of points close the solution found. It is crucial to note that that the points evaluated have to fulfill the condition of changing the classification output. Checking this grid around the solution found, we can ensure that the solution is a minimum and not a maximum or a saddle point. If the solution is a minimum, then the distance between the original instance and the counterfactual is the smallest possible distance that changes the classification output. This means that our method finds the most similar counterfactual to the original instance given that the classification output changes. When enforcing the similarity property with integers, we do the same but with rounding the pertinent features to the closest integer.

Newton's method is designed to obtain these critical points, and we have ensured that the solution found is a local minimum. We can also check to see if it is the global minimum with the same technique, but we cannot guarantee it, as we don't have the computing power to check the entire space of possibilities.

4.3   Plausibility

Plausibility is often treated informally in counterfactual-explanation work. It is not yet a well defined metric and many methods lack a way of justifying that property [Keane et al., 2021]. This paper proposes as well a way of checking for invalid or bad counterfactuals, which are those that are too dissimilar to the original data distribution. They propose to use methods like Local Outlier Factor [19] to check whether the counterfactuals are in-distribution or not.

We assessed each generated counterfactual with two sanity checks. The first one is checking whether the counterfactual is *plausible* in the sense of being within the empirical bounds of the training data, checking if its variables are bigger than their minimum value in the dataset and smaller than their maximum. The second one checks whether it is *in-distribution* according to the LOF model fitted on the training set.

Both conditions were true for every record in the test set, indicating that the data respect domain bounds and exhibit no density anomalies under LOF (using 20 neighbours and $\alpha = 0.1$).

4.4   Efficiency

Efficiency is a big part of any algorithm. It is one of the desired properties of any counterfactual explanation method. We can analyse our method in terms of efficiency by focusing on the possible bottlenecks of the algorithm. The main bottleneck of the algorithm is the computation of the Hessian matrix, which is done using PyTorch's autograd. As we mentioned earlier, it is not a problem for now, as we don't work with very big datasets and we only use a subset of the features. The other bottleneck is the number of iterations needed to reach the optimal solution. This is something that we can control by setting a maximum number of epochs. When trying out the algorithm, we found that the number of epochs needed to reach the optimal solution is usually around 5-15 epochs, depending on if we are applying the integer regularizer or not. We have decided to put a maximum of 100 epochs just in case, but when reaching that number of epochs in past iterations of the algorithm, there was something wrong that needed to be fixed; and either way when reaching that number of epochs, update step was too small to make a difference.

When comparing the efficiency of our method with other state-of-the-art methods, we found that our method is much more efficient than most of them. For example, DiCE [22] takes around 0.4s to output a counterfactual, SGNCE [23] takes around 20-30s to output a counterfactual, while our method takes on average 0.014s to output a counterfactual.

We have analysed the efficiency of our method in terms of time and number of epochs needed to reach the optimal solution for different number of features, to check for the scalability of the algorithm.

Features	✗ Integer	✓ Integer
8	0.0143 ± 0.0049	0.0222 ± 0.0144
24	0.0331 ± 0.0216	0.0417 ± 0.0157
57	0.0806 ± 0.0297	0.0998 ± 0.0383
100	0.0970 ± 0.0424	0.1403 ± 0.0561
200	0.1661 ± 0.0528	0.2491 ± 0.0706

Table 1: Average training time per run (seconds) for 176 instances
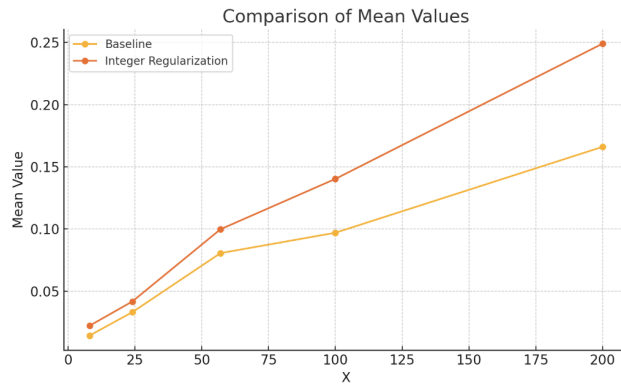
Figure 3: Time comparison for using integer regularization

Features	✗ Integer	✓ Integer
8	6.1818	8.9886
24	6.9653	9.9545
57	10.3594	11.0595
100	4.4674	6.2065
200	4.7283	6.4783

Table 2: Average epochs per run for 176 instances

Table 1 shows that the time grows almost linearly with the dimensionality of the search space: increasing the number of features from 8 to 200, a 25× jump, multiplies the average run-time by only 11.6× ($0.014\,\text{s} \rightarrow 0.166\,\text{s}$). The small standard deviations confirm that these averages are representative across the 176 test instances.

Adding the integer regularization increases the runtime in $\approx$ 30-50% throughout the data. This extra time is spent on the additional epochs taken to apply the integer regularizer explained in Section 3.4. Even in the extreme $d = 200$ case the full version still delivers a counterfactual in well under a third of a second, keeping the method responsive for interactive use.

It is important to note that the experiments with 8 and 24 features come from a loan dataset [14], the 57-feature point corresponds to the Spambase dataset [15], and the 100 and 200 settings belong to the Santander Customer Transaction Prediction dataset [16].

This explains the relatively big change in the average epochs from 100-200 epochs to 8-24 epochs: the dataset makes a difference in the number of epochs needed to reach the optimal solution.

Across all settings the optimiser finds a feasible solution in 4-11 epochs, far below the conservative cap of 100. Integer regularisation increases the count by roughly two epochs on average, but the absolute numbers remain very small (Figure 3 illustrates the resulting trade-off). The proposed method delivers a counterfactual in less than 0.25 s for inputs with up to **200 features**, converging in less than gradient steps even when discrete constraints are enforced. Combined with the stability improvements reported in Section 4.6, these results confirm that the algorithm is both *fast*, *scalable* and , making it well suited for real-time recourse in practical applications.

4.5 Actionability

Like plausibility, actionability is not a well defined metric in the literature either. Actionability is defined as the property of a counterfactual explanation that measures whether the counterfactual can be acted upon by the user [2]. Like every explainer, this is one of the most important properties after validity, as the end user is the one that will act according to the counterfactual explanation.

One of the most important requisites for a counterfactual to be actionable is that the features that are changed are actually changeable by the user. For example, if we are trying to explain a loan application, the features that are changed should be those that the user can actually change, such as the income, the amount of debt, etc. No counterfactual explanation should change features that are not changeable by the user, such as the age, race, sex, etc. This poses no problem for our method, as we can select the features that we want to change in the counterfactual explanation by *deactivating* the features that we don't want to change in the weight vector.

Another important aspect of actionability is being able to generate counterfactuals that are capable of yielding integer values for discrete features. For example, a loan term feature or the number of credit lines cannot take continuous values, so the counterfactual explanation should generate integer values for those features. In our method, we can enforce this by using the regularizer explained in Section 3.4, which penalizes the distance function for discrete features and ensures that the counterfactuals generated will have integer values for those features by rounding them to the closest integer when the optimization process is close to the optimal solution.

In other papers, as well as using the common definition of actionability, they have provided more extensive definitions and metrics for this property. One of the most mentioned is aligned with the diversity property [22]. If a counterfactual method is able to generate multiple counterfactuals that are all valid and actionable, then it is considered to be more actionable

than a method that can only generate one counterfactual. This is because the user can choose the counterfactual that is most suitable for their needs, and therefore the method is more flexible and adaptable to the user's needs.

Many methods in the literature are able to generate multiple counterfactuals by their original definition of the solution they provide. In our case, as we are solving an optimization problem, only one counterfactual is generated for a given instance and set of weights. However, by changing the weights, we can generate multiple counterfactuals that are all valid and actionable by the original definition. Other methods in the literature are able to generate multiple counterfactuals by using a diversity metric, but we have revolutionised the definition of diversity and actionability at the same time. Not only are we able to generate multiple counterfactuals, but we are able to generate the most suited counterfactual for the user, as they are able to specify which features are easier to change than others. This, aligned with the fulfillment of the similarity property, makes our method one of the most actionable methods in the literature. In summary, if an actionable counterfactual exists, meaning that the user is able to make the changes proposed, our method will find it by having a back-and-forth with the user.

Other definitions of actionability, introduced in [4] is the availability or success-within-budget. This define the metric as the amount of counterfactuals that have a cost lower than a certain threshold. As we have mentioned, our method is able to find the minimum cost counterfactual, and because the metric is budget-dependent, we are able to fulfill it depending on the budget set. If the *budget of change* is too low, the method will not be able to find an actionable counterfactual (by this definition), because it does not exist.

		w_{Loan}	
	0.1	1	10
w_{Inc} 0.1	+11.14%	+18.09%	+19.29%
1	+2.30%	+11.14%	+18.09%
10	+0.24%	+2.28%	+11.14%

Table 3: Percentage change in Income varying weights.

		w_{Loan}	
	0.1	1	10
w_{Inc} 0.1	-11.40%	-1.85%	-0.19%
1	-23.53%	-11.39%	-1.84%
10	-26.34%	-23.55%	-11.39%

Table 4: Percentage change in LoanAmount varying weights.

In Table 3 and Table 4 we can see how the percentage change in the Income and LoanAmount features changes depending on the weights of their respective features in the

Loan Dataset [14]. The first column is the weight of the Income feature, and the first row is the weight of the LoanAmount feature. The values in the table are the percentage change in the Income and LoanAmount features when changing the weights of the other features. We can see that when we increase the weight of the Income feature, the percentage change in the Income feature increases, while the percentage change in the LoanAmount feature decreases. 0.1 and 10 are the minimum and maximum weights we have decided but if needed we could increase the range of weights to see how the percentage change in the features changes.

We can also see that the percentage changes are very similar for proportional weights, meaning that the weights are not absolute and they work relative to each other. It is important to note that the values for the weights presented in the tables are not the only ones that can be used, but they are the ones that we have used for the experiments.

4.6   Stability

Stability evaluates how sensitively our explainer reacts to small variations in the input. Formally, given an instance $\mathbf{x}$ and its neighbourhood $\mathcal{N}_x$, the metric measures the average distance between the set of counterfactuals returned for $\mathbf{x}$ and those obtained for every $\mathbf{z} \in \mathcal{N}_x$. If two very similar instances receive vastly different counterfactual explanations, the explainer is deemed *unstable*. Conversely, a value close to 1 means that nearby points are mapped to almost identical counterfactuals, signalling robust and trustworthy behaviour. The formula they use to calculate the stability of the models in [3] is:

$$max \frac{\|e_x - e_{x'}\|}{\|x - x'\|}, \ \forall x' \in \mathcal{N}_x \tag{4.1}$$

2

Features	✗ Integer	✓ Integer
loan [14]	0.9414 ± 0.0318	2.5196 ± 1.7558
spam [15]	0.7968 ± 0.1320	0.8119 ± 0.1326
santander [16]	0.9948 ± 0.0072	0.9951 ± 0.0068

Table 5: Average stability per dataset using neural networks

Table 5 reports the average stability (and its standard deviation) obtained with our neural-network explainer on the three benchmark datasets over 176 instances. The integer column correspond to calculating the metric using the integer regulariser described in Section 3.4 and detailed in Algorithm 1.

---

[2]This notion was introduced in the counterfactual-XAI literature as "if two instances are similar, their counterfactual explanations should be similar as well" [3].

Loan dataset. Without integer constraints the explainer is already stable (0.94 ± 0.03), yet activating the discrete regulariser more than doubles the score to 2.52 on average. This dataset has a lot of integer features, so it explains the big difference in stability.
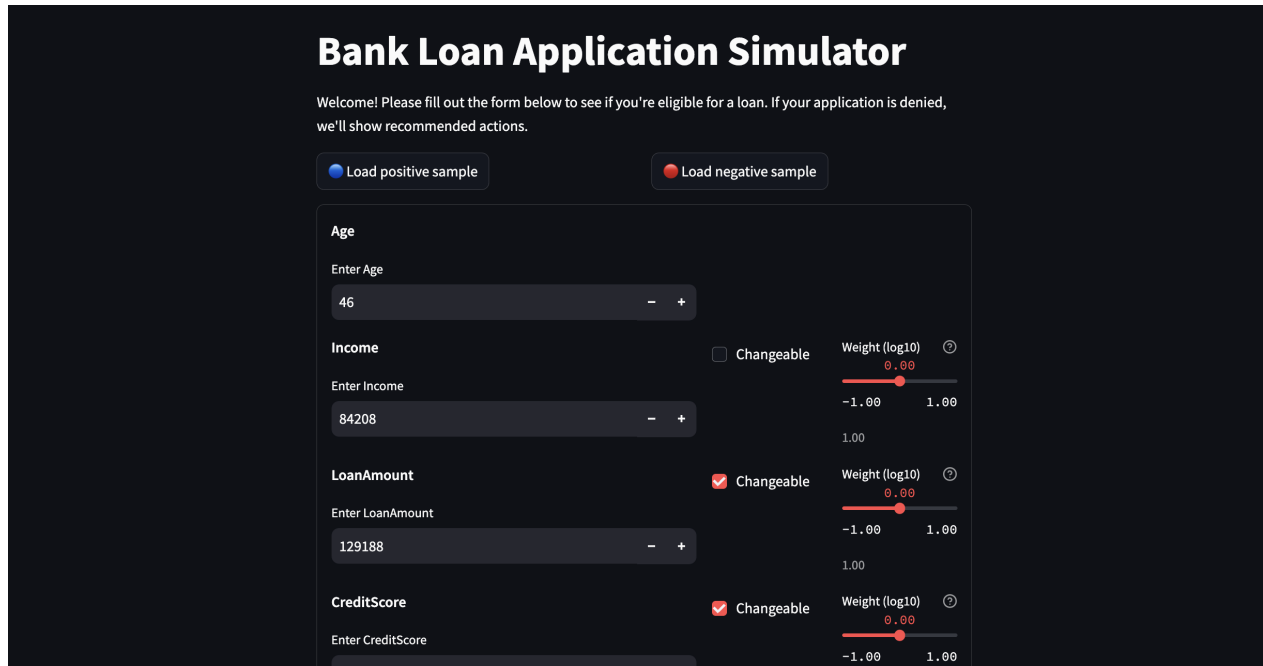
Spam. E-mail features are mostly continuous TF–IDF weights, so enforcing integrality has virtually no effect $(0.7968 \pm 0.1320 \ \& \ 0.8119 \pm 0.1326)$. The relatively low absolute figures indicate that slight textual changes (such as the addition or removal of a word) has very little effects on the counterfactual proposed.

Santander. Both variants achieve near-perfect stability $(\approx 0.995$ with deviations $< 0.01)$, confirming that the carefully engineered numeric features already impose strong local smoothness. Discrete regularisation has virtually no effect because all attributes are continuous.

Overall, the numbers confirm that the stability metric is dataset dependent. The little change in the stability for the Spam and Santander datasets and the big change in the Loan one suggests that the integer regularizer is much more unstable than the base counterfactual explainer, but overall it is fairly stable compared to other methods in the literature [3].

5 Website

To finally ground the algorithm in a real-world application, we have used the [14] dataset to create a simple interactive web application that allows users to input their data and obtain counterfactual explanations. The web application is built using `streamlit`, a python library that allows to create interactive web applications easily. This is only an example, the website is designed to by easily adaptable to any dataset which has been cleaned and preprocessed.

Figure 4: Example implementation of the algorithm in a web application

As shown in Figure 4, the user can input their data in the form, select the features they want to change (or that they *can* change), and the weights of those features. The weights are between -1 and 1, but this done like this for a smoother and more intuitive user experience. In reality we take the log_{10} of the value displayed. The minimum value is 0.1 and the maximum value is 10 as mentioned earlier, but this can be adjustable to any value.

Categorical fields are one-hot encoded and the user can select the value. Going back to Figure 4, we can see that the *changeable* toggle button allows the user to select which features they want to change and the *weights* slider allows the user to select the importance of each feature in the counterfactual explanation. These is not the case for every feature, and the entity can select which features they want the user to be able to change (for example, age is not a changeable value). The categorical features are deactivated as well, as we do not yet support the optimisation of these features in the algorithm. We have as well added two buttons to showcase the website and fill the values of the form with a sample instances from the dataset, representing both classes.
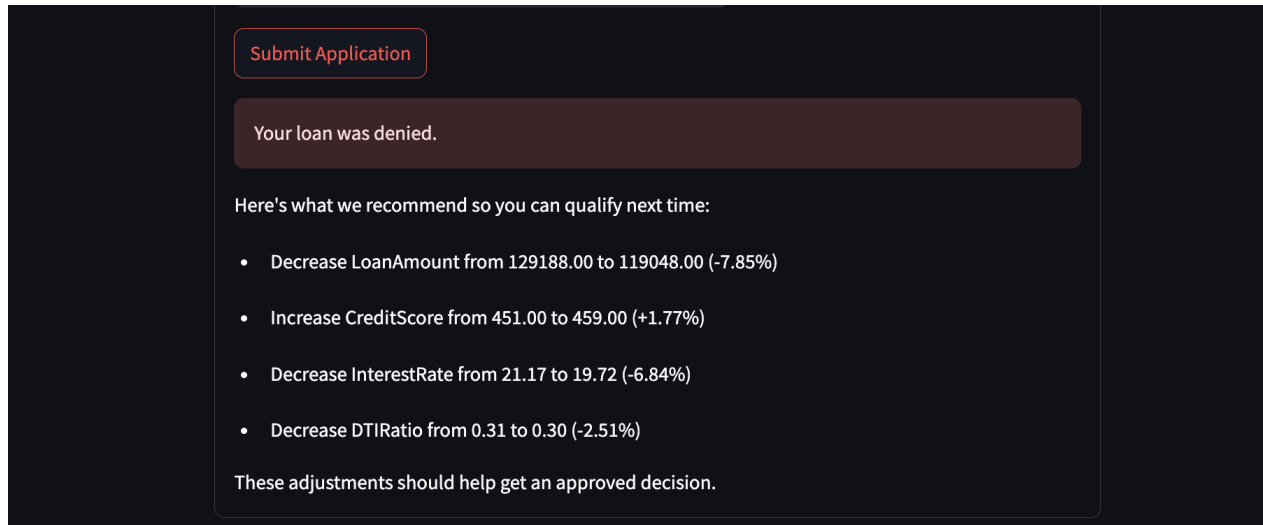
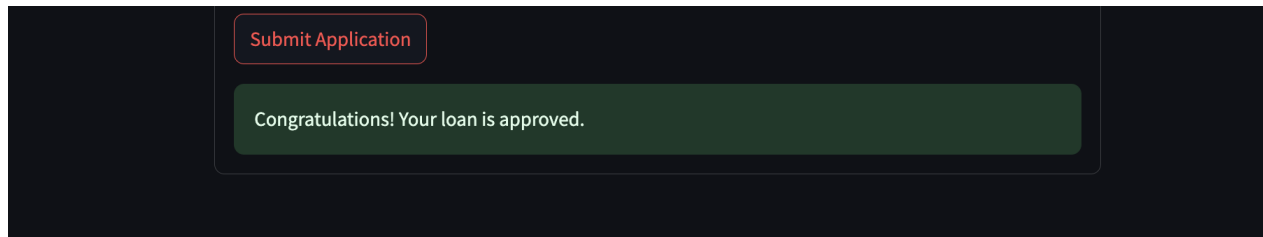Figure 5: Example output of a sample classified as denied

Figure 6: Example output of a sample classified as approved

When the user clicks on the *Submit Application* button, the instance is run through the model, and if the application is denied, the algorithm is run to find a counterfactual explanation. The output is shown in Figure 5, where it shows the output of the algorithm, which is the counterfactual explanation. Here, it shows the original value and the minimal changes that have to happen to each individual feature to change the classification output of the model. If we were to change each feature to the value shown, the model would classify the instance as approved and return what we see in Figure 6.

6   Conclusions and Future Work

This paper introduces the first counterfactual-explanation algorithm fully grounded in Newton optimisation. By using second derivatives, the method converges in only a handful of iterations to the *minimum* perturbation that flips the decision of any differentiable classifier (e.g. neural networks, logistic regressions).

We are also the first to incorporate an explicit *weight vector* in the cost function, encoding the real-world difficulty of changing a feature directly into the optimisation objective.

This simple yet powerful mechanism allows end-users to steer the explanation toward truly actionable recommendations.

The `PyTorch` code is written as a modular library: it works on any tabular dataset but can be tailored to organisation-specific rules with a few lines of configuration. Typical customisations could include the creation of granular continuous features, enforcing that a variable changes with a specific step size (e.g. salary changes only come in $5\,000$ or that the loan term can only be expressed in whole years). As we mentioned before, the entity can also select which features appear changeable and which are not (e.g. the interest rates are set by themselves, or they are not changeable by the user).

The combination of weighted controllability, flexible feature handling and Newton's rapid convergence yields counterfactuals that are valid, minimal, plausible, efficient, stable, and immediately actionable, while keeping runtime under a quarter of a second for datasets with up to 200 features. Because the optimiser is model-agnostic and the weighting scheme encodes domain knowledge directly, the approach is ready for deployment in highly regulated settings such as credit approval.

In the future we would like to extend the algorithm to ensure some kind of minimality with respect to the features, adding a regularizer that penalizes the number of features changed. We have already played around with this idea, but we still need to decide how do we manage the balance between similarity and minimality.

We would also like to explore the possibility of optimising the weights of the features to provide an initial value or guess instead of starting with all weights equal to 1. We would need to record data from the users to be able to do this, asking them which sets of weights have been more useful or that have yielded the more actionable counterfactuals. This would allow us to provide a more personalised experience for the user, as the algorithm would be able to learn from the user's preferences and adapt to them.

Finally, we would like to explore the possibility of extending the algorithm to work with categorical features, as we have not yet implemented this in the algorithm. This would allow us to provide counterfactual explanations for a wider range of datasets and use cases, making the algorithm more versatile and applicable to real-world scenarios. We have theorised how to do this with entropic regularizers, but we have not yet implemented it in the algorithm. The idea is to treat each one-hot group as a probability simplex, apply an entropic regulariser to keep the vector near the vertices, and then snap to the highest-probability value at the end, similar to the approach used for discrete features.

References

[1] S. Wachter, B. Mittelstadt, and C. Russell. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *SSRN Electronic Journal*, 2017.

[2] R. Guidotti. Counterfactual Explanations and How to Find Them: Literature Review and Benchmarking. *Data Mining and Knowledge Discovery*, 38:2770–2824, 2024.

[3] F. Bodria, F. Giannotti, R. Guidotti, F. Naretto, D. Pedreschi, and S. Rinzivillo. Benchmarking and survey of explanation methods for black box models. *Data Mining and Knowledge Discovery*, 37(5):1719–1778, 2023.

[4] B. Ustun, A. Spangher, and Y. Liu. Actionable Recourse in Linear Classification. *Proceedings of KDD '19*, 2019.

[5] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv preprint arXiv:1602.04938*, 2016.

[6] S. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. *arXiv preprint arXiv:1705.07874*, 2017.

[7] S. N. Cohen, D. Snow, and L. Szpruch. Black-Box Model Risk in Finance. *SSRN Electronic Journal*, 2021.

[8] N. Ghatasheh. Business analytics using random forest trees for credit risk prediction: a comparison study. *International Journal of Advanced Science and Technology*, 72:19–30, 2014.

[9] Anonymous. Point of View: Using Random Forest for credit risk models (Machine learning and Credit Risk: a suitable marriage?), 2019.

[10] J. Pizarroso, J. Portela, and A. Muñoz. NeuralSens: Sensitivity Analysis of Neural Networks. *Journal of Statistical Software*, 102(7):1–36, 2022.

[11] J. Fliege, L. M. G. Drummond, and B. F. Svaiter. Newton's Method for Multiobjective Optimization. *SIAM Journal on Optimization*, 20(2):602–626, 2009.

[12] Vapnyarskii, I. B. (2001). [1994] 'Lagrange multipliers'. Encyclopedia of Mathematics.

[13] J. M. Papakonstantinou. Historical Development of the BFGS Secant Method and its Characterization Properties. 2009.

[Keane et al., 2021] Keane, M. T., Kenny, E. M., Delaney, E., & Smyth, B. (2021). *If only we had better counterfactual explanations: Five key deficits to rectify in the evaluation of counterfactual XAI techniques.* Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI-21).

[14] H, M. Y. (2022). Loan default dataset. Kaggle. `https://www.kaggle.com/datasets/yasserh/loan-default-dataset`.

[15] Hopkins, M., Reeber, E., Forman, G., & Suermondt, J. (1999). Spambase [Dataset]. UCI Machine Learning Repository. . `https://doi.org/10.24432/C53G6X`.

[16]   Piedra, M., Dane, S., & Jimenez, S. (2019). Santander Customer Transaction Prediction [Competition]. Kaggle. `https://kaggle.com/competitions/santander-customer-transaction-prediction`.

[17] H. Robbins and S. Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[18] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[19] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000.

[20] A. Borodin. Determinantal point processes. *arXiv preprint arXiv:0911.1153*, 2009. `https://arxiv.org/abs/0911.1153`.

[21] Y. Ramon, D. Martens, F. Provost, and T. Evgeniou. A comparison of instance-level counterfactual explanation algorithms for behavioral and textual data: SEDC, LIME-C and SHAP-C. *Advances in Data Analysis and Classification*, 14(4):801–819, 2020.

[22] R. K. Mothilal, A. Sharma, and C. Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency (FAT* '20)*, pages 607–617, 2020.

[23] K. Mohammadi, A.-H. Karimi, G. Barthe, and I. Valera. Scaling Guarantees for Nearest Counterfactual Explanations. *arXiv preprint arXiv:2010.04965*, 2021.

[24] Nocedal, J., Wright, S. J. (2006). Numerical optimization. New York, NY: Springer. ISBN: 978-0-387-30303-1

[25] SciPy Development Team. *scipy.optimize.newton* (Version 1.15.3) [Computer-software documentation], 2025. SciPy. Retrieved June 4, 2025, from `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton.html`.