Algorithms for Generating Robust and Diverse Counterfactuals

Abbie Griffiths

8th May 2025

Module:	CM3203
Supervisor:	Nico Potyka
Moderator:	Hiroyuki Kido

School of Computer Science and Informatics at Cardiff University

Acknowledgements

Firstly, I would like express my gratitude and appreciation to my supervisor, Nico Potyka, for all of his time and help throughout this project. I would also like to thank Hiroyuki Kido, for taking the time to moderator my work.

Additionally, I would like to thank my family for all of their love and support throughout my degree.

Contents

List of Figures

List of Tables

1 Introduction

Machine learning models increasingly influence critical decisions in our lives, yet they often operate as "black-box" models, meaning that their decision making process remains unclear to end users. This dissertation aims to contribute to the field of Explainable Artificial Intelligence by developing an algorithm that provides robust and diverse counterfactual explanations.

Counterfactual explanations provide actionable insights by showing a user how input features can be modified to achieve a different outcome. For instance, in a loan application scenario, a counterfactual explanation might inform a rejected applicant: "Your loan would have been accepted if your income was 5% greater, or if your debt level was 10% lower." These counterfactual explanations represent an optimisation problem: finding the minimal changes needed to alter the model's output.

While several approaches that generate counterfactuals exist, current methods face similar limitations. The first being diversity, providing a user multiple alternative explanations, and the second being robustness, ensuring small changes in the input receive small changes in the output.

To address these challenges, a genetic algorithm has been implemented to generate diverse counterfactual explanations. By formulating the search for counterfactuals as an evolutionary optimisation problem, the algorithm efficiently explores the feature space to identify multiple viable counterfactuals.

The proposed genetic algorithm successfully generates counterfactual explanations. Through experimental evaluation across multiple datasets using established comparison metrics, the approach produces diverse counterfactual explanations while maintaining robustness to input perturbations, comparing well with existing methodologies.

2 Literature Review

2.1 Counterfactual Explanations Foundation

An original formulation by Wachter et al. [Wachter et al., 2017] recognised the need for explanations for "black-box" machine learning systems. They proposed three main areas that these explanations need to cover, these were:

- to help the user understand why a particular decision was reached,
- to provide authority to contest inauspicious decisions,
- and to understand what could be changed to receive a desired result in the future from the current model.

The core mathematical framework they proposed was an optimisation problem to generate the counterfactual. This meant that given an input x_i, the goal of achieving counterfactual x' minimises:

$$\arg\min_{x'}\max_{\lambda} \lambda(f_w(x') - y')^2 + d(x_i, x') \tag{1}$$

Where $f_w(x')$ is the model prediction for the counterfactual, w meaning the model weights, y' is the desired output, $d(x_i, x')$ is a distance function measuring the distance between the counterfactual x' and the original input x_i, and λ balances the diversity of the desired outcome due to the trade-off between reaching the outcome and making as small a change as possible.

In the context of financial applications, such as a loan application, the feedback they provided the user is along the lines of:

"You were denied a loan because your annual income was £30,000. If your income had been £45,000, you would have been offered a loan."

This counterfactual provides clear and useful feedback to the applicant. However, one of the challenges this model faced is that the framework does not account for the stability of explanations across similar inputs. This brings to light the issue of robustness and how small changes in the input can greatly change the output of the counterfactual. For the genetic algorithm discussed in this paper, the robustness of the explanations produced will be evaluated by testing the algorithm with similar inputs to ensure that the outputs are not too different.

While Wachter et al. established the foundation of counterfactual explanations, their approach has notable drawbacks. Their optimisation approach attempts to balance finding the "closest possible world" whilst ensuring feasibility through the use of the $\lambda(f_w(x') - y')^2$. However, this formulation creates stability issues as when λ approaches infinity, feasibility becomes the only focus and minimising the distance becomes disregarded.

The approach of using the genetic algorithm addresses these fundamental limitations by exploring a solution space more comprehensively and maintaining a diverse population of potential solutions across generations.

2.2 Key Technical Approaches

A similar implementation that also uses a genetic algorithm is the Adaptive Feature Weight Genetic Explanation (AFWGE) [AlJalaud and Hosny, 2024], which enhances the generation of counterfactual explanations by embedding feature weights directly into the solution. This approach consists of a "chromosome" that has two parts, the first part being the feature values and the second being the feature weights that indicates feature importance. These weights are initialised equally (1/n per feature) and then adaptively evolved during the search process alongside the feature values.

AFWGE uses a feature mutability constraint which classifies features as either mutable (like body weight, blood pressure, or salary) which can be changed or immutable (like sex, race, or age) which remain fixed across all generated chromosomes. Some features may also follow partial constraints; for example, age can increase but not decrease. Although this paper's approach shares AFWGE's evolutionary framework, it does not implement feature weights. This was originally done using the coefficients of importance from the machine learning models but the focus shifted to implementing other constraints.

AFWGE focuses on adapting feature weights within their algorithm, other approaches like DICE have a similar focus [Mothilal et al., 2020]. The DiCE algorithm is implemented to derive a counterfactual explanation, which is outputted in a table format that includes the relevant changes needed, as well as an explanation as to why it has been suggested. To ensure the feasibility of what the user can realistically change, they have added weights to the different features. For example, for a loan application, acquiring a higher education is harder to change than the number of hours a person works a week, so education would have a higher feature weight attached, similar to AFWGE.

The approach employed in this project shares DiCE's multi-step process but differs in implementation. Where DiCE uses gradient-based optimisation, this paper's method of a genetic algorithm uses an evolutionary exploration process to explore the solution space more thoroughly. This is particularly useful for discrete feature spaces, where gradient methods are less effective, but it also works well with continuous data too. Although DiCE achieves high diversity, in terms of robustness, this factor was not considered in the quantitative evaluation stage of the method. In addition, their gradient based optimisation approach limits applicability to differentiable models only. This excludes widely used non-differentiable models such as decision tree ensembles, such as Random Forests and XGBoost, which cannot be explained using methods that rely on gradients. The preposed method addresses this limitation by functioning as a model-agnostic method that only requires access to the model's outputs, disregarding its internal structure or differentiability.

The DiCE approach achieves diversity through gradient-based optimisation, whereas FACE (Feasible and Actionable Counterfactual Explanations) [Poyiadzi et al., 2020] takes a different direction by focusing primarily on the feasibility and actionability of explanations rather than their diversity.

FACE concentrates on ensuring the counterfactual explanations represent feasible and achievable paths of change for a user. They argue that current work mainly focuses on trying to find the smallest changes to achieve the required output, the “closest possible world” approach, and that this is not enough to ensure the counterfactual is feasible and actionable to an end user. Therefore, their algorithm accounts for both the nature of the counterfactual and quantifies the actionability of the explanation. The algorithm accomplishes this by using density-weighted metrics to find the shortest paths through high-density regions of the data space. This approach prevents suggesting counterfactuals that lie in low-density regions (which might be unrealistic outliers) or that require unrealistic transformations. However, this approach is computationally heavy and is difficult to deploy in real-time. As well, as it estimates the data’s density, it can become increasingly more computationally challenging.

The implementation of the proposed genetic algorithm in this paper incorporates FACE’s insight on the importance of feasibility, but with a different approach. Instead of relying on complex density estimation, it utilises the population-based nature of genetic algorithms to explore the solution space naturally. The selection function favours solutions that follow the common data distribution, as extreme outliers typically perform poorly against the objective function. Additionally, by returning multiple counterfactuals, the approach provides users with different options that represent diverse potential changes. Domain constraints are incorporated into the genetic operators to promote feasibility by:

- Restricting mutations to stay within the observed variable domains for each feature
- Using different mutation strategies for different continuous and categorical variables that respect their natural constraints
- Clipping numerical values to ensure minimum and maximum bounds are considered
- Ensuring categorical variables can only take the values observed in the training data

These mechanisms help guide the search toward solutions that respect the data’s natural constraints, although they do not guarantee strict feasibility for all domain specific constraints that might exist.

FACE addresses the feasibility challenge counterfactuals have while other research has explored mathematical approaches such as a mixed-integer programming approach, that finds diverse and coherent counterfactuals [Russell, 2019]. Here, a set of constraints called “mixed polytope” are used to find coherent explanations.

They provide coherent counterfactual explanations in a basic format consisting of:

```
You got score ____.
 One way you could have got score  ____ is if:
  ____ took value  ____ rather than  ____
Another way you could have got score  ____ is if:
  ____ had taken value  ____ rather than  ____
```

Figure 1: Mixed Integer Programming Output Example

They work with mixed data sets of contiguous data and discrete variables and have achieved a method for generating counterfactual explanations that are diverse. Overall, they conclude that more reliable methods to produce coherent and diverse counterfactuals are still needed, which is why the motivation for this dissertation is justifiable.

While much of the original work on counterfactual explanations uses tabular data, recent research has expanded into more complex structures such as graphs. A paper addresses Graph Counterfactual Explain-ability (GCE) which explains the predictions of Graph Neural Networks by generating alternative graphs that would lead to different predictions [Prado-Romero et al., 2024]. They also come across the robustness issue and found that in GCE methods, minor changes to the input can lead to vastly different explanations. To overcome this they also return multiple diverse counterfactual explanations to improve the robustness using MACCS (a method that generates molecular counterfactuals by exploring chemical space without pre-trained models [Prado-Romero et al., 2024]le). This shows that the diversity and robustness challenge extends to beyond different data structures, suggesting that they are inherent to counterfactuals.

These existing approaches highlight the common issue between computational efficiency, explanation feasibility, robustness, and diversity. Genetic algorithms are a promising middle ground as they work with multiple features well (such as the continuous and categorical features this paper will work with), explore a solution space effectively, and can generate diverse solutions in a single run. The genetic algorithm implementation focuses on these insights, specifically on robustness through diversity, combining the evolutionary search capabilities of genetic algorithms with explicit diversity constraints.

2.3 Robustness Challenges

As identified in the methods above, a key challenge researchers have identified when applying counterfactual explanations to machine learning models is their potential lack of robustness across similar inputs. This robustness issue stems from the algorithm's sensitivity to model parameters, such as in local search methods like hill-climbing, it can be highly sensitive to input perturbations. As Leofante and Potyka [Leofante and Potyka, 2024] demonstrate, minor changes in the input can cause major changes in the explanation generated. This creates potential confusion for users, as two similar inputs could receive completely different counterfactual explanations. For example, if two people with similar profiles apply for a loan, they might receive substantially different feedback from the counterfactuals about how to improve their applications. This inconsistency may lead users to question the credibility and reliability of the model creating the counterfactual explanations.

Russell [Russell, 2019], who's approach of mixed-integer programming mentioned earlier, recommends that for nonlinear functions, to achieve robustness, an approach would be k-nearest neighbour or random forests [Russell, 2019]. However, implementing something like k-nearest neighbour would not sufficiently address the robustness challenges counterfactuals face as the approach would still operate within the same solution space as the original problem, meaning they would find a similar solution to the original input feature space. Additional limitations to these suggestions include the computational complexity which can increase significantly when counterfactual methods try to incorporate multiple approaches to increase robustness, making them impractical for real-life use cases.

The suggested genetic algorithm addresses these robustness challenges though multiple mechanisms. Firstly, the selection process maintains diversity in the population throughout the evolution. As well, by returning multiple diverse counterfactuals instead of just one "best" solution, diversity is increased which promotes robustness. This approach aligns with Leofante and Potyka's [Leofante and Potyka, 2024] research as this provides the user with more stable explanations.

2.4 Diversity Methods

Diversity is a key factor to consider for producing useful counterfactual explanations and can help improve the robustness of the counterfactuals. The explanations provide a user with comprehensive understanding of their available options and by representing multiple counterfactuals, they allow a user to clearly see the possible options they can select that aligns best with their individual goals.

The requirement for diversity in the explanations is achieved through multi-objective optimisation [Dandl et al., 2020]. This is needed due to the competing factors of proximity to the input and diversity of the output. Therefore, to balance the two objectives, a common work around for this issue is that multiple counterfactuals are generated where:

- Each counterfactual maintains a reasonably small distance from the input
- But that these counterfactuals have a large distance between each other to ensure diversity

The genetic algorithm presented in this research will try to implement these two factors stated above, keeping the changes small between input and output, but the options varied for the user, and consequently having a bigger distance between the counterfactuals outputted.

2.5 Literature Overview

Table 1: Literature Overview of Different Counterfactual Approaches

Approach	**Computational Efficiency**	**Robustness**	**Diversity**	**Multiple Explanations**	**Optimality Guarantees**
Wachter et al.	Moderate	Low	Limited	Supported but not optimised	Moderate
AFWGE (GA)	Medium	Medium-High	High	Yes	Low
DiCE	Medium	Medium	High	Yes	Low
FACE	Medium	Medium-High	Medium-Low	No	Medium
Mixed-Integer	Low	Medium	Medium	Yes	High
Robustness (Leofante & Potyka)	High	High	High	Yes	Medium
Suggested Approach	Medium-high	Medium	High	Yes	Medium

The justification for the rating given to each method in Table 1 is explained below:

Wachter et al.

- Computational Efficiency: Moderate - uses gradient based optimisation which works well for different differentiable models but may struggle with non-differentiable ones
- Robustness: Low - does not explicitly consider robustness
- Diversity: Limited - focuses on finding the closest counterfactual rather than diverse alternatives
- Multiple Explanations: Supported but not optimised - can generate multiple counterfactuals by rerunning the algorithm
- Optimality Guarantees: Moderate - provides local optima but not global optimality

AFWGE

- Computational Efficiency: Medium - genetic algorithms typically require multiple generations
- Robustness: Medium-High - adaptive feature weights help improve the robustness but no formal guarantees
- Diversity: High - genetic algorithms naturally maintain a diverse population of solutions
- Multiple Explanations: Yes - can return multiple solutions from the population
- Optimality Guarantees: Low - no formal guarantees of optimality

DiCE

- Computational Efficiency: Medium - requires gradient based optimisation with additional computations for diversity
- Robustness: Medium - does not specifically address robustness
- Diversity: High - explicitly optimises for diversity for counterfactuals
- Multiple Explanations: Yes - provides multiple counterfactuals
- Optimality Guarantees: Low - provides local optima but not global

FACE

- Computational Efficiency: Medium - requires graph based computations
- Robustness: Medium-High - consideration of data distribution improves robustness
- Diversity: Medium-Low - primarily designed for feasibility rather than diversity
- Multiple Explanations: No - primarily focuses on finding one feasible path
- Optimality Guarantees: Medium - guarantees shortest path but not necessarily the global optimal one

Mixed-Integer Programming

- Computational Efficiency: Low - a computationally intensive approach
- Robustness: Medium - robustness constraints used
- Diversity: Medium - can guarantee diversity through the constraints used
- Multiple Explanations: Yes - it can iteratively solve for multiple solutions
- Optimality Guarantees: High - provides global optima within the formulated constraints

Robustness Approach (Leofante & Potyka)

- Computational Efficiency: High - uses efficient filtering mechanisms
- Robustness: High - explicitly designed to maximise robustness
- Diversity: High - uses diversity as a way to achieve robustness
- Multiple Explanations: Yes - returns multiple diverse counterfactuals
- Optimality Guarantees: Medium - uses approximation strategies

While this comparison focuses on tabular approaches most relevant to this paper, it's worth noting that Graph Counterfactual Explain-ability [Prado-Romero et al., 2024] mentioned earlier also faces challenges in robustness and similarly employ diversity strategies to address them.

This literature review reveals that while significant progress has been made to understand these black-box models, challenges remain balancing diversity, robustness, and computational feasibility, and efficiency. The algorithm proposed addresses these challenges by leveraging the genetic algorithm to maintain diverse solution populations. By utilising the natural evolutionary process of genetic algorithms, the approach efficiently generates and preserves a diverse set of counterfactual explanations, taking advantage of the algorithm's computational strengths in population-based optimisation. By building on these insights of different approaches, the implementation aims to support the state of counterfactual explanations particularly in areas of robustness and diversity.

3 Background

3.1 Counterfactual Explanations and Genetic Algorithms

Counterfactual explanations give users the ability to better understand machine learning model decisions. A counterfactual explanation can be defined formally as the following:

Given a classifier, $f : \mathcal{X} \rightarrow \mathcal{Y}$ [Verma et al., 2020] and an input instance $x \in \mathcal{X}$ with classification $f(x) = y$, a counterfactual c is the solution to the optimisation problem:

$$\min_{c \in \mathcal{X}} \quad d(x, c) \tag{2}$$

$$\text{subject to} \quad f(c) \neq f(x) \tag{3}$$

Where:

- $d(x, c)$ is the distance metric measuring the difference between x and c
- and the constraint $f(c) \neq f(x)$ requires that the counterfactual c has a different prediction from the original input x (equation 3)

To solve such an optimisation problem, an efficient search strategy that can explore possible counterfactuals while satisfying the distance constraint is needed. Genetic Algorithms are well suited for this problem as they are stochastic search algorithms, meaning that they rely on randomness to explore the search space and find a solution [Shapiro, 1999].

Genetic algorithms apply what Charles Darwin discovered in On The Origin of Species [Darwin, 1859] about natural selection and survival of the fittest, which is the nature of this optimisation technique. John Holland was a pioneer in the field of genetic algorithms, with his book Adaptation in Natural and Artificial Systems [Holland, 1992] establishing the foundational framework for genetic algorithms.

Genetic algorithms are a population-based search method that keeps a population of solutions and modifies them in parallel [Bajpai and Kumar, 2010]. This differs from other optimisation methods, such as simulated annealing, which iteratively moves through the configuration space one solution at a time [Rutenbar, 1989]. A population is made up of chromosomes, which are a string of "genes" representing a possible solution to the problem. Genetic operators are then used to transform the current population through selection, crossover, mutation, and replacement methods to find a solution. A fitness function that will be maximised (most commonly) will be applied to distinguish between fit or unfit chromosomes depending on the problem statement. Pseudocode of how a genetic algorithm uses these operators can be seen in Algorithm 1.

Algorithm 1 Genetic Algorithm for Counterfactual Generation

```
Initialise population P of size N
for each generation g from 1 to g do
    Evaluate fitness of all chromosomes in P
    Select parents using selection operator
    Create offspring through crossover with probability p_c
    Apply mutation to offspring with probability p_m
    Replace population using replacement operator
    if termination condition is met then
        break
    end if
end for
Return best counterfactuals found
```

As shown in Algorithm 1, the genetic algorithm for counterfactual generation begins by initialising a population, which is usually random. For each generation, chromosomes are evaluated using a fitness function that rewards chromosomes respecting the problem's constraint, which for counterfactual generation, is the constraint shown in equation 3, while minimising distance from the original input (equation 2). Parents are selected from the population for reproduction through a selection process that is based on their fitness. The parents then undergo recombination (crossover) that combines genetic martial from two parents to create offspring, with probability p_c. The offspring are then mutated with probability p_m, introducing random changes to genes in individuals to maintain diversity. The next generation is formed through replacement, which preserves the best solutions, ensuring that good solutions are not lost. This process iterates for g generations, or until a termination condition is met, gradually improving the population until satisfactory counterfactuals are found.

The effectiveness of genetic algorithms for counterfactual generation stems from several key advantages, one of them being the population-based nature of theirs naturally promotes diversity in the generated counterfactuals, which is crucial for providing users with multiple meaningful alternative explanations. They are also effective in finding good solutions for complex optimisation problems, with their random nature helping to avoid local optima [Bajpai and Kumar, 2010]. They also do not need gradient information, making them suitable for non-differentiable or discontinuous objective functions. Adaptability is another advantage to genetic algorithms as modification to the chromosome representation and genetic operators can be made to suit an individual problem.

However, parameter tuning is a limitation genetic algorithms face, as the performance is reliant on the parameter settings such as the population size, mutation probability etc. Finding these optimal parameters can be challenging [Karafotias et al., 2015]. There is also no guarantee that the genetic algorithm will find the global optima [Wehrens et al., 1999]. Another challenge is implementing appropriate chromosome representation for solution encoding [Rothlauf, 2006].

For this optimisation problem to find counterfactual explanations, genetic algorithms seem an appropriate choice. They offer advantages in their ability to explore diverse solutions simultaneously, aligning well with the goal of generating multiple diverse counterfactuals. The population based approach naturally executes a diverse set of solutions which can mean alternative explanations to users. Additionally, the adaptability of genetic algorithms means that the objective, fitness, and distance functions can be flexible for implementation. Other researchers have also used genetic algorithms for generating counterfactuals [AlJalaud and Hosny, 2024], and while our approaches differ, the fact that genetic algorithms have been used to solve this problem adds to the appropriateness of this approach.

4 Methodology and Algorithm Design

A genetic algorithm has been used for this problem as their adaptive nature makes them good for optimisation problems [Beasley et al., 1993]. Having studied them in a previous module and finding their evolutionary approach interesting, applying them to this optimisation problem seemed like a good opportunity.

This section consists of an overview for the general framework for counterfactual generation, followed by a detailed explanation of each genetic algorithm component. For each genetic algorithm operator, the theoretical justification is described, alongside its specific implementation details. The section concludes with a full overview of the genetic algorithm, integrating all the previously explained components.

4.1 Framework Overview

The proposed counterfactual explanation framework operates as follows:

Inputs to the framework:

- A trained supervised classification model (more information on the chosen model can be found in the appendix)
- A reference input (the instance to be explained)

- Framework parameters (population size, generations, mutation rate, etc.)

Framework process:

Apply the genetic algorithm to generate counterfactuals that:

- Attain a different output compared to the reference input
- Minimise the distance from the counterfactual to the reference input
- Ensuring diversity and robustness in the outputted counterfactuals

Output:

- A set of k diverse counterfactual explanations

4.2 Genetic Algorithm Components

This section provides a detailed explanation of the components used for this genetic algorithm.

4.2.1 Chromosome Representation

For counterfactual generation, the chromosome must represent a potential counterfactual instance. In this implementation, chromosomes directly represent feature vectors that match the input space of the classifier, ensuring that generated counterfactuals are valid inputs to the model. This representation enables straightforward implementation of distance metrics and other operators too. Therefore, for this algorithm:

- Each chromosome represents a potential counterfactual input vector
- A chromosome's feature is encoded as a gene

4.2.2 Population Initialisation

The initial population in this work is generated by randomly selecting instances from the dataset, specifically using the feature columns (excluding the target variable). This is implemented by first loading the dataset and for each member of the population, a random instance is selected. This approach ensures diversity in the starting population by sampling from the entire feature space of the dataset, allowing the genetic algorithm to explore different regions when searching for counterfactuals.

Maaranen et al. [Maaranen et al., 2007] investigated several initialisation methods including pseudo-random number generation (where values are randomly generated within intervals), quasi-random sequences like the Niederreiter generator (which aim for better coverage of the search space), and spatial point processes (which maintain minimum distances between points). While their approach generates synthetic points throughout the entire feature space, the method employed in this paper samples from the existing data distribution, which may provide an advantage by starting with naturally occurring feature combinations.

Despite these differences, Maaranen's findings support this random population initialisation approach in two ways. First, they demonstrated that simple random methods performed just as well as the more complex initialisation strategies, especially in high-dimensional spaces. Second, they found that initialisation methods with good genetic diversity showed faster convergence in early generations.

The random sampling-based method employed in this paper ensures diversity while maintaining feasibility, as all initial solutions represent real instances from the problem domain. This provides a natural starting point for the evolutionary search process which avoids the generation of potentially invalid feature combinations. Consequently, in this algorithm, this initial population is achieved by:

- Randomly choosing an instance (chromosome) from the dataset
- Repeating this process population_size times to form the complete initial population

4.2.3 Constraint

By definition, counterfactual explanations must obtain a contrasting output, i.e., in terms of the mushroom dataset (one of the datasets evaluated in this paper), if the original chromosome's output is edible, then the counterfactual's output must be poisonous. This constraint is fundamental to counterfactual explanations as it ensures the generated solutions actually change the model's prediction.

For this genetic algorithm, this constraint is enforced through the output constraint function, which compares the predictions of potential counterfactuals with the original instance's prediction. The function returns True when the predictions differ and False when they are the same. This binary outcome is then used in the fitness calculation, where chromosomes that fail to change the prediction (returning False) receive a fitness score of 0, effectively removing them from consideration in future generations. This ensures that only valid counterfactuals that achieve the desired prediction change can survive and reproduce in the population.

4.2.4 Distance Calculation

A distance metric is used as a diversity optimisation criterion to improve the diversity of the counterfactuals. The distance is calculated based on the proximity to the original input the counterfactual is, where higher scores are assigned to solutions requiring fewer changes, prompting the genetic algorithm to find counterfactuals that are both valid and a minimal distant from the original input. The distance calculation used was the Manhattan distance, shown below in equation 4:

$$d(x, x') = \sum_{i=1}^{n} |x_i - x'_i| \tag{4}$$

Where:

- n is the total number of features
- x_i refers to the value of the i-th feature of the original input and x'_i refers to the value of the i-th feature of the counterfactual

4.2.5 Fitness Function

A fitness function is an important feature in a genetic algorithm and for this problem, it was implemented to calculate the fitness score of a chromosome based off of two factors.

The first being the primary constraint (explained in Section 4.2.3) to assess whether the candidate counterfactual produces the opposite classification outcome compared to the original instance. This forms a hard constraint in the fitness evaluation where chromosomes that fail this constraint receive a fitness score of 0, effectively removing them from consideration in future generations.

For chromosomes that satisfy this constraint, a second factor evaluates their quality using the Manhattan distance metric described in Section 4.2.4. The distance is incorporated into an objective function (to be maximised):

$$\text{objective} = \frac{1}{\text{distances} + \epsilon} \tag{5}$$

Where *distances* is the distance that represents the difference between the counterfactual and the reference input and ϵ is a very small positive number (1e-10 in the implementation) to prevent division by zero. Inverting the objective rewards solutions with minimal changes. This has been designed to ensure that small changes in the distance between the initial reference input and the outputted counterfactual results in meaningful changes to the objective function. This occurs due to the inverse relationship with distance that the objective function creates.

Due to the adaptive nature of genetic algorithms, the fitness function can be adapted personally for the task at hand and this approach seemed an appropriate way to evaluate a chromosome in this problem.

4.2.6 Selection Operator

Two of the most common selection methods in genetic algorithms are:

1. Roulette Wheel: In this method, a virtual wheel is divided into N sectors where each individual is assigned a slice proportional to its fitness value relative to the total population fitness. The larger the fitness of an individual, the more likely it is to be selected when the wheel is spun [Kaya, 2011].
   - Advantage:
     - Proportionally represents fitness values, giving higher probability to fitter individuals while still maintaining diversity [Mitchell, 1998]
   - Disadvantage:
     - Chromosomes with high fitnesses may dominate the selection process, potentially leading to premature convergence. If a few individuals have significantly higher fitness values than others, they will receive disproportionately large slices of the wheel, reducing genetic diversity [Brownlee, 2024].
2. Tournament: This randomly selects chromosomes and compares their fitness scores, keeping those with the highest fitness, which are then added to the population [Kaya, 2011].
   - Advantage:
     - Preserves genetic diversity and prevents premature convergence by reducing selection pressure, allowing less dominant but potentially valuable solutions to survive [Miller et al., 1995]
     - They perform better in terms of convergence rate and time complexity than other methods, such as the roulette wheel [Shukla et al., 2015]
   - Disadvantage:
     - Parameter sensitivity to selection pressure, the tournament size directly affects selection intensity. Too small a tournament size leads to slow convergence (less selective pressure), while too large a size causes rapid convergence toward local optima [Miller et al., 1995]

Consequently, the tournament selection process was found to be a good choice for a selection method for this algorithm. A small selection size of three has been chosen to promote diversity and give chromosomes with lower fitnesses a chance [Miller et al., 1995].

The Tournament selection for this algorithm works by:

1. Three randomly selected chromosomes are chosen from the population
2. The chromosome with the highest fitness is chosen for reproduction

Algorithm 2 Tournament Selection

```
function TOURNAMENT_SELECTION(population, fitness_scores, population_size)
    selected_counterfactuals ← [ ]
    for i ← 1 to population_size do
        tournament_idx ← random_sample(range(len(population)), 3)
        tournament_fitness ← [fitness_scores[i] for i in tournament_idx]
        winner ← tournament_idx[index_of_max(tournament_fitness)]
        selected_counterfactuals.append(population[winner])
    end for
    return selected_counterfactuals
end function
```

While tournament selection is effective for maintaining population diversity during evolution, an additional selection method is employed to this specific genetic algorithm to ensure the final set of counterfactuals are meaningfully different from each other. This diversity selection method acts as a post-processing step in the genetic algorithm, working as follows:

1. Sorting all chromosomes by fitness and selecting the best individual first
2. Iteratively adding chromosomes that are sufficiently different from the previously selected ones
3. Uses the distance function to calculate diversity (Section 4.2.4)
4. If not enough diverse solutions are found, it returns the best remaining chromosomes with a message that the selection process could not find any better

Algorithm 3 Diversity Selection

```
function DIVERSITY_SELECTION(population, fitness_scores, k)
    if population size ≤ k then
        return population
    end if
    Sort population by fitness
    selected ← [best individual]
    for each individual in population do
        if selected size = k then
            break
        end if
        diverse ← True
        for each solution in selected do
            if too similar to solution then
                diverse ← False
                break
            end if
        end for
        if diverse then
            Add individual to selected
        end if
    end for
    if selected size < k then
        Add best remaining individuals until selected size = k
    end if
    return selected
end function
```

4.2.7 Recombination Operator

Zainuddin et al. [Zainuddin et al., 2020] compared 13 different crossover methods for different optimisation problems. They concluded that for binary and real problem representation, multipoint crossover was an effective method of choice. Out of the 13 methods, it was the only one that worked with both binary and real data representation and therefore was the chosen crossover method in this genetic algorithm as the crossover method works well with the two types of data used in this paper.

The multiple point crossover operator works in two stages. First, a crossover rate determines whether recombination occurs between the selected parents. If recombination is triggered, the operator then selects multiple random points along the chromosome length to create crossover segments. These segments are alternately taken from each parent to create two new offspring. The crossover process used in this algorithm:

- Uses a crossover rate to determine whether recombination occurs
- If recombination is triggered, selects multiple random points to split the parents
- Creates two diverse children by alternating segments from each parent

Algorithm 4 Multi-Point Crossover

```
procedure MULTI_POINT_CROSSOVER(parent1, parent2, crossover_rate)
    Decide if crossover should happen based on crossover_rate
    if no crossover then
        return copies of parent1 and parent2
    end if
    Set number of crossover points (minimum of 2 and length-1)
    Choose random points to split the parents
    Sort these points in order
    Initialise children as copies of parents
    Create list of segments including start and end points
    Set current parents to original parents
    for each segment between points do
        if this is an odd-numbered segment then
            Swap current parent references
        end if
        Copy segment from current parents to children
    end for
    return both children
end procedure
```

4.2.8 Mutation Operator

For binary features, the mutation operator implements a simple bit-flip mechanism. Each gene in the chromosome, representing a binary feature, has a probability of being flipped - changing from 0 to 1 or vice versa. This probability is determined by the mutation rate parameter, which controls how frequently mutations occur. For example, in a chromosome representing categorical features encoded as binary values, a mutation might change a feature from "present" (1) to "absent" (0) or vice versa. This straightforward approach is particularly efficient for categorical data as it directly manipulates the binary representation while maintaining the validity of the feature values [Kala, 2024].

For continuous features, the implementation uses a mutation operator that adds random noise to the feature values. The mutation process is implemented as follows:

- For each continuous feature, a value is drawn from a normal distribution with mean 0 and scale parameter equal to 50% of the feature's standard deviation
- This value is added to the current feature value
- The result is clipped to ensure it remains within the feature's valid range (minimum to maximum observed values)
- For the special case of age, an exponential distribution is used instead, ensuring only positive changes for logical reasons

The choice of using 50% of the feature's standard deviation as the scale parameter ensures that mutations are proportional to the natural variance in the data, while the mean of zero means mutations are equally likely to increase or decrease the value. This approach allows for meaningful exploration of the feature space while maintaining the validity of the generated values.

Algorithm 5 Random Mutation

```
procedure RANDOM_MUTATION(chromosomes, variable_names, variable_domains, mutation_rate)
    mutated ← []
    for each chromosome in chromosomes do
        Create copy of chromosome
        for each gene in chromosome do
            if random chance < mutation_rate then
                if feature is numeric then
                    if feature is categorical numeric (small range of integers) then
                        Randomly select different valid value
                    else if feature is age then
                        Add exponential random change (positive only)
                        Ensure age only increases
                    else
                        Add normal random noise scaled by feature's standard deviation
                    end if
                    Keep value within valid range
                else
                    Randomly select different valid categorical value
                    Exclude "?" and NaN values
                end if
            end if
        end for
        Add mutated chromosome to new population
    end for
    return mutated
end procedure
```

4.2.9 Replacement Operator

Common replacement strategies are:

1. Elitist: where very fit solutions are kept from both the current population and mating pool and used in the next generation [Du et al., 2018]

2. Generational Replacement (Delete all process): where the current population gets replaced by the new population from the mating pool [Vavak and Fogarty, 1996]

3. Steady state: where N chromosomes become replaced from the current population with N new members from the mating pool [Vavak and Fogarty, 1996]

Elitist is chosen for this genetic algorithm as it is good practice to keep the top preforming chromosomes, as well, it is a method to avoid premature convergence [Shapiro, 1999]. Elitism also works well with the tournament selection process already implemented. Keeping the best preforming chromosomes and ensuring that they do not get lost during the selection process improves efficiency for finding top chromosomes. This algorithm applies the elitist replacement operator by:

1. Combining the current population and offspring

2. Getting the top fitness preforming chromosomes

Algorithm 6 Elitist Replacement

```
function ELITIST_REPLACEMENT(population, offspring, fitness_func, population_size)
    Combine current population and offspring
    Calculate fitness scores for all individuals
    Sort individuals by fitness in descending order
    Select top population_size individuals
    return selected individuals
end function
```

4.2.10 Diversity Optimisation Function

While the genetic algorithm naturally promotes diversity through its evolutionary process, an additional diversity function was implemented to ensure that the final set of counterfactuals provides meaningfully different explanations to users. This is crucial because similar counterfactuals would offer redundant suggestions, limiting their practical utility. The diversity function uses the distance metric (explained in Section 4.2.4) to enforce a minimum difference between counterfactuals, ensuring each one represents a distinct alternative scenario.

The function operates by:

- Comparing each new chromosome against the most recently found counterfactuals (up to the last 10)
- Calculating the Manhattan distance between chromosomes using the pre-defined distance function
- Rejecting chromosomes that are too similar (distance less than min_distance_threshold)

This approach ensures that users receive multiple distinct alternative scenarios, with each counterfactual representing a meaningfully different way to achieve the desired outcome.

Algorithm 7 Diversity Check

```
function DIVERSITY(chromosome, existing_counterfactuals, min_distance_threshold = 0.61)
    if existing_counterfactuals is empty then
        return true
    end if
    if len(existing_counterfactuals) > 10 then
        recent_counterfactuals ← last 10 elements of existing_counterfactuals
    else
        recent_counterfactuals ← existing_counterfactuals
    end if
    distances ← distance_criterion(recent_counterfactuals)
    return all distances ≥ min_distance_threshold
end function
```

4.3 Complete Genetic Algorithm Methodology

Having described the individual components, the genetic algorithm combines these functions into a cohesive process for generating counterfactual explanations. The algorithm proceeds as follows:

- Initialising a random population from the dataset
- Evaluates the fitness using vectorised operations for efficiency
- Employs the evolution processes where the Selection, Crossover, Mutation, and Replacement functions are called
- Diversity fucntion is called to ensure the outputted counterfactuals offer a user different enough options
- Early stopping is used if there have been no improvements after 15 generations
- An additional diversity selection method is used which calls the diversity selection function to select diverse counterfactuals from the current valid pool of counterfactuals
- Returns the output of 4 counterfactual explanations

Algorithm 8 Genetic Algorithm for Diverse Counterfactual Generation

```
function GENETIC_ALGORITHM
    Create initial population
    Initialise empty lists for storing valid solutions
    Cache reference input encodings and predictions
    Initialise stagnation counter and best fitness
    for each generation do
        Batch calculate fitness for current population
        for each individual in population do
            if solution is valid and diverse then
                Add to collection of good solutions
            end if
        end for
        Check for early stopping:
        if no improvement for 15 generations then
            break
        end if
        Select parents using tournament selection
        Create offspring through crossover
        Mutate offspring
        Create new population using elitism
    end for
    if more than 4 solutions found then
        Take top 50% by fitness
        Use diversity selection to get diverse solutions
    else
        Keep all solutions found
    end if
    return best solutions and their fitness scores
end function
```

5 Implementation

This section explains the implementation of the genetic algorithm, detailing the challenges encountered and the solutions developed to overcome them as well as demonstrating counterfactual output examples. Additionally, instructions on how to execute the algorithm are included.

5.1 Challenges and Solutions

Some unforeseen changes had to be implemented when working on the construction of the algorithm. This subsection will explain in more detail the obstacles faced and the solutions used.

5.1.1 Handling Mixed Data Types

The initial implementation worked well with categorical data but not with continuous data, this was resolved by introducing the changes:

- Creating type-specific checks in operators to separate how continuous and categorical features are handled
- Adapting the distance calculation to work efficiently for mixed data-types by implementing the Manhattan distance. Originally it was only counting the distance if there was a difference in the chromosome for binary use, such as (0,1,1,1) and (0,1,0,1) would have a distance of one, however this would no longer work with continuous data so the Manhattan distance measure was used.

5.1.2 Diversity Promotion Mechanisms

As diversity is one of the most important factors for achieving good counterfactuals, and robustness, two key diversity mechanisms were implemented to improve the diversity of the explanations.

The first being the diversity selection function (explained in Section 4.2.6). The tournament selection was still used to initialise the population during evolution, however, the diversity function was applied in the main genetic_algorithm function to select diverse solutions within the counterfactuals.

The diversity selection function implements a clustering-based approach to select a diverse subset of counterfactuals from the final population, this specifically ensures the final set of explanations are meaningfully different from each other. This approach ensures that each selected counterfactual has a high fitness, is sufficiently different from other selected solutions, and provides unique insights to the user.

The second being a diversity optimisation function (introduced in Section 4.2.10) that implements a minimum threshold for differences between counterfactuals to ensure they provide meaningful explanations. This addresses the issue where small numerical changes (like minor decimal variations in age or income) were being treated as significant differences, resulting in explanations that were technically different but practically similar.

This ensures that each counterfactual represents a meaningfully different scenario, making the explanations more actionable and valuable to users. The threshold is chosen to balance between generating diverse explanations and maintaining computational efficiency. The function is integrated into the genetic algorithm's main loop, filtering out counterfactuals that do not provide sufficiently distinct insights while keeping those that offer meaningful alternative scenarios.

5.1.3 Runtime Optimisation

The initial implementation had very long runtimes, this was addressed by:

- The genetic algorithm function implements an early stopping mechanism to improve computational efficiency. This mechanism tracks fitness stagnation using a counter that increments when no improvement in the best fitness score is observed between generations. When the stagnation counter reaches a defined limit, the algorithm terminates and returns the current best solution.
- Replacing loops with numpy's vectorised operations in the fitness function, as shown in Figure 2.

```python
def fitness(self, population: List[np.ndarray])-> List[float]:
    fitness_scores = []

    for chromosome in population:
        objective_result = self.objective_function(chromosome)
        #checking that the output constraint is true ie different prediction
        is_different = self.output_constraint(chromosome)
        if is_different:
            #for valid counterfactuals, the higher the score the fewer the changes which is better
            fitness_score = objective_result
        else:
            #invalid counterfactuals get a score of 0
            fitness_score = 0

        fitness_scores.append(fitness_score)

    return fitness_scores
```

(a) Old Fitness Function

```python
def fitness(self, population: List[np.ndarray])-> List[float]:
    #batch process fitness calculation
    objective_results = self.objective_function(population)
    different_predictions = self.output_constraint(population)

    #vectorised fitness calculation
    fitness_scores = np.where(different_predictions, objective_results, 0)
    return fitness_scores.tolist()
```

(b) New Fitness Function

Figure 2: Comparison of Fitness Functions

- Batch processing has been used to handle the entire population of candidate solutions in a single operation, rather than processing each individual one at a time. In this genetic algorithm, batch processing is used in several key places, such as when encoding the population, making model predictions, and calculating distances. For example, instead of predicting the model output for each chromosome individually, the algorithm predicts the outputs for the whole population at once using vectorised operations. This approach significantly improves computational efficiency and reduces the runtime.

5.1.4 Crossover and Mutation Operators

In the crossover function, a crossover rate was added to provide more control over the exploration trade-off. It also allows some good solutions to pass unchanged to the next generation. Diversity is an important factor in this algorithm and by including a crossover rate, it promotes the creation of new and diverse chromosomes within the population as it determines how often chromosomes exchange their "genetic material". This helps the algorithm to explore a broader solution space and find more varied counterfactuals.

To further promote diversity in the population, a multiple-point crossover function was implemented to replace the single-point approach. This mechanism creates multiple split points within each chromosome during recombination, resulting in more extensive genetic exchange between parent solutions. By introducing greater variability in offspring chromosomes, the algorithm generates a more diverse set of candidate counterfactuals.

The mutation function was also updated to better work with both continuous and categorical data. It checks for whether the genes are categorical or continuous to either bit flip (for categorical data) or implement the normal distribution (to work with continuous data).

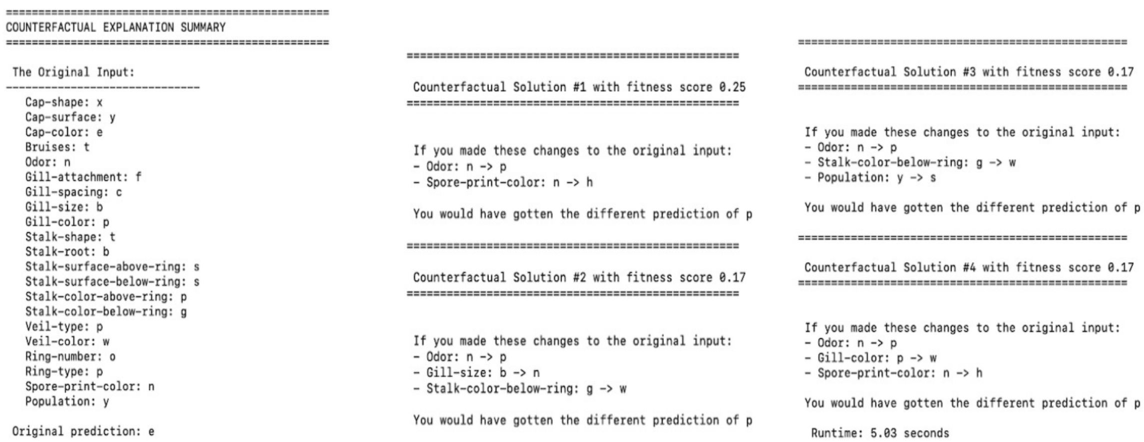
For whether it is categorical or continuous, it is determined by:

- If a feature is a string, it is treated as categorical.
- If the feature is numeric, the algorithm checks the number of unique values. If there are only a few unique values, it is likely categorical. If the feature has many unique values or is a floating-point number, it is treated as continuous.

5.2 Implementation Results

The genetic algorithm successfully outputs four different counterfactual explanations from the three datasets used, giving a user multiple options to implement.

An example output can be seen in Figure 3 working with one of the datasets. Examples of the outputs for the other datasets tested with this algorithm have been included in the appendix.



```
==================================================
COUNTERFACTUAL EXPLANATION SUMMARY
==================================================

 The Original Input:
------------------------------
   Cap-shape: x
   Cap-surface: y
   Cap-color: e
   Bruises: t
   Odor: n
   Gill-attachment: f
   Gill-spacing: c
   Gill-size: b
   Gill-color: p
   Stalk-shape: t
   Stalk-root: b
   Stalk-surface-above-ring: s
   Stalk-surface-below-ring: s
   Stalk-color-above-ring: p
   Stalk-color-below-ring: g
   Veil-type: p
   Veil-color: w
   Ring-number: o
   Ring-type: p
   Spore-print-color: n
   Population: y

 Original prediction: e

==================================================

 Counterfactual Solution #1 with fitness score 0.25
==================================================


 If you made these changes to the original input:
 - Odor: n -> p
 - Spore-print-color: n -> h

 You would have gotten the different prediction of p

==================================================

 Counterfactual Solution #2 with fitness score 0.17
==================================================


 If you made these changes to the original input:
 - Odor: n -> p
 - Gill-size: b -> n
 - Stalk-color-below-ring: g -> w

 You would have gotten the different prediction of p

==================================================

 Counterfactual Solution #3 with fitness score 0.17
==================================================


 If you made these changes to the original input:
 - Odor: n -> p
 - Stalk-color-below-ring: g -> w
 - Population: y -> s

 You would have gotten the different prediction of p

==================================================

 Counterfactual Solution #4 with fitness score 0.17
==================================================


 If you made these changes to the original input:
 - Odor: n -> p
 - Gill-color: p -> w
 - Spore-print-color: n -> h

 You would have gotten the different prediction of p

  Runtime: 5.03 seconds
```

Figure 3: Improved Implementation Output Example

5.2.1 Output Usability

The counterfactual output was designed to be user friendly, clearly showing:

1. The original input and its classification
2. The fitness score for each counterfactual
3. Multiple counterfactual explanations (typically four)
4. The specific feature changes for each counterfactual

5.3 Implementation Usage

The primary use case for this implementation is to generate counterfactual explanations for predictions made by a machine learning model. This is useful for:

- Model interpretability: Helping users understand what minimal changes to input features would alter the model's prediction and identifying which features most influence the model's decisions
- User targeted explanations: Providing actionable feedback to end-users

In order for the system to work, there must be:

1. A dataset:
   - The dataset should be in CSV format and contain the features and target variable.
   - Users select the dataset by editing dataset.py and uncommenting the relevant line.
2. A machine learning model:
   - The system supports several models (e.g., Logistic Regression, Random Forest).
   - Users select the model by editing ml_model.py and uncommenting the relevant line.

3. Algorithm parameters:
   - Users can adjust parameters such as population size, number of generations, mutation rate, etc., in genetic_algorithm.py.
4. Reference input:
   - By default, the system selects a random instance from the dataset as the reference input.

To run the genetic algorithm, the following command must be executed in the terminal:

```
python3 genetic_algorithm.py
```

The resulting output will print to the console the original input and its prediction as well as several diverse counterfactual explanations, each showing the feature changes required to alter the reference input's prediction.

6 Evaluation Strategy

6.1 Datasets

The implementation will be evaluated on three datasets with varying dimensionality to asses the algorithm's robustness. From the number of features ranging from 7 to 21 and the number of instances going from 286 to 8124, the algorithm's performance is evaluated across different dataset complexity levels. Two categorical datasets from the UCI machine learning repository (https://archive.ics.uci.edu/) and one used from Smith et al. [Smith et al., 1988] for continuous data representation were employed, with a mix of quantitative and qualitative data being handled. Each dataset will be split into training and testing sets.

The datasets used are:

- Mushroom Dataset: A categorical dataset that classifies if a mushroom is poisonous or edible. To download visit: `https://archive.ics.uci.edu/dataset/73/mushroom`
- Breast Cancer Dataset: A categorical dataset that diagnoses breast cancer as recurring or non-recurring. To download visit: `https://archive.ics.uci.edu/dataset/14/breast+cancer`
- Diabetes Dataset: A continuous dataset that classifies if a person has diabetes or not. To download visit:
  `https://github.com/fraleo/robust_counterfactuals_aaai24/tree/main/datasets/diabetes`
  (however the "Outcome" heading was changed to "Class" in the CSV for the target variable)

The diabetes dataset was also used in Leofante and Potyka's [Leofante and Potyka, 2024] research, who also implemented the DiCE algorithm [Mothilal et al., 2020] with it. Therefore, including this dataset to evaluate the effectiveness of this genetic algorithm allowed comparison between the different methodologies.

6.2 Hardware

All experiments were conducted on a Mac with 16 GB RAM and Apple M1 processor running macOS [version 12.1].

6.3 Protocol

Before the genetic algorithm can begin, a reference input is required. In this implementation, the reference input is randomly selected from the valid instances in the dataset. The genetic algorithm then begins by initialising this random population. Each individual is evaluated using a fitness function and an evolutionary process is continued until the number of generations is reached or until the early stopping criteria are met, producing four counterfactual explanations.

6.4 Evaluation Metrics

Performance metrics for the counterfactuals include the distance between the reference input and the outputted counterfactual using the Manhattan distance.

To evaluate diversity, two different papers' approaches were applied. The first being the diversity calculation from Promoting Counterfactual Robustness through Diversity (used in [Leofante and Potyka, 2024]) shown in equation 6.

$$k - \text{diversity}(S) = \frac{1}{\binom{|S|}{2}} \sum_{j=1}^{|S|-1} \sum_{l=j+1}^{|S|} dist(x'_j, x'_l) \tag{6}$$

Where:

- S is a set of counterfactuals
- dist is the Manhattan distance

The second being the two diversity calculations from Explaining Machine Learning Classifiers through Diverse Counterfactual Explanations (used in [Mothilal et al., 2020]) shown in equations 7 & 8.

$$\text{Count-Diversity} : \frac{1}{C_k^2 d} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{l=1}^{d} \mathbf{1}_{[c_i^l \neq c_j^l]} \tag{7}$$

$$\text{Diversity} : \frac{1}{C_k^2} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \text{dist}(c_i, c_j), \tag{8}$$

Where:

- k is the total number of counterfactuals generated
- d is the total number of input features
- c represents a counterfactual instance
- dist is the Manhattan distance

6.5 Parameters

Parameter tuning for genetic algorithm operators is challenging as there are many different parameters used. A Python file called parameter_testing.py was created to test multiple random parameter values (population size, number of generations, mutation rate, crossover rate, minimum distance threshold, and a stagnation limit) to optimise the diversity (equation 6) whilst ensuring the runtime was taken into account.

The algorithm involves six key parameters that influence its behaviour and output quality (the tested parameter value ranges being shown in brackets):

1. Population size (30-100): Controls the number of candidate solutions maintained in each generation
2. Mutation rate (0.05-0.3): Determines the probability of random changes to individual features
3. Crossover rate (0.6-0.9): Controls the likelihood of combining features from parent solutions
4. Minimum distance threshold (0.3-0.8): Sets the required dissimilarity between counterfactuals
5. Number of generations (50-200): Defines the maximum evolution iterations
6. Stagnation limit (10-30): Triggers early stopping when no improvements occur

For each combination, 10 random parameter combinations were generated within these ranges and were evaluated using the diversity metric in equation 6 [Leofante and Potyka, 2024]:

- the genetic algorithm is run
- a measure of performance metrics is taken
- outputs k-diversity and runtime

7 Results

The project deliverables consist of evaluating the algorithms ability to produce counterfactuals that are diverse, robust, and have an appropriate runtime. This section shows that the results do achieve these targets.

7.1 Parameters

To optimise the genetic algorithm's performance across all three datasets, a systematic parameter testing process was conducted. For each dataset, it was tested using multiple parameter combinations to find the optimal combination that balances diversity with computationally efficiency. The parameter testing ranges are outlined in Section 6.5.

For each dataset, 10 different parameter combinations were tested measuring the k-diversity score (equation 6) and the runtime performance. The results shown in Table 2, demonstrates the best combinations for the three datasets.

Table 2: Parameter Testing Results (using parameter_testing.py)

	Cancer Recurrence	**Diabetes**	**Mushroom**
Population Size	42	68	62
Mutation Rate	0.15	0.14	0.20
Crossover Rate	0.72	0.74	0.77
Min Dist Threshold	0.61	0.55	0.67
No. Generations	136	85	122
Stagnation Limit	24	10	12
Diversity Score	6.00	1.82	9.00
Runtime (seconds)	1.53	3.69	3.30

After analysing the results, the average for each parameter was taken to create a general purpose configuration that preforms well with all three datasets. The recommended parameter values are:

1. Population size : 57 (balanced for computational efficiency and solution exploration)
2. Mutation rate : 0.16 (maintains solution quality while also encouraging exploration)
3. Crossover rate : 0.74 (encouraging exploration for different chromosomes)

4. Minimum distance threshold : 0.61 (ensures meaningful diversity between counterfactuals)
5. Number of generations : 114 (sufficient for convergence while avoiding unnecessary computations)
6. Stagnation limit : 15 (allows enough room for improvement while preventing excessive runtimes)

7.2 Runtimes

Algorithm efficiency was evaluated across multiple configurations. This was achieved by measuring the run time with the three datasets of Cancer Recurrence, Diabetes, and Mushroom, with five reference inputs per dataset.

For each dataset, three different population sizes were tested (30, 40, 50) and three generation sizes (50, 100, 150). The runtime was taken from the start of the genetic algorithm until it produced the final set of counterfactuals.

The results, shown in Figures 4, 5, and 6 shows the relationship between different population sizes and number of generations with runtimes.

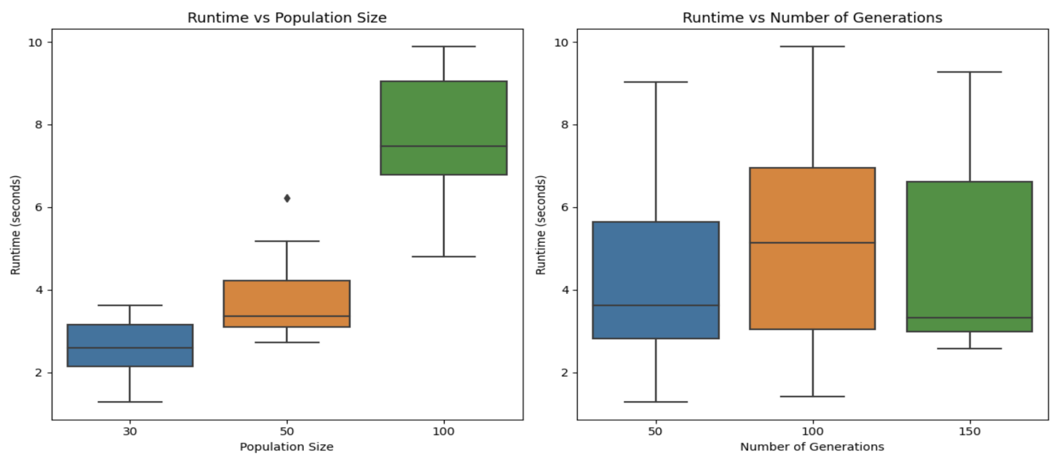


Figure 4: Mushroom Dataset Runtime Evaluation Results

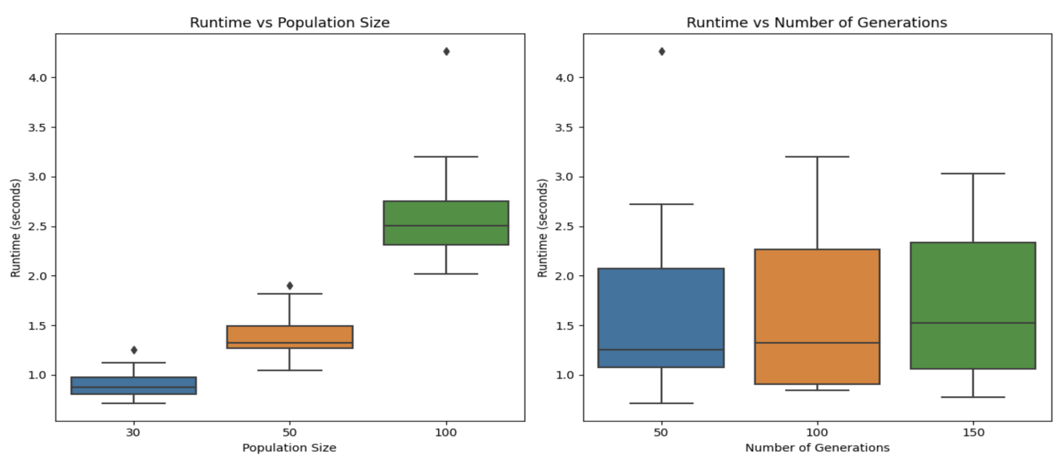


Figure 5: Cancer Recurrence Dataset Runtime Evaluation Results

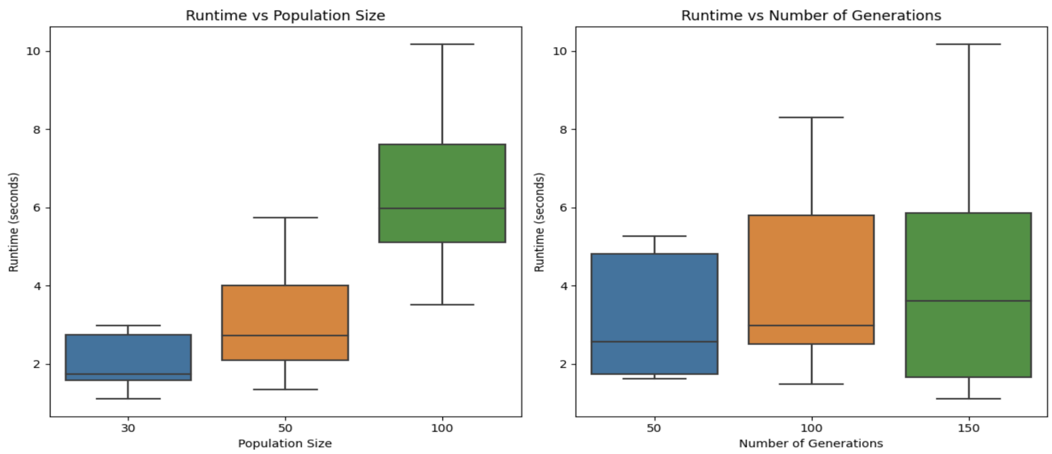


Figure 6: Diabetes Dataset Runtime Evaluation Results

Table 3: Runtime Analysis Testing

Parameter	**Cancer Recurrence**		**Mushroom**		**Diabetes**	
	mean	**std**	**mean**	**std**	**mean**	**std**
By Population Size						
population_size = 30	0.90	0.15	2.60	0.72	2.03	0.67
population_size = 50	1.39	0.23	3.82	1.01	3.15	1.52
population_size = 100	2.62	0.57	7.72	1.46	6.22	1.80
By Number of Generations						
num_generations = 50	1.61	0.95	4.44	2.37	3.08	1.51
num_generations = 100	1.63	0.80	5.07	2.73	4.02	2.26
num_generations = 150	1.67	0.73	4.62	2.38	4.30	2.80

These Figures 4, 5, and 6 and Table 3 show that the number of generations does not have as significant an impact as the population size to runtimes. The impact was especially clear with an increase in population size from 50 to 100, a significant increase in runtime occurs, with at least a 50% increase in runtime for every dataset. Hence a population size of 57 has been used in the genetic algorithm as it still has a relatively quick runtime, but additionally, as shown in the parameter testing results, all datasets worked efficiently at this population size as well (Section 7.1).

7.3 Diversity

To evaluate the diversity of the counterfactual explanations generated by the algorithm, two different diversity metrics were employed: Equation (6), the k-diversity metric proposed by Leofante and Potyka [Leofante and Potyka, 2024] and the diversity measure used in DiCE [Mothilal et al., 2020], shown in Equation (7) and (8).

This was tested across the three datasets: Cancer Recurrence, Diabetes, and Mushroom classification, and for each dataset, the diversity was measured along with the runtime. For each dataset, 50 different reference instances were chosen randomly and used to generate counterfactual explanations. The choice of 50 runs was made to match the evaluation methodology of Leofante and Potyka [Leofante and Potyka, 2024], allowing comparison between results.

The results are shown in Table 4, and illustrate the mean and standard deviation for both diversity metrics.

Table 4: Diversity Results Comparing K-diversity and DiCE Metrics

Dataset	**Diversity Metric**	**Diversity Score**	**Average Runtime (s)**	**Number of Counterfactuals**
Cancer recurrence	K-diversity	3.22 (±1.24)	1.64 (±0.29)	4
	DiCE	3.56 (±1.58)	1.70 (± 0.30)	4
Mushroom classification	K-diversity	5.68 (±1.34)	5.07 (± 1.15)	4
	DiCE	5.23 (± 1.24)	5.13 (± 1.18)	4
Diabetes	K-diversity	1.31 (±0.64)	3.95 (± 1.26)	4
	DiCE	0.25 (± 0.11)	4.08 (± 1.39)	4

Significant variation can be seen across the datasets in terms of k-diversity scores. The categorical datasets (Cancer Recurrence and Mushroom classification) show higher diversity scores compared to the Diabetes dataset. This discrepancy can be due to the nature of the different feature spaces due to changes in the categorical features representing more significantly different counterfactuals than small adjustments to continuous variables (which the features in the diabetes dataset has).

Interestingly, out of the two diversity metrics, the algorithm demonstrates similar performance when being evaluated with the k-diversity metric and DiCE diversity metric for the categorical data, but the Diabetes dataset achieves a much worse diversity score with the DiCE diversity metric.

The clear difference in scores for the Diabetes dataset between the two evaluation metrics is caused from their different normalisation approaches: DiCE normalises continuous diversity by both the number of continuous features and feature pairs, whereas Leofante and Potyka normalise the total distance solely by the number of pairs. This methodological difference explains the dissimilar results observed when comparing these algorithms.

Through the normalisation of the diversity metric DiCE describes, the continuous features in Diabetes have smaller ranges, leading to smaller differences. In contrast, the Leofante and Potyka metric produces higher diversity scores because it uses the Manhattan distance consistently across all features without normalisation, allowing continuous features to contribute their full range to the distance calculation.

The runtime performance across both diversity evaluations are consistent, with the DiCE diversity calculation always taking slightly longer. The Cancer Recurrence and Mushroom datasets runtime only differs by 0.06 seconds and the Diabetes dataset by 0.13.

Diversity Comparison Results

To compare the results to the algorithms employed by Leofante and Potyka [Leofante and Potyka, 2024], and DiCE [Mothilal et al., 2020] with this paper, Table 5 shows the diversity scores achieved implementing these different algorithms, using the same Diabetes dataset.

Table 5: Comparison Diversity Results

Metric	**Genetic Algorithm (This Paper)**	**Leofante and Potyka**	**DiCE**
Diversity	1.31 (±0.64)	1.39 (±0.46)	1.32 (±0.19)
Runtime (s)	3.95 (± 1.26)	0.02 (± 0.00)	72.66 (± 32.57)

As shown in Table 5, this algorithm shows promise of diversity within the counterfactuals compared with the other two approaches. While the diversity scores are all relatively close, the differences in runtime are particularly more significant. The genetic algorithm approach (3.95s) is between the extremely fast approach (0.02s) and the computationally intensive gradient-based method (72.66s). This positioning suggests that the genetic algorithm approach offers a reasonable trade-off between computational efficiency and solution quality.

The diversity score of 1.31 also has greater variation as the standard deviation is 0.64, compared to the 0.46 and 0.19 achieved in the other algorithms. This could be due to genetic algorithms being stochastic, and although each run has a controlled randomness from the algorithm design, more variation occurs naturally in genetic algorithms.

This relatively high diversity score is not primarily due to small numerical differences. Given the number of features in the Diabetes dataset, tiny changes (e.g., 53.0 $\rightarrow$ 53.1) would contribute minimally to the final score. Suggesting that the algorithm is effectively generating diverse solutions for continuous features.

However, as the different features in the Diabetes dataset are not all on the same scale, this can inflate the diversity score. For example, a small change in a feature with a large range (e.g., Age) will contribute much more than a small change to a feature with a small range.

Therefore, this paper's diversity (1.31) is only slightly lower than Leofante and Potyka (1.39) and Dice (1.32), implying that it is still a good algorithm for generating diverse counterfactuals with a reasonable runtime.

7.4 Robustness

The robustness testing evaluates how stable the genetic algorithm's counterfactual generation is when faced with small variations in the input. The test works by:

1. Taking an original input and generating its counterfactuals
2. Creating 50 slightly modified versions of this input (perturbations) where:
   - 1-2 random features are changed
   - For numeric features, a 10% noise level is applied based on the feature's standard deviation
   - For categorical features, a different valid value is selected
3. Generating counterfactuals for each perturbed input
4. Measuring the Manhattan distance between counterfactuals from the original and perturbed inputs
5. Calculating an overall robustness score based on the average distances

This approach helps verify that the algorithm produces consistent and stable counterfactual explanations even when the input data has minor variations. The two metrics used are:

1. Average Distance: This is the mean Manhattan distance between counterfactuals from original and perturbed inputs. Lower values indicate more stable counterfactuals
2. Average Distance per Feature: The average (Manhattan) distance divided by the number of features, normalising for the different dataset feature spaces. This helps compare robustness across different datasets, with lower values also indicating more stable counterfactuals

The robustness of the counterfactuals for each dataset can be seen in Table 6.

Table 6: Algorithm Robustness Results

Dataset	**Average Distance**	**Average dist per Feature**
Cancer recurrence	5.34 ($\pm$ 2.15)	0.75
Mushroom classification	8.19 ($\pm$ 2.56)	0.42
Diabetes	0.68 ($\pm$ 0.27)	0.12

Direct comparison between datasets requires knowledge of their characteristics. The average distance values are significantly influenced by both the type and number of features in each dataset. Categorical features (as in Cancer Recurrence and Mushroom datasets) typically show larger distance changes when perturbed compared to continuous features (as in the Diabetes dataset). This is because that changing categorial features results in a larger distance for the perturbations. Since robustness means that small changes should result in small differences, it is perfectly acceptable that larger differences are shown for categorical variables (as they are justified by larger changes).

Additionally, the number of features varies considerably: Cancer Recurrence has 8 features, Diabetes has 7, and the Mushroom dataset consists of 21 features. This means that changing 1-2 features represents a different proportion of the total features in each dataset (13-29% for Cancer/Diabetes vs 5-10% for Mushroom). To enable meaningful comparison, the average distance per feature is also calculated, which normalises the robustness score by the number of features and provides a more balanced metric for comparing datasets with different characteristics.

The Diabetes dataset achieves the lowest score of 0.68, showing the highest levels of robustness. The very low distance score suggests that counterfactuals remain very similar even when the input is perturbed. However, this could represent a lack of diversity, as the diabetes dataset has the lowest diversity scores. Hence the algorithm may be finding very similar counterfactuals regardless of input perturbations. As well, small changes may not have as big an impact compared to when a categorical feature is changed, so it does not necessarily mean that the categorical data has worse robustness.

The Cancer Recurrence dataset shows moderate robustness and moderate diversity, implying a good middle ground between diversity and robustness. The counterfactuals change enough to provide some different explanations, but not enough to lack major stability.

The Mushroom dataset shows the least amount of robustness in terms of absolute distance (8.19), but when normalised per feature, the Mushroom dataset actually shows better robustness (0.42) compared to the Cancer Recurrence dataset (0.75). This is particularly significant because the Mushroom dataset has 21 features, meaning each perturbation affects a smaller proportion of the total features (5-10%) compared to Cancer Recurrence (13-25%). Despite having more features that could potentially change, the average change per feature is smaller, this suggests that the algorithm maintains more stability for the Mushroom dataset. This apparent contradiction between absolute and normalised distances highlights the importance of considering the per-feature metric when comparing robustness across datasets with different characteristics.

8 Future Recommendations

This genetic algorithm framework provides a foundation for further improvements, with the insights gained informing future methods areas of strengths and weaknesses when applying a genetic algorithm to generate counterfactual explanations.

In terms of feasibility, while steps have been taken to ensure that any change to a feature is valid (such as ranges for continuous data or choosing a new feature category), better checks for the feasibility of a counterfactual could be included. However, since no domain knowledge was known for the datasets used, only implementing age to be able to increase was included. To personalise counterfactual generation to users, the current algorithm could be modified to include user-defined constraints through a simple dictionary interface. For example, users could specify which features they want to modify (e.g., "age": True, "pregnancies": False), and the genetic algorithm would respect these constraints during mutation and crossover operations. Consequently, integrating domain knowledge and user suggestions could further improve the quality and usability of the counterfactuals.

For robustness, the current distance-based evaluation could be enhanced by integrating sensitivity to input perturbations directly into the search process. This would involve penalising solutions that are highly sensitive to small changes in the input, ensuring that the generated counterfactuals are stable and reliable. Additionally, the diversity mechanism could be improved by encouraging changes to different sets of features across the outputted counterfactuals, providing users with more varied and meaningful alternatives.

These improvements would enhance the practicality of the counterfactual explanations while maintaining the algorithm's core strengths in generating diverse and robust valid solutions.

9 Conclusion

This research approach demonstrates that genetic algorithms can successfully produce multiple counterfactual explanations that reasonably balance diversity and robustness while maintaining fair computational efficiency.

The genetic algorithm's inherent stochastic nature proved to be both an advantage and a challenge. This randomness naturally helps improve diversity among the explanations, but also undermines the robustness as similar inputs did not always have consistent explanations.

Computational efficiency remains a significant challenge in counterfactual explanation generation, as found in previous research. However, this implementation reduces this by: early stopping mechanisms that avoid unnecessary iterations and efficient vectorised operations for fitness calculations. These enhancements allowed the algorithm to perform competitively despite the inherently complex search space.

A strength of the genetic algorithm approach is its flexibility across different data types and problem statements. The algorithm handles mixed feature types (continuous and categorical) through specialised encoding mechanisms, implements the Manhattan distance metrics tailored to different use cases, and adapts objective functions to specific explanation requirements. This adaptability makes the approach applicable across diverse ML models and problems.

This work also identified limitations of genetic algorithms for computing counterfactuals. While the approach improves robustness through diversity, genetic algorithms cannot give theoretical guarantees regarding the stability of explanations under input perturbations. The runtime performance should also be further optimised for user experience.

Finally, the balance between diversity and minimal changes needed continues to present an ongoing research challenge. While the flexible parameters allow greater balance between the two, a more structured approach might involve multi-objective optimisation techniques or adaptive weighting schemes that dynamically balance these goals based on the specific explanation context.

Overall, this genetic algorithm-based approach represents a valuable contribution to the field of counterfactual explanations, offering a flexible and effective framework that can be further refined to address the remaining challenges in robustness, diversity, and efficiency.

10 Reflection on Learning

This research project has significantly deepened my understanding of Explainable AI while developing valuable research and technical skills. Through this work, insights have been developed in multiple areas.

Regular communication with my supervisor proved invaluable in improving the approach and strengthening the justification for the algorithmic choices. I learnt to articulate technical solutions concisely and respond to constructive feedback. Following a project plan while remaining flexible enough to incorporate new ideas or overcome unplanned challenges whilst satisfying the research objectives was a further self-development skill I have gained.

Although my initial focus was to generate any counterfactual that changed the prediction, I came to recognise that feasibility constraints (such as respecting that age cannot decrease) are equally as important for producing useful explanations. This realisation has reinforced the importance of incorporating domain knowledge to explainability solutions that must align with real-world constraints.

An important insight from this project was understanding the relationship between diversity and robustness in counterfactual explanations. My initial assumption that maximising diversity would instinctively improve explanation quality proved over-simplistic. This experience highlights a valuable lesson in research methodology that even when experimental outcomes differ from initial expectations, they can still provide useful insights.

This research highlighted how explanations that seem intuitive can still be problematic if they are not robust; consequently, questioning assumptions about what makes technology user-friendly is an important consideration. An understanding of the trade-offs between explanation simplicity, computational efficiency, and explanation reliability has also been gained.

Problem solving and adaptability were additional skills gained during this project. When initial approaches failed to generate sufficiently robust or diverse counterfactuals, I learnt to pivot to new ideas like the diversity selection function and refining the evaluation approach for diversity calculation.

Developing an evaluation framework for counterfactual quality taught me that effective assessment requires multiple metrics. Evaluating robustness, parameter sensitivity, diversity, and computational efficiency improved my critical assessment skills and taught me the value of building on established metrics in the field while adapting them to the specific context of my research.

I really enjoyed this research topic, particularly since Explainable AI represents a critical emerging field that will transform how users understand and trust AI-powered systems in the future, specifically in domains like healthcare and finance. Working with counterfactual explanations not only enhanced my own technical abilities, but illustrated how these approaches can provide users with transparent, actionable insights.

The skills I have developed through this project have provided me with a strong foundation for future work in this field. Being able to say I have made an algorithm that helps to understand how these models make their decisions, is a proud moment for me. Overall, these learnings have significantly strengthened my technical and research capabilities, providing a stronger foundation for future work in Explainable AI and algorithm development generally.

11 Appendix

Model Selection

To see which machine learning model would work best, four different ones have been evaluated:

- Logistic Regression
- Random Forests
- Decision Trees
- Multi-Layer Perceptron (neural network classifier)

Here an analysis was conducted to see which machine learning model works best for these factors:

  - Producing model accuracy on the testing and training data.
  - Average Runtimes

All datasets were evaluated however the Mushroom dataset produced 100% accuracy across all models due to the features being able to almost perfectly separate the classes (edible vs. poisonous) therefore the other two datasets, Diabetes and Cancer Recurrence, were used for evaluation. Every test was run ten times and an average and standard deviation was taken.

Random Forest Model

Results:

1. Cancer Recurrence:
   - Average Training Accuracy : 0.976 ($\pm$0.002)
   - Average Testing Accuracy : 0.771 ($\pm$0.072)
   - Average Runtime : 1.75 seconds ($\pm$0.25)
2. Diabetes:
   - Average Training Accuracy : 1.000 ($\pm$0.000)
   - Average Testing Accuracy : 0.787 ($\pm$0.056)
   - Average Runtime : 4.417 seconds ($\pm$ 1.2726)

The random forest results indicate over-fitting, as the training data accuracy is much higher than the test data accuracy.

Logistic Regression Model

Results:

1. Cancer Recurrence:
   - Average Training Accuracy : 0.797 ($\pm$0.013)
   - Average Testing Accuracy : 0.771 ($\pm$0.072)
   - Average Runtime : 2.401 seconds ($\pm$0.196)
2. Diabetes:
   - Average Training Accuracy : 0.780 ($\pm$0.004)
   - Average Testing Accuracy : 0.775 ($\pm$0.031)
   - Average Runtime : 4.649 seconds ($\pm$ 1.477)

The Logistic Regression testing and training accuracy values are similar for both datasets, meaning it is working well with unseen data.

Decision Tree Classifier Model

This model works by making a series of binary decisions at each node, following branches based on feature values until reaching a leaf node that indicates the classification [Ying et al., 2015].

Results:

1. Cancer Recurrence:
   - Average Training Accuracy : 0.978 ($\pm$0.002)
   - Average Testing Accuracy : 0.650 ($\pm$0.073)
   - Average Runtime : 1.4280 seconds ($\pm$ 0.1541)
2. Diabetes:
   - Average Training Accuracy : 1.000 ($\pm$0.000)
   - Average Testing Accuracy : 0.687 ($\pm$0.053)
   - Average Runtime : 3.517 seconds ($\pm$ 0.947)

These results indicate over-fitting as the decision tree seems to have memorised the training data and therefore preforms worse with the test data as it is unseen.

Multi-Layer Perceptron Model

A Multi-Layer Perceptron Model was analysed after. This is a type of neural network with multiple layers of nodes [Jaiswal, 2025].

Results:

1. Cancer Recurrence:
   - Average Training Accuracy : 0.918 ($\pm$0.011)
   - Average Testing Accuracy : 0.764 ($\pm$0.072)
   - Average Runtime : 1.942 seconds ($\pm$0.288)
2. Diabetes:
   - Average Training Accuracy : 0.809 ($\pm$0.007)
   - Average Testing Accuracy : 0.761 ($\pm$0.056)

- Average Runtime : 3.626 seconds (± 0.996)

This indicates the same over-fitting problem as seen in the decision trees but on a slightly smaller scale. This could be due to the fact that the dataset size is too small relative to the neural networks and consequently it can easily memorise the training data set. It also shows a convergence warning whenever it is run using this machine learning model, indicating that the model has not reached a stable solution within the allocated solutions and so this is an additional reason why this was not used.

Improved Implementation Outputs

```
==================================================
COUNTERFACTUAL EXPLANATION SUMMARY
==================================================

 The Original Input:
------------------------------
  Glucose: 132.0
  BloodPressure: 0.0
  SkinThickness: 0.0
  Insulin: 0.0
  BMI: 32.9
  DiabetesPedigreeFunction: 0.302
  Age: 23.0

 Original prediction: 1

==================================================

 Counterfactual Solution #1 with fitness score 1.80
==================================================


 If you made these changes to the original input:
 -Glucose: 132.0 -> 119.0
 -BMI: 32.9 -> 33.7
 -DiabetesPedigreeFunction: 0.302 -> 0.29
 -Age: 23.0 -> 25.1

 You would have gotten the different prediction of 0

==================================================

 Counterfactual Solution #2 with fitness score 1.18
==================================================


 If you made these changes to the original input:
 -Glucose: 132.0 -> 119.0
 -BMI: 32.9 -> 32.4
 -DiabetesPedigreeFunction: 0.302 -> 0.141
 -Age: 23.0 -> 24.0

 You would have gotten the different prediction of 0

==================================================

 Counterfactual Solution #3 with fitness score 0.63
==================================================


 If you made these changes to the original input:
 -Glucose: 132.0 -> 119.0
 -BMI: 32.9 -> 33.2
 -DiabetesPedigreeFunction: 0.302 -> 0.1
 -Age: 23.0 -> 35.0

 You would have gotten the different prediction of 0

==================================================

 Counterfactual Solution #4 with fitness score 0.41
==================================================


 If you made these changes to the original input:
 -Glucose: 132.0 -> 93.0
 -BMI: 32.9 -> 21.1
 -DiabetesPedigreeFunction: 0.302 -> 0.342
 -Age: 23.0 -> 25.0

 You would have gotten the different prediction of 0

  Runtime: 3.35 seconds
```

Figure 7: Diabetes Improved Implementation output Example

```
==================================================
COUNTERFACTUAL EXPLANATION SUMMARY
==================================================

 The Original Input:
------------------------------
   age: 30-39
   menopause: premeno
   tumor-size: 30-34
   inv-nodes: 3-5
   node-caps: no
   deg-malig: 3
   breast: right
   breast-quad: left_up

 Original prediction: recurrence-events

==================================================

 Counterfactual Solution #1 with fitness score 0.50
==================================================


 If you made these changes to the original input:
 - deg-malig: 3 -> 2

 You would have gotten the different prediction of no-recurrence-events

==================================================

 Counterfactual Solution #2 with fitness score 0.50
==================================================


 If you made these changes to the original input:
 - deg-malig: 3 -> 1

 You would have gotten the different prediction of no-recurrence-events

==================================================

 Counterfactual Solution #3 with fitness score 0.50
==================================================


 If you made these changes to the original input:
 - menopause: premeno -> lt40

 You would have gotten the different prediction of no-recurrence-events

==================================================

 Counterfactual Solution #4 with fitness score 0.25
==================================================


 If you made these changes to the original input:
 - age: 30-39 -> 20-29
 - deg-malig: 3 -> 1

 You would have gotten the different prediction of no-recurrence-events

  Runtime: 2.57 seconds
```

Figure 8: Cancer Recurrence Improved Implementation output Example

References

[AlJalaud and Hosny, 2024] AlJalaud, E. and Hosny, M. (2024). Counterfactual explanation of ai models using an adaptive genetic algorithm with embedded feature weights. *IEEE Access*, 12:74993–75009.

[Bajpai and Kumar, 2010] Bajpai, P. and Kumar, M. (2010). Genetic algorithm–an approach to solve global optimization problems. *Indian Journal of computer science and engineering*, 1(3):199–206.

[Beasley et al., 1993] Beasley, D., Bull, D. R., and Martin, R. R. (1993). An overview of genetic algorithms: Part 1, fundamentals. *University computing*, 15(2):56–69.

[Brownlee, 2024] Brownlee, J. (2024). *Genetic Algorithm Afternoon*. Algorithm Afternoon Publishing.

[Dandl et al., 2020] Dandl, S., Molnar, C., Binder, M., and Bischl, B. (2020). Multi-objective counterfactual explanations. In Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., and Trautmann, H., editors, *Parallel Problem Solving from Nature – PPSN XVI*, pages 448–469, Cham. Springer International Publishing.

[Darwin, 1859] Darwin, C. (1859). *On the origin of species*.

[Du et al., 2018] Du, H., Wang, Z., Zhan, W., and Guo, J. (2018). Elitism and distance strategy for selection of evolutionary algorithms. *IEEE Access*, 6:44531–44541.

[Holland, 1992] Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[Jaiswal, 2025] Jaiswal, S. (2025). Multilayer perceptrons in machine learning: A comprehensive guide.

[Kala, 2024] Kala, R. (2024). Chapter 15 - an introduction to evolutionary computation. In Kala, R., editor, *Autonomous Mobile Robots*, Emerging Methodologies and Applications in Modelling, pages 715–759. Academic Press.

[Karafotias et al., 2015] Karafotias, G., Hoogendoorn, M., and Eiben, A. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE transactions on evolutionary computation: a publication of the IEEE Neural Networks Council*, 19(2):167–187.

[Kaya, 2011] Kaya, M. (2011). The effects of a new selection operator on the performance of a genetic algorithm. *Applied Mathematics and Computation*, 217(19):7669–7678.

[Leofante and Potyka, 2024] Leofante, F. and Potyka, N. (2024). Promoting counterfactual robustness through diversity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 21322–21330.

[Maaranen et al., 2007] Maaranen, H., Miettinen, K., and Penttinen, A. (2007). On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, 37:405–436.

[Miller et al., 1995] Miller, B. L., Goldberg, D. E., et al. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3):193–212.

[Mitchell, 1998] Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT Press.

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

[Poyiadzi et al., 2020] Poyiadzi, R., Sokol, K., Santos-Rodriguez, R., De Bie, T., and Flach, P. (2020). Face: Feasible and actionable counterfactual explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, AIES '20, page 344–350, New York, NY, USA. Association for Computing Machinery.

[Prado-Romero et al., 2024] Prado-Romero, M. A., Prenkaj, B., Stilo, G., and Giannotti, F. (2024). A survey on graph counterfactual explanations: Definitions, methods, evaluation, and research challenges. *ACM Comput. Surv.*, 56(7).

[Rothlauf, 2006] Rothlauf, F. (2006). *Representations for Genetic and Evolutionary Algorithms*, pages 9–32. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Russell, 2019] Russell, C. (2019). Efficient search for diverse coherent explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, page 20–28, New York, NY, USA. Association for Computing Machinery.

[Rutenbar, 1989] Rutenbar, R. (1989). Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26.

[Shapiro, 1999] Shapiro, J. (1999). Genetic algorithms in machine learning. In *Advanced course on artificial intelligence*, pages 146–168. Springer.

[Shukla et al., 2015] Shukla, A., Pandey, H. M., and Mehrotra, D. (2015). Comparative review of selection techniques in genetic algorithm. In *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, pages 515–519.

[Smith et al., 1988] Smith, J. W., Everhart, J. E., Dickson, W., Knowler, W. C., and Johannes, R. S. (1988). Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the annual symposium on computer application in medical care*, page 261. American Medical Informatics Association.

[Vavak and Fogarty, 1996] Vavak, F. and Fogarty, T. C. (1996). A comparative study of steady state and generational genetic algorithms for use in nonstationary environments. In *AISB workshop on Evolutionary Computing*, pages 297–304. Springer.

[Verma et al., 2020] Verma, S., Dickerson, J., and Hines, K. (2020). Counterfactual explanations for machine learning: A review. *arXiv preprint arXiv:2010.10596*, 2(1):1.

[Wachter et al., 2017] Wachter, S., Mittelstadt, B., and Russell, C. (2017). Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841.

[Wehrens et al., 1999] Wehrens, R., Pretsch, E., and Buydens, L. (1999). The quality of optimisation by genetic algorithms. *Analytica Chimica Acta*, 388(3):265–271.

[Ying et al., 2015] Ying, L. et al. (2015). Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130.

[Zainuddin et al., 2020] Zainuddin, F. A., Abd Samad, M. F., and Tunggal, D. (2020). A review of crossover methods and problem representation of genetic algorithm in recent engineering applications. *International Journal of Advanced Science and Technology*, 29(6s):759–769.