Metaheuristic and Evolutionary Algorithms in Explainable Artificial Intelligence

Hardik Prabhu *, Patrick Siarry †, Jayaraman Krishnamoorthy Valadi ‡, Aamod Sane, and Renu Dhadwal

Abstract There has been an increasing interest in Explainable Artificial Intelligence (XAI) in recent years. Complex machine learning algorithms, such as deep neural networks, can accurately predict outcomes, but provide little insight into how the decision was made or what factors influenced the outcome. This lack of transparency can be a major issue in high-stakes decision-making scenarios, where understanding the reasoning behind a decision is crucial. XAI aims to address the problem of the "black box" in machine learning models, where the AI's decision-making process is not transparent, and humans cannot understand how the AI arrived at a particular decision or prediction. Evolutionary and metaheuristic techniques offer promising avenues for achieving explainability in AI systems, and there is a lot of ongoing research in this area to further explore their potential. Our work is a concise literature review that explores the potential adoption of these techniques to facilitate the attainment of explainability in AI systems. We have highlighted some of the contributions of evolutionary and metaheuristic techniques in different approaches to achieving

Hardik Prabhu
FLAME University, Pune, INDIA 412115 e-mail: hardik.prabhu@flame.edu.in

Aamod Sane
FLAME University, Pune, INDIA 412115 e-mail: aamod.sane@flame.edu.in

Renu Dhadwal
FLAME University, Pune, INDIA 412115 e-mail: renu.dhadwal@flame.edu.in

Jayaraman Krisknamoorthy Valadi
FLAME University, Pune, INDIA 412115 e-mail: jayaraman.vk@flame.edu.inr

Patrick Siarry
Univ Paris Est Creteil, LISSI, 94400, Vitry, France e-mail: siarry@u-pec.fr

* Corresponding author

† Corresponding author

‡ Corresponding author

explainability, such as counterfactual explanations, local surrogate modelling, and the development of transparent models.

1 Explainable Artificial Intelligence (XAI)

Artificial intelligence (AI) has witnessed enormous progress and advancement in recent years. With the advent of deep learning and other sophisticated techniques, AI has surpassed human-level performance in many complex tasks. AI systems have found their application in diverse fields, including financial modelling, natural language processing, medical diagnosis, self-driving cars, and various other domains. However, the rise of AI has also raised concerns about the need for explainability [34].

There are ethical and societal implications associated with the use of AI. Incorporating AI in decision-making processes can perpetuate existing biases and discrimination [28]. Explainability can help to identify and address these biases, help to build trust in AI systems, ensure fair and unbiased decision-making, and enable accountability. As AI continues to advance and become more ubiquitous, the need for explainability will only continue to grow.

XAI Techniques

XAI in ML could be broadly classified into two categories; i) Transparent Models and ii) Post-hoc Explainability [2]. Transparent Models are those which are self explainable and they do not require any external tools in order to explain their predictions. Building Transparent models is ideal for explainability. However, it is very challenging to interpret many complex models such as deep neural networks which consist of thousands of parameters and are practically black-boxes. Another alternative is to build post-hoc tools which do not interfere with the model training but could be used to explain its prediction post-training. The post-hoc approaches are further divided into two; i) Model specific and ii) Model agnostic. The model-specific methods such as backwards propagation in neural networks necessitate awareness of the internal design of the model to facilitate the flow of information in the reverse direction. [29]. Model agnostic methods do not make any assumptions regarding the internal workings of a black-box model and rely only on querying the black-box model to make predictions whenever required.

Interpretability and Accuracy Trade-off

There is an inverse relation between model complexity and interpretability. From an interpretability standpoint, it is desired to have models which are simple and yet perform well on a given task. As the task gets increasingly complex, the simple models

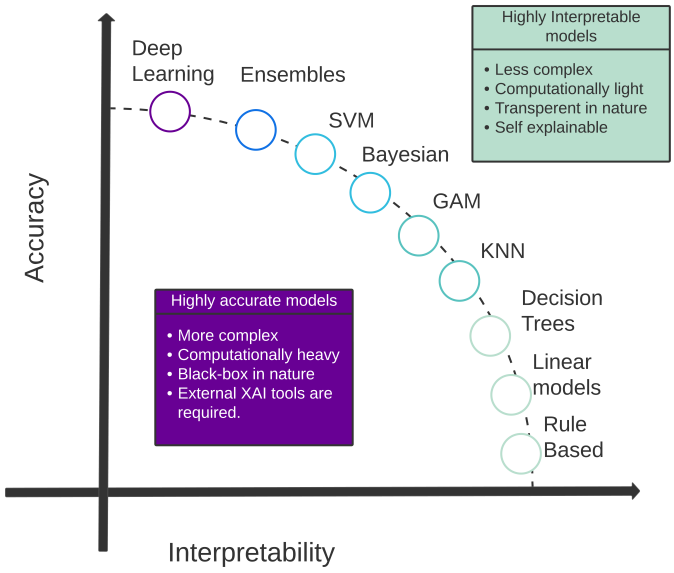

Fig. 1 The trade-off between machine learning model interpretability and accuracy.

may not be flexible enough and may perform poorly. Opting for more sophisticated models may boost the performance provided the training data is sufficiently large. It is important to note that complex models are prone to overfitting and may not generalise well if sufficient data is not available for training. If the task to learn is fairly complex such that a simple model may not perform well, and the training data is also sufficiently large such that a complex model may generalize well, then there exists a tradeoff between accuracy and interpretability as shown in Figure 1.

The remaining sections of this chapter are organized as follows: Section 2 presents a comprehensive overview of several common metaheuristics and evolutionary algorithms. Following this, Section 3 examines the application of these techniques in attaining explainability within AI systems.

2 Evolutionary and Metaheuristic Algorithms

Evolutionary algorithms draw inspiration from the natural selection process, where a population of candidate solutions is evolved through selection, crossover, and mutation operations, leading to the discovery of better solutions over time. Common examples of evolutionary algorithms include genetic algorithms, evolutionary strategies, and genetic programming.

Metaheuristic algorithms are a family of optimization techniques which, unlike traditional optimization methods, do not rely on mathematical models and explicit

problem-specific information. Metaheuristics are problem-independent and do not require a precise problem formulation. Instead, they use a set of heuristic rules and strategies to explore the search space and gradually improve the solution quality. These algorithms are intended to efficiently explore the search space by balancing exploration and exploitation of the solution space. Common examples of metaheuristic algorithms include simulated annealing, particle swarm optimization, differential evolution and ant colony optimization.

Evolutionary algorithms and metaheuristics are powerful tools that can be used out of the box to address a vast array of problems, including XAI [3]. This section provides detailed descriptions of some of these algorithms, which are employed in the XAI techniques discussed in the following section.

2.1 Genetic Algorithms

Genetic algorithms (GAs) [18] are a class of optimization algorithms that are inspired by the principles of natural selection. These algorithms work by imitating the process of evolution. These algorithms work by iteratively improving a population of potential solutions to a problem through operations like selection, crossover, and mutation. GA has been extensively researched and implemented in a variety of disciplines, such as engineering, computer science, economics, and biology, to solve complex optimization problems that are difficult or impossible to solve with traditional methods. Genetic algorithms are popular because of their ability to find optimal or near-optimal solutions in large, complex search spaces and their ability to handle non-linear and non-continuous objective functions.

Genetic Representation

To implement GA, one must first devise a way to represent the solution space. Encoding in genetic algorithms entails representing each potential solution to a problem as a chromosome which is a set or a vector of genes. This process is also known as chromosome encoding. In earlier works, binary encoding was popularized. Real value representation is another popular encoding mechanism. A more detailed review of various types of encoding is done in [23]. Value encoding is a good choice for encoding real-valued variables. For example, if the objective is to maximize $f(x_1, x_2) = sin(x_1)+x_2$ then it may seem logical to encode the inputs as $x = [x_1, x_2]^\top$, a real valued chromosome.

Main Components of the GA

- Fitness Function: Each solution in the population is assigned a fitness value by the fitness function, which assesses the quality of a solution based on how well it solves the problem.
- Selection: The selection operation chooses a subset of solutions from the existing population, based on their fitness values, where solutions with better fitness values have a greater probability of being chosen.
- Crossover: The crossover operation combines the genetic material (parts of the chromosome) of the two parent solutions to generate new offspring solutions.
- Mutation: The mutation operation incorporates small random changes to an individual solution's chromosome. It aids in introducing new genetic material into the population, preventing the algorithm from being stuck in local optima.

Classical GA Pseudocode

There are various versions of implementing genetic algorithms. We present the pseudocode of the classical method [20]. It is the easiest in terms of implementation and it also contains all the main components of GA.

1. Initialize population: $P = \{x_1, x_2, ..., x_n\}$, where x_i is a potential solution.
2. Evaluate fitness: Calculate the fitness value $f(x_i)$ for each x_i in P.
3. Repeat until a termination condition is met:
   a. Select parents: Choose two individuals from P with a probability proportional to their fitness. Let p_a and p_b be the selected parents.
   b. Crossover : Create a new individual p_o by recombining the genes of p_a and p_b.
   c. Mutate: With a small probability, mutate one or more genes in p_o.
   d. Evaluate fitness: Calculate the fitness value $f(p_o)$ for p_o.
   e. Replace: Replace one of the individuals in P with p_o based on a fitness.
4. Return the best solution(s) found during the process.

2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) [21] is a metaheuristic algorithm that draws inspiration from the social behaviour of fish schools and flocks of birds. Its appeal stems from its ease of use and effectiveness in addressing complicated optimization problems in continuous space. In PSO, a swarm of particles travel iteratively across a search space while employing social interaction and self-experience to navigate towards the best solution. Each particle's position represents a potential solution to the problem, and its movement is governed by both its best solution as well as the best solution discovered until now by the swarm.

Particle Velocity and Movement

At iteration t, the particle i moves in the search space by updating its position x_i^t by adding current velocity v_i^t to its previous position.

$$x_i^t = x_i^{t-1} + v_i^t \tag{1}$$

$$v_i^t = wv_i^{t-1} + c_1 r_1 (pbest_i^{t-1} - x_i^{t-1}) + c_2 r_2 (gbest^{t-1} - x_i^{t-1}) \tag{2}$$

Where $pbest_i^{t-1}$ is the personal best position (solution) achieved so far by the particle i. It is calculated using a fitness function. And $gbest^{t-1}$ is the best position for the entire swarm so far. c_1 and c_2 are the cognitive and social coefficients which are set by the user. r_1 and r_2 are random values generated uniformly between 0 and 1.

There are three components to the velocity of a particle; i) Momentum, ii) Cognitive, and iii) Social.

- Momentum (wv_i^{t-1}) : Weighted previous velocity vector. The weight is also referred to as inertia.
- Cognitive ($c_1 r_1 (pbest_i^{t-1} - x_i^{t-1})$): It is a component in the direction of the current personal best position for the particle from its previous position.
- Social ($c_2 r_2 (gbest^{t-1} - x_i^{t-1})$: It is a component in the direction of the current global best position for the particle from its previous position.

PSO Pseudocode

1. Initialize swarm size, the maximum number of iterations, and other parameters (w, c_1, c_2).
2. Randomly initialize each particle's position (x^0) and velocity (v^0).
3. Evaluate the fitness of each particle.
4. Set the personal best position $(pbest^0)$ and personal best fitness of each particle to its current position and fitness.
5. Determine the global best position $(gbest^0)$ based on the personal best position and fitness of all particles.
6. For each iteration (t):
   a. Update each particle's velocity $(v^t)$and position (x^t) using the Eqs. 1 and 2.
   b. Evaluate the fitness of each particle.
   c. For each particle, update its personal best position and its personal best fitness. Additionally, update the global best position.
7. Return the best solution(s) found during the process.

2.3 Differential Evolution

Differential evolution (DE) [43] is another population-based metaheuristic optimization approach. Similar to GA, DE is also inspired by natural selection, but they differ in the way they mimic it. In DE, an initial population of possible solutions is produced at random. Each individual in the population is represented by a vector of real values, hence individuals are also referred to as vectors. The elements of the vector are often called genes. DE uses the difference between the vectors of two individuals in the population to generate a new candidate solution. It involves the selection, crossover, and mutation of individuals to create new individuals for the next generation. It is important to note that GA and DE have distinct characteristics and applications. DE is a specialized algorithm that uses a specific mechanism to create new individuals, while GA is a more general approach that encompasses a range of methods. DE is often used for continuous optimization problems with few constraints.

In DE, a vector in the current population, also known as the target vector, might get substituted with a trial vector. To create the trial vector corresponding to the target vector, first, a mutant vector is created, which is a linear combination of some randomly picked vectors from the population. The trial vector is then created by performing a crossover operation on the mutant vector and the target vector. The target vector and trial vector are compared and based on the fitness only one of them is passed to the next generation.

Types of Vectors

- Target Vector (x_i): A vector in the population, which may get replaced.
- Mutant Vector (v_i): A vector which is generated by adding the difference between two randomly chosen vectors in the population to a third vector. All the chosen vectors are not the target vector.
- Trial vector (u_i): A vector which is generated by performing a crossover operation between the target vector and the mutant vector.

Let the population at iteration t be $P(t) = \{x_1^t, x_2^t, x_3^t ... x_n^t\}$. Vector x_i^{t+1} for $P(t+1)$ is obtained by first setting x_i^t as the target vector. Then r_1, r_2 and r_3 are three distinct vectors obtained from $P(t)$ which are not the same as the target vector. They are used to create a mutant vector v_i^{t+1} (Eq.3). F is an user-defined constant between $[0, 2]$.

$$v_i^{t+1} = r_1 + F(r_2 - r_3) \tag{3}$$

Trial vector u_i^t is obtained by performing a crossover operation between the mutant and the target vectors. The jth element of u_i^{t+1} is obtained by first generating a random value r_j using uniform distribution between 0 and 1. p_c is the pre-defined crossover probability. Either the jth element of the target vector or the mutant vector is selected according to Eq. 4.

$$u_{ij}^{t+1} = \begin{cases} x_{ij}^{t} & if\, r_j > p_c \\ v_{ij}^{t+1} & otherwise \end{cases} \tag{4}$$

Ultimately, depending on fitness, either the target vector is retained or the trial vector replaces the target vector in $P(t+1)$.

DE Pseudocode

1. Initialize population $P(0)$ with random vectors (solutions) and evaluate their fitness
2. While the stopping criterion is not met, for each iteration (t):
   a. For each individual x_i^t in the population, $P(t)$ do:
      i. Select three distinct individuals r_1, r_2, and r_3 from $P(t)$, not including x_i^t
      ii. Generate a mutant vector v_i^{t+1}
      iii. Generate a trial vector u_i^{t+1} by applying crossover between v_i^{t+1} and x_i^t
      iv. Evaluate the fitness of u_i^{t+1}
      v. If the fitness of u_i^{t+1} is better than x_i^t, then $x_i^{t+1} = u_i^{t+1}$ else $x_i^{t+1} = x_i^t$
3. Return the best solution(s) from $P(T)$, T is the total number of iterations.

2.4 Multi-Objective Optimization using Non-dominated Sorting Genetic Algorithm II

The practice of simultaneously optimising many objectives is known as multi-objective optimisation. Unlike traditional optimization problems that focus on optimizing a single objective function, real-world situations often entail the need to optimize multiple objectives concurrently. For instance, minimizing costs while maximizing performance is a common example of conflicting objectives that need to be optimized together.

Multi-objective optimization techniques aim to find a set of solutions that represent a trade-off between conflicting objectives. These solutions are called Pareto-optimal solutions and represent the best possible compromise between the objectives. The Pareto front is the set of all Pareto-optimal solutions. Pareto optimal solutions are solutions which cannot be dominated by any other solution. A solution is said to be dominating any other solution if, for all the objectives, the dominant solution is performing better or equal to the other solution. Refer to Figure 2 for an illustration.

One way to simplify a multi-objective optimization task is to transform it into a single-objective optimization problem by computing a weighted sum of all the objectives. The problem is that it may not be straightforward to assign weights as choosing weights for the different objectives is often subjective and arbitrary. It is difficult to justify why one weight should be given more importance than another, and different weights can lead to different optimal solutions. Moreover, multi-objective

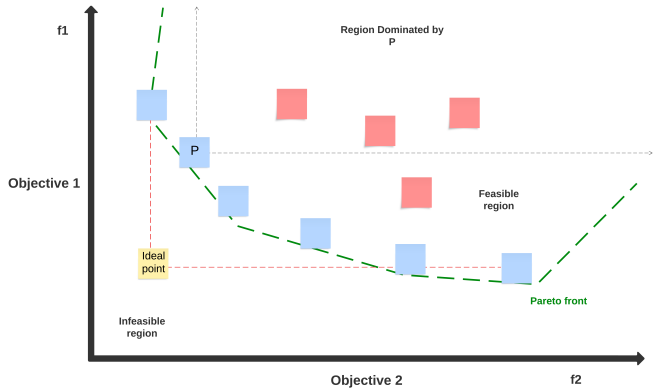

Fig. 2 Pareto optimal front for two objective functions. The region above the green dotted line is the feasible region. The goal is to minimize both objectives. Blue points indicate the non-dominated solutions which are also known as Pareto-optimal solutions. The yellow point indicates an ideal point which minimizes both objectives but lies in the infeasible region. Red points indicate sub-optimal solutions. Note that the points in the figure are in the "objective space" and don't represent their spatial arrangement. An individual is mapped to a vector consisting of the values of the objectives $x \rightarrow (f_1(x), f_2(x))$.

optimization is often concerned with finding Pareto-optimal solutions which are as diverse as possible. Evolutionary algorithms are well-suited for multi-objective optimization tasks because they are population-based and can explore multiple regions in the solution space simultaneously.

NSGA-II (Non-dominated Sorting Genetic Algorithm II) [10] is a popular evolutionary multi-objective optimization algorithm. The NSGA-II algorithm is based on the genetic algorithm framework and employs several novel strategies to address some of the limitations of its predecessor NSGA. NSGA-II is able to efficiently handle multiple objectives simultaneously. The algorithm operates by generating and evolving a population of candidate solutions (chromosomes), where each chromosome represents a potential solution to the optimization problem. NSGA-II uses a combination of non-dominated sorting and crowding distance measurement to rank the chromosomes based on their fitness values. The non-dominated sorting method is used to identify the non-dominated solutions, while the crowding distance measure is utilized to maintain diversity within the population.

Non-dominated sorting involves categorizing solutions into various levels or fronts.s, where each level contains solutions that are dominated by the solutions in the previous levels. The first front contains solutions that are not dominated by any other solutions in the population, while the second front contains solutions that are not dominated by the solutions not in the first front, and so on. The algorithm continues to sort the solutions until all the solutions are classified into different fronts. Crowding distance measures the degree of crowding around each solution in the population. Solutions with higher crowding distances are preferred over those with lower crowding distances. The solutions are first ranked based on their non-

dominating front. A solution in the 1st front is superior to a solution in the 2nd front. For ranking solutions within the same front, crowding distance is used.

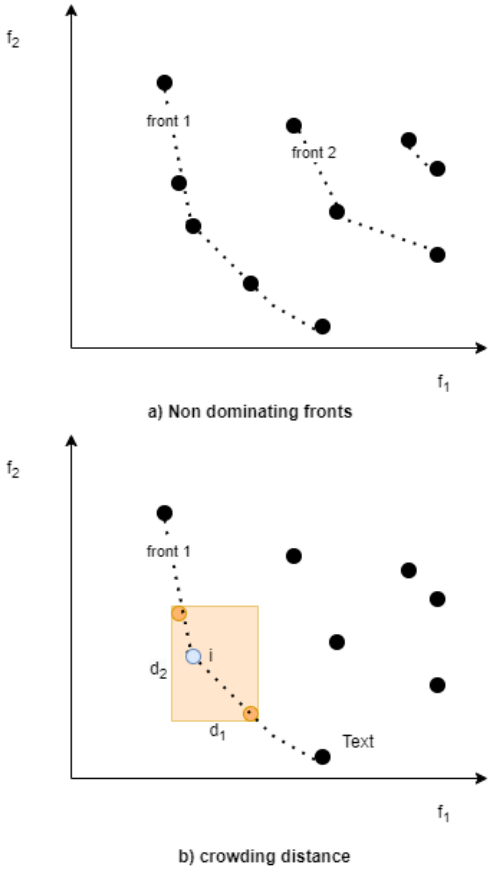

Fig. 3 For the multiobjective task involving minimization of both the objectives, f_1 and f_2, the figures illustrate the concept of non-dominating fronts (a) and crowding distance (b). Crowding distance depends on the sides of the hyper-cuboid formed by the nearest solutions belonging to the same front in the objective space, $cdist(i) = \frac{d1}{f_1^{max} - f_1^{min}} + \frac{d2}{f_2^{max} - f_2^{min}}$.

The Main Loop

NSGA-II is a genetic algorithm which specializes in solving a multi-objective optimization task. The algorithm starts by generating a random population P_0 of size N.Non-dominant sorting and crowding distance are used to rank all solutions within the population. The fitness assigned to the solutions is equivalent to the level of the non-dominating front to which they belong. Hence, fitness is to be minimised. Binary tournament selection method is employed for the selection process. After performing crossover and mutation, the offspring population Q_0 is generated. At every iteration t, by incorporating an elitism mechanism S, the combined population

$R_t = Q_t \cup P_t$ of size $2N$ is reduced to size N which is the population at iteration $t+1$, $P_{t+1} = S(R_t, N)$. Figure 4 illustrates the elitism mechanism.

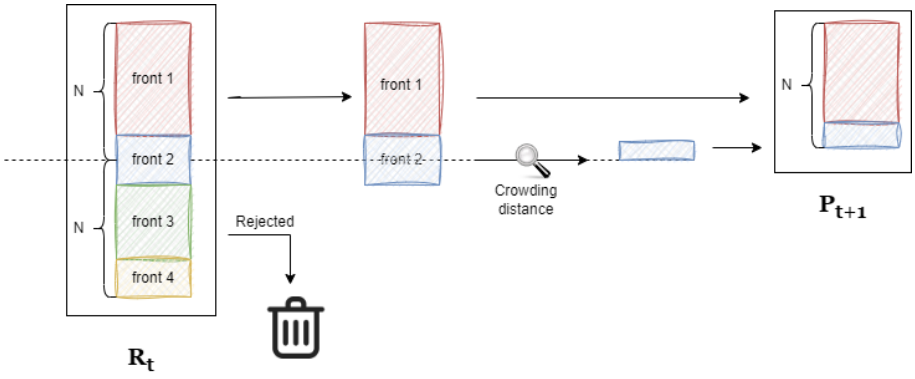

Fig. 4 Elitism in NSGA - II. The combined population R_t is first sorted into non-dominating fronts. Starting from the lower fronts, if the size of a front is less than the available size, it is added to P_{t+1}. After adding some fronts in order, if the size of the latest front to be added is less than the available size, then the front is sorted with the help of the crowding distance. The sorted front is clipped so that it could be fitted in the available space.

2.5 Genetic Programming

Genetic Programming (GP) [22] uses a technique inspired by natural evolution to create computer programs by automating the program development process, reducing the need for human intervention. It uses a population of computer programs, represented as a string of symbols, a tree or code, and evolves them over time through the application of genetic operators such as mutation and crossover. The fitness of each program is evaluated based on how well it performs on a set of test cases. GP has the potential to be highly beneficial in creating models for tasks involving regression or classification. For example, the function $f(x, y) = yx(2x + y)$ is represented as a binary tree in Figure 5. The goal of genetic programming is to start with a random population of trees and then through the application of genetic operators evolve the population of trees such that their fitness is improved. The fitness of a tree measures the accuracy of the function represented by it, for example, the mean square error over a test dataset.

A simple GP has the following basic components.

1. A mechanism to convert the population individual to a corresponding executable code.
2. A mechanism to measure the fitness of an individual.
3. Terminal set T containing all the input variables and constants.

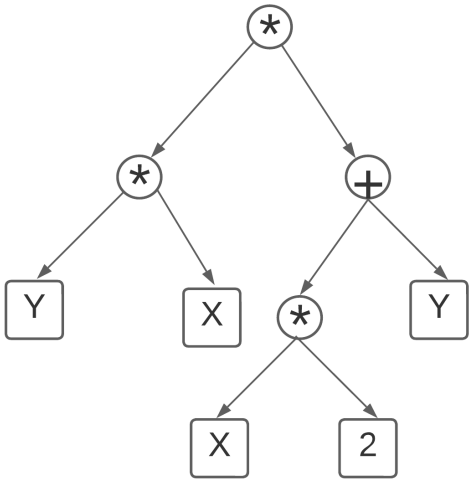

Fig. 5 Binary tree representing the expression: $f(x, y) = yx(2x + y)$. The terminal nodes (square-shaped) contain the model inputs and constants. The internal node represents functions and operations.

4. Function set F containing all the simple functions and operations applicable on an internal node. For example, $F = \{+, -, *, /, max\}$
5. Control parameters similar to genetic algorithms which include population size, mutation rate, and crossover rate, among others.

Genetic operations

1. Selection: After the fitness of the population has been evaluated, the algorithm selects fitter individuals as parents with probability proportional to the fitness.
2. Reproduction: Some of the fittest individuals are simply carried on to the next generation.
3. Crossover: It involves randomly selecting nodes for a crossover between the two parent trees. The subtrees rooted at the selected nodes (below) are exchanged between the two parents and hence two offspring are created.
4. Mutation: It involves randomly selecting a node for mutation. A randomly generated tree replaces the subtree rooted at the selected node. See Figure 6 for an illustration.

Operationally GP is the same as GA, it begins with a random population of trees and applies genetic operations at each iteration to improve the population's fitness through incremental updates. The final population generated after meeting certain termination criteria is used to determine the fittest program tree(s).

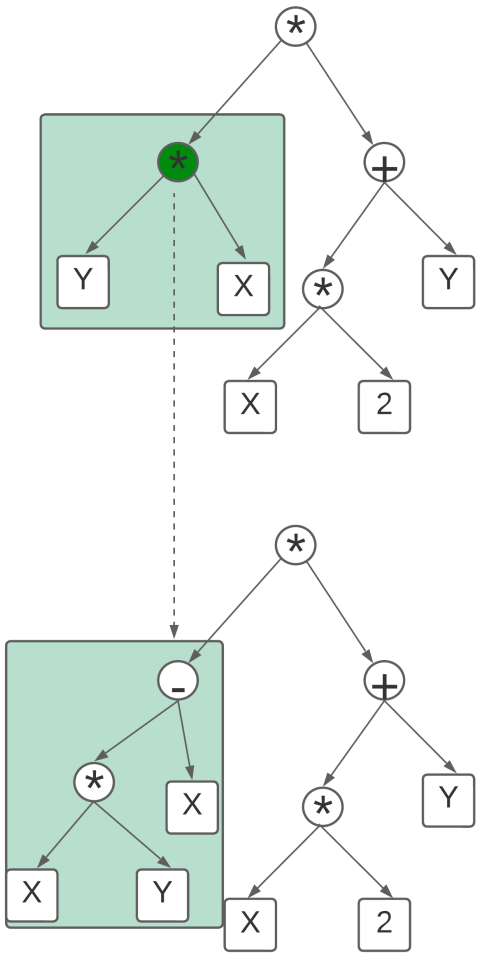

Fig. 6 A binary tree representing the expression: $f(x, y) = yx(2x + y)$ is mutated to a tree given just below. It is done by first selecting a node in the tree (green coloured). Then the subtree rooted at the selected node is replaced by a randomly generated tree. Essentially replacing yx in the expression by some $g(x, y)$.

3 Application of Metaheuristic and Evolutionary Algorithms in XAI

Explainability could be viewed through the lens of optimization. For example, balancing the accuracy and interpretability of AI systems is itself a multi-objective optimization task, as there is a trade-off between these two factors. Achieving high accuracy often requires more complex models that can be difficult to interpret, while achieving high interpretability may require sacrificing some level of accuracy.

In addition to balancing accuracy and interpretability, post-hoc techniques such as searching for counterfactual explanations can be viewed as a constrained optimization task. These techniques aim to find a set of inputs that, when modified, would result

in a different output from the AI system. This requires searching for input values that satisfy certain constraints, such as similarity to the original input, belonging to the data distribution and giving desired outputs from the model.

Another popular XAI technique of fitting local surrogate models involves generating samples that are close to the point that needs to be explained and belong to the data distribution. These objectives can be formulated as optimization problems.

Evolutionary and metaheuristic algorithms are well-suited to solving optimization problems in XAI because they are flexible in terms of problem definition. These algorithms can be applied to a wide range of optimization tasks, including multi-objective optimization and constrained optimization. Furthermore, they can be used to optimize complex, non-linear functions that are difficult to solve using traditional optimization techniques.

Overall, viewing XAI as an optimization task provides a useful framework for developing new methods and algorithms for achieving better interpretability in AI systems. By applying evolutionary and metaheuristic algorithms to these optimization problems, it is possible to develop more effective and efficient techniques for achieving explainability in AI systems. In this section, we will look at some of the existing implementations.

3.1 Counterfactual Explanations

The first formalization of counterfactual explanations in machine learning can be traced back to the work of [49]. In this paper, the authors proposed using counterfactual explanations as a way to comply with the transparency and accountability requirements of the European Union's General Data Protection Regulation (GDPR).

A counterfactual explanation provides an explanation by showing how changing a particular set of input features would have resulted in a different prediction by a complex machine learning model. Counterfactual explanations are beneficial in situations where the model's output needs to be changed or improved. By generating a counterfactual explanation, users can see how performing minimal changes to specific input features can lead to a more desirable outcome. For example, after getting a loan application rejected by a bank, one might wonder what went wrong, more specifically why the application fell short. In this regard, the bank could return a counterfactual, which could be in the form of "if the personal income had been more by x amount and the current debt was lower by y amount the loan may have been approved". By doing so, it not only properly justifies its decision-making but also provides an actionable recourse. It is to be noted that for some features, the change suggested may not be actionable. For example, someone cant age in the reverse direction nor can change their race. One way to deal with it is to limit the search space of the counterfactual to what changes of feature values are realistically possible. Nevertheless, such explanations may help uncover hidden biases in the system.

Evolutionary and metaheuristic algorithms are advantageous for finding counterfactuals because they are highly effective at navigating and searching through large solution spaces, this helps in producing accurate and diversified counterfactual explanations. Additionally, they can be used to optimize multiple objectives simultaneously [8]. However, It should be noted that an algorithm's effectiveness in finding optimal counterfactuals depends heavily on the reliability of the objective function. If the objective function is not well-defined or does not reflect the desired outcome precisely, the algorithm may converge on suboptimal solutions. For a classification model, the objective function could be as simple as Eq. 5.

$$\begin{aligned} &\min_{c} d(x, c) \\ &s.t. f(x) \neq f(c) \end{aligned} \tag{5}$$

Where x is the original instance, $d(x, c)$ is the distance between the counterfactual c and x, and f is the black-box model.

These methods are also model agnostic, which means they can work on a wide range of model types without the requirement of model parameters as inputs. This is particularly useful when a company is reluctant to share the model's internal processes due to privacy concerns but still needs to comply with regulations.

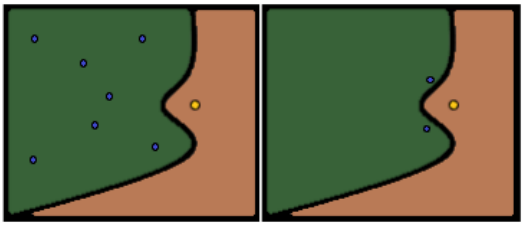

Fig. 7 Any population-based algorithm begins with an initial population of sub-optimal solutions (left). The population is improved over several generations and finally, the solutions which have high fitness are returned as counterfactuals (right).

Before discussing the implementation details of different counterfactual generation algorithms, it is important to understand the concept of distance, which plays a crucial role in counterfactual generation.

3.1.1 Data Types and Distance Measures

A key feature of a counterfactual is that it should be as close to the input instance as possible. A proper distance measure is crucial for generating counterfactuals because it determines how close the generated counterfactuals are to the original instance. The distance measure is used to define the closeness between the original instance and a counterfactual instance in the feature space. Defining a good distance measure may not be straightforward as real-world data comes in different forms.

Tabular Data

First, let's examine the scenario where all the characteristics are numeric. In this case, each data point is situated in an R^n space, where n is the total number of features. As distance measures, two extremely popular options are the L_1 (mean absolute error or MAE) distance and L_2 (mean squared error or MSE) distance.

$$MAE(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{6}$$

$$MSE(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2 \tag{7}$$

Using these distances may not be appropriate as the features may not be of equal scale, and this may lead to some features dominating others. The right approach is to take a weighted distance.

One possible definition for the distance function d could be the weighted L_1 distance, weighted by the inverse of the Median Absolute Deviation (MAD) [49] of each feature taken over some data points P, typically the training dataset.

$$d(x, y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{w_i} \tag{8}$$

Where

$$w_i = median_{j \in P}(x_{j,i} - median_{l \in P}(x_{l,i})) \tag{9}$$

In many real-world scenarios, data is not purely numeric, but rather a mix of different data types. This is particularly true in the banking industry, where data can include not only numeric values such as account balances, transaction amounts, and interest rates, but also non-numeric data such as customer names, addresses, and account types. A proper distance measure should be defined in order to deal with mixed data types.

One such distance measure that can accommodate both numerical and categorical features is the Gower distance [8]. The Gower distance between x and y with mixed attributes is the following.

$$d(x, y) = \frac{1}{n} \sum_{i=1}^{n} \delta(x_i, y_i) \tag{10}$$

Where

$$\delta(x_i, y_i) = \begin{cases} \frac{1}{R_i}|x_i - y_i| & \textit{if numerical} \\ \mathbf{1}_{x_i \neq y_i} & \textit{if categorical} \end{cases} \tag{11}$$

R_i is the observed range for numerical feature i. Typically calculated on the training dataset.

3.1.2 Counterfactuals using Genetic Algorithms

Implementing a genetic algorithm to produce counterfactuals includes randomly picking values for each variable to build an initial population of potential counterfactuals. The fitness function would then evaluate each solution in the population in terms of how similar it is to the original input, and whether it is delivering the intended prediction from the black-box model. The algorithm would then employ genetic operations such as selection, crossover, and mutation to generate a new population of potential counterfactuals. This process is repeated until a condition for termination is reached or a predefined number of iterations is reached.

Counterfactual Explanations for Robustness, Transparency, Interpretability, and Fairness of Artificial Intelligence (CERTIFAI) [41] uses a custom genetic algorithm to generate counterfactuals for a given instance. The black-box model is used for a classification task. The genetic algorithm is used to get a counterfactual belonging to a different class than the original instance. An open-source implementation is present at https://github.com/Ighina/CERTIFAI. The algorithm's key components are outlined below.

Choice of the Distance Measure

The authors have defined two distance measures, one for mixed attributes tabular data (Eq. 12) and another for image data (Eq. 13). The $NormAbs$ is the weighted L_1 distance defined in Eq. 8. $SimpMat$, a simple matching distance is selected to measure the distance between categorical attributes.

$$d(x, c) = NormAbs(x, c) + SimpMat(x, c) \tag{12}$$

$$d(x, c) = \frac{1}{SSIM(x, c)} \tag{13}$$

For images, $SSIM$ (Structural Similarity Index Measure) is used to define the distance.

Custom GA

The optimization objective used is the simple one defined earlier (Eq.5). The fitness function is defined as the reciprocal of the distance from the selected instance x. The closer the point to x, the higher its fitness (Eq. 14).

$$fitness(c) = \frac{1}{d(x, c)} \tag{14}$$

Since we are interested in the points which are close but also belong to a different class after every iteration the population is filtered such that only solutions that

satisfy the belonging to the different class constraint remain and then fitness is calculated. The black-box model is used to determine the class of a solution. The initial population is generated such that all the solutions belong to a class which isn't the same as the class of the instance x. Further constraints could be imposed on the search space to make the counterfactual realistically attainable, for example, the race of a person cannot be changed. This is done by passing bounds for the features as parameters. Infeasible solutions are removed after every iteration. The mutation operation is done by slightly perturbing a few of the feature values. And crossover is done by simply interchanging some feature values between two solutions.

Experimental Simulations

In CERTIFAI, the authors showcase multiple use cases of counterfactual generation. The experiments are done on the UCI adult dataset and the Pima Indian diabetes dataset, along with some standard datasets such as iris and breast cancer. Counterfactuals generated are used for the following.

1. Robustness: Measuring the robustness of the machine learning model against any adversarial attack. This is done by treating counterfactuals as adversarial examples and measuring the expected distance between input instances and their corresponding counterfactuals.
2. Fairness: The concept of burden is defined as the expected distance between input instances belonging to a particular group having an undesired outcome, and their corresponding counterfactuals. It reflects the degree of change required for instances belonging to the different partitions of the population, to change the undesired outcome. The burden is calculated for different ethnic groups present in the UCI adult dataset and based on the values, it can be seen that the black-box model is biased towards whites and unfair towards the black ethnic group.
3. Explainability: For explaining instances from both the UCI adult and the Pima Indian datasets, multiple counterfactuals are generated with or without constraints. The counterfactuals are also used for identifying the importance assigned to the features by the black-box model. The importance of a feature depends upon the number of times that feature is changed in order to generate counterfactuals across the dataset.

The study did not directly compare the new technique with existing counterfactual generation methods that were available at the time of the study.

3.1.3 Counterfactuals using Particle Swarm Optimization and Differential Evolution

In the work of Anderson et al. [1], DE and PSO alternatives are suggested as a replacement for simple GA. For an optimization task (counterfactual generations) in a continuous space, a simple GA algorithm can struggle to con-

verge [36, 17]. On the other hand, PSO and DE are often used in optimization tasks in continuous spaces. An open-source implementation is present at https://github.com/HaydenAndersen/ECCounterfactuals. The key components of both algorithms are outlined below.

Choice of the Distance Measure

The algorithms are proposed to work on continuous spaces, which means all features are numerical. That's why the L_1 norm (Eq. 6) is preferred as the choice of the distance measure. However, in their work, the data is assumed to be scaled beforehand. If not, it may be preferable to use weighted distance instead (Eq. 8). The authors generated the starting population using a $U(0, 1)$ uniform distribution. This is because the data is assumed to be scaled, and all the features are lying between 0 and 1.

Fitness Function

The choice of fitness function for both algorithms is the same, it is exactly the distance from the selected instance x. Since the distance is directly taken as the fitness function, the lower the fitness, the better the generated candidate. If the prediction made by the black-box model for a candidate is not a desired one, for example, in classification, if the candidate is predicted to belong to the same class as x. Then the fitness assigned is ∞.

Using PSO

The algorithm is mostly the same as the one discussed in section 2. The initial population is generated randomly. Each particle is assigned fitness as ∞, as lower fitness is better. After every iteration, the personal best position for a particle is the one with the lowest fitness attained by it so far. The parameters take the standard values, these are $w = 0.7298$, $c_1 = 1.4962$, and $c_2 = 1.4962$ [5].

Using DE

Similar to PSO, the DE algorithm is almost the same as discussed in section 2. Just as in PSO, the initial population is generated randomly and each vector is assigned fitness as ∞. Instead of taking a fixed value of $F \in [0, 2]$, it is drawn uniformly at every mutation from $U(F_{min}, F_{max})$, the range (F_{min}, F_{max}) is user-defined. The default parameter values are used for DE as given in the SciPy Python library.

Experimental Simulations

The authors have performed a qualitative study as well as a comparative study. The datasets used for the studies have all continuous attributes. The proposed algorithms are compared with CERTIFAI. The comparison is done by measuring the mean number of features modified while generating counterfactuals, and also the mean of L1 and L2 distances from the original points to the counterfactuals. The datasets used for the study have all continuous attributes. The idea is that if the compared metrics are lower then the quality of the counterfactuals generated is higher. It is to be noted that this won't account for the diversity observed in the generated counterfactuals. Qualitative analysis is done on the Penguins dataset to verify if the changes suggested by the counterfactuals are reasonable.

3.1.4 Multi Objective Counterfactuals using NSGA-II

Dandl et al. [8] propose the search of counterfactuals as a multi-objective search. Their work formalizes the concept of using multi-objective optimization for counterfactual generation (MOC). For the instance to be explained, MOC returns a diverse collection of counterfactuals, which corresponds to the Pareto front of a four-objective optimization problem.

Let $f : \mathcal{X} \rightarrow R$ be the black-box model mapping the feature space $\mathcal{X}$ to a real-valued output and x is the selected instance for which counterfactuals are to be generated. $Y \subset R$ is the subset of the desired outcome. It could be a range of values such that $f(x) \notin Y$. For the classification task, $f(x)$ could be the probability returned by the black-box model for a particular class. A counterfactual c has certain desired properties.

1. $f(c)$ should be as close as possible to the desired set Y
2. c is very close to x in $\mathcal{X}$ space
3. c differs from x in only few features
4. c is lying within the data distribution in $\mathcal{X}$

These properties are expressed in terms of a multi-objective optimization task.

$$\min_c O(c) := \min_c(O_1(f(c), Y), O_2(c, x), O_3(c, x), O_4(c, X^{obs})) \tag{15}$$

Where

$$O_1(f(c), Y) = \begin{cases} 0 & iff(c) \in Y \\ \inf\limits_{y \in Y} |f(c) - y| & otherwise \end{cases} \tag{16}$$

$$O_2(c, x) = Gower\ distance(x, c) \tag{17}$$

$$O_3(c, x) = ||c - x||_0 \tag{18}$$

$$O_4(c, X^{obs}) = \sum_{j=1}^{k} w^{[j]} Gower\ distance(x^{[j]}, c) \tag{19}$$

The objective functions O_1, O_2, O_3 and O_4 reflect the desired properties of a counterfactual. Gower distance (Eq. 10) measures the distance between a solution c and x. The objective O_3 counts the number of features for which x and c don't have the same values. It is done by counting the number of non-zero elements of $x - c$. The objective O_4 involves a collection of observations X^{obs}, and the training data used to train the black-box model is the preferred choice $X^{train} = X^{obs}$. It measures the weighted distance between c and its first k neighbours in X^{obs}.

For searching counterfactuals with multiple objectives, a slightly modified version of NSGA - II is implemented. Since the solution could be a mixture of continuous and discrete attributes, a mixed integer strategy [25] is used for the genetic algorithm. The crowding distance is also modified to take into account not only the distance in the objective space but also the distance in the solution space. An open-source implementation is present at https://github.com/dandls/moc.

Experimental Simulations

The authors demonstrate the viability of developing a diverse range of counterfactuals for cases in the German credit dataset. A comparative study is also done with state-of-the-art counterfactual generation tools such as DiCE [31], Recource [48] and Tweaking [44]. This is done on the datasets accessed from the OpenML platform. The metric used for comparison is the coverage rate [50], which is the measure of the relative frequency of the counterfactuals generated by a particular method dominated by MOC. If the coverage rate is 1 then there exists at least one counterfactual generated by MOC which dominates it.

3.1.5 Additional Works

It can be observed from Table 1 that further research has been conducted on generating counterfactuals utilizing evolutionary and metaheuristic algorithms. The table also reflects the significance of regarding counterfactual generation as a multi-objective optimization problem. The objectives are associated with some of the desirable qualities of counterfactuals [32] such as i) correctness in outcome (validity), ii) closeness to the original instance (proximity), iii) less number of features changed (sparsity), iv) within feasibility constraints (actionability), v) several alternatives for recourse (diversity) and vi) belonging to the data distribution.

Table 1 Counterfactuals using Evolutionary and Metaheuristic algorithms

Paper	Technique Used	Open Source Implementation
GeCo [40]	Custom Genetic Algorithm	https://github.com/mjschleich/GeCo.jl
GIC [24]	Genetic Algorithm with local search	https://github.com/michael-lash/Inverse_Classification
Plausible Counterfactuals [4]	Speed-constrained Multi-Objective Particle Swarm Optimization	None Found
CARE [37]	Multi-Objective Optimization with NSGA - III	https://github.com/peymanrasouli/CARE
Model-Agnostic Counterfactual Explanations in Credit Scoring [9]	Custom Genetic Algorithm	None Found
MOOD [30]	Multi-Objective Optimization with U-NSGA-III	https://github.com/wmonteiro-ai/xmoai
ProCE [11]	Multi Objective Optimization with NSGA-II	https://github.com/tridungduong16/multiobj-scm-cf
PermuteAttack [16]	Custom Genetic Algorithm	https://github.com/masoudhashemi/PermuteAttack

3.2 Local Surrogate Modelling

Local surrogate modelling involves creating an interpretable model that approximates the behaviour of the black-box model in a specific region of input space. The resulting surrogate model can be used to provide insights into how the black-box model works in that region of input space. exists an inherent compromise between the interpretability and the performance of a model. A simple interpretable model may not perform well on the entire test dataset when compared to the more intricate black-box model. Therefore, the objective is not to mimic the complex black-box model entirely, instead, the idea is to use a simple interpretable model to mimic the black-box model in a locality. This is analogous to the idea of Taylor series expansion in mathematics which suggests that a real-valued function could be approximated to a lower-order polynomial around a small neighbourhood. Local surrogate modelling has gained a lot of popularity in XAI. LIME [38] is a very popular technique which falls under this category. All these techniques have a generic framework which is the following.

1. Select an instance of interest from the dataset for which the prediction has to be explained.
2. Generate a neighbourhood of local data points around the instance of interest by perturbing the features of the instance within a certain range.
3. Train a simpler, interpretable model, such as a decision tree or linear regression, on the local data points and their corresponding black-box model predictions as targets.

4. Examine the surrogate model to get a more understandable view of the black-box model's prediction mechanism in the vicinity of the instance of interest.

Local Neighbourhood generation

The quality of explanations produced by local surrogate models is highly influenced by the quality of the local neighbourhood. If the generated data is biased or not representative of the population, the explanations may not accurately reflect the behaviour of the black-box model on the population instances. LIME uses independent Gaussian distributions to generate a sample. It has been shown that explanations using such a simple mechanism of generating a local sample are prone to adversarial attacks [42].

Another challenge with local datasets is that they may not contain enough information to capture the complexity of the black-box model accurately. This can result in the surrogate model being overly simplistic, and the explanations generated by the model may not provide a complete understanding of the behaviour of the black-box model.

3.2.1 Local Rule-Based Explanations

The quality of a local explanation relies heavily on the quality of the local neighbourhood generated around the instance of interest. LORE [15] uses a genetic algorithm for generating a good neighbourhood around the instance to be explained for a binary classification task. A decision tree model is fitted and a single rule is obtained along with some counterfactual rules extracted from the tree structure which are returned as an explanation. The LORE algorithm investigates the black-box model's decision boundaries around the instance for which the prediction has to be explained. The idea is the decision boundary is best captured by not only the region near the point of interest but also the region around the nearest counterfactual and by sampling points in these regions, the generated neighbourhood has points belonging to both classes. The genetic algorithm is used only for the neighbourhood generation. An open-source implementation is present at https://github.com/riccotti/LORE. The algorithm's key components are outlined below.

Local Neighbourhood Generation using GA

The genetic algorithm uses two custom fitness measures $fitness^x_=$ and $fitness^x_{\neq}$, for generating the set of points belonging to the same and the different class denoted by $Z_=$ and $Z_{\neq}$ respectively. $Z = Z_= \cup Z_{\neq}$ is the generated neighbourhood on which a decision tree model is fitted.

$$fitness^x_=(z) = 1_{f(x)=f(z)} + (1 - d_x(z)) - 1_{x=z} \tag{20}$$

$$fitness^{x}_{\neq}(z) = 1_{f(x)\neq f(z)} + (1 - d_x(z)) - 1_{x=z} \tag{21}$$

Where $x \in \mathcal{X}$ is the instance of interest. $d : \mathcal{X} \rightarrow [0, 1]$ is a function which measures the distance from x. $f(z)$ is the black-box model prediction. Assume that there are m features overall, and h of which are categorical features. Then the distance is a weighted sum of the distance between the categorical and the numerical features.

$$d_x(z) = \frac{h}{m} SimpMat(x, z) + \frac{m - h}{m} NormEuclid(x, z) \tag{22}$$

The genetic algorithm is used twice, for generating $Z_{=}$ and $Z_{\neq}$. The parameters of the algorithm are $x, fitness, N, p_c$ and p_m. First, the population of size N is initialized $P_0 = \{x_i | x_i = x; \forall i = 1, 2...n\}$. All the values are set to x. The generation counter i is set to 0. While $i < G$, P_{i+1} is updated from P_i in the following way.

1. Perform the selection operation and select a portion of the fittest sub-population of P_i as parents.
2. Perform a 2-point crossover operation to generate offsprings using crossover probability p_c. Update the population as the union of the parents and the offsprings.
3. Perform mutation on a portion of the population using probability p_m. Update the population as the union of the mutated and unmutated individuals.
4. Based on fitness, select the top N individuals as the new population P_{i+1}.

Explanation using Tree

After the neighbourhood generation using GA, a decision tree model is trained using the neighbourhood data Z and the target labels $f(Z)$ as the black-box model predictions on it. LORE returns an explanation $e = \{r, \Phi\}$. Where r is a single rule and Φ is a collection of counterfactual rules. The conjunction of split conditions along a path of the local decision tree which is satisfied by x forms r. See Figure 8 for an illustration.

The number of split conditions that are not fulfilled by x is tallied for each possible counterfactual rule. For a counterfactual rule, if it is below a certain threshold, then the rule is added to Φ.

Experimental Simulations

The authors have performed experiments are on three datasets containing both categorical and numerical attributes. These are adult, compas and german. All the datasets represent attributes of a person. The datasets are used to perform binary classification. After performing a train-test split, a black-box model (b) is fitted and the quality of the mimicking the black-box model locally of LORE is assessed.

In order to fit a decision tree (c) around an instance x, a train set Z is generated. This set Z is used to measure 5 evaluation metrics. These are i) fidelity: measured

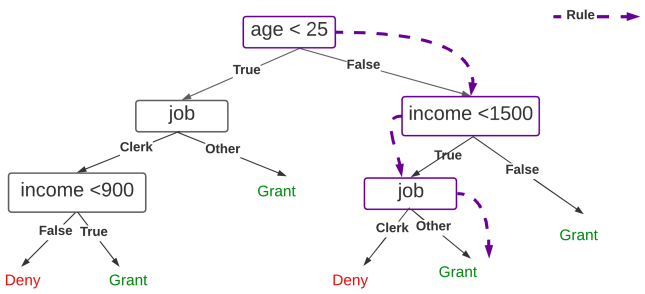

Fig. 8 For x = {age: 27, income: 1200, job: other}, the purple path shows the single rule for the explanation of the instance, r = {age $\geq$ 25, income $\leq$ 1500, job: other} $\rightarrow$ "Grant". All the paths leading to the class "Deny" are candidates for the counterfactual rules.

by comparing the predictions of b and c over Z. ii)l-fidelity: The subset of Z which satisfy the single rule generated using LORE is used for calculating fidelity. iii) cl-fidelity: The subset of Z which satisfies the counterfactual rules generated using LORE is used for calculating fidelity. iv)hit: It compares $b(x)$ and $c(x)$ directly. If $b(x) = c(x)$, then 1 else 0. v) c-hit: It measures the hit of one instance which is derived from the counterfactual rules. The average value is measured for all the evaluation metrics over the test dataset.

A comparison study is also done by using these metrics. LIME and ANCHOR [39] are the two state-of-the-art surrogate modelling techniques used for the study.

3.2.2 Genetic Programming Explainer

LIME operates under the assumption that for a complex black-box model, a linear model serves as a reliable local approximation. To explain a model prediction for an instance, a linear model is fitted on a weighted neighbourhood around the instance and the coefficients of the model are returned as the feature importance values. In some cases, this assumption could lead to a considerable reduction in the precision of the local model's ability to replicate the complex black-box model's behaviour in the surrounding area of the instance of interest.

Genetic Programming Explainer (GPX) [14] uses genetic programming to generate a more accurate interpretable mimicking model. GPX assumes that all the features are numeric. A candidate model is represented as a binary tree. For more details regarding GP refer to section 2. The model generated is not bounded by linearity but at the same time has a closed mathematical expression. If the function set of GP contains only simple arithmetic operations and differentiable functions, then it allows the local surrogate model to be differentiable, and the partial derivative of a feature could be calculated to study the sensitivity of the black-box model with respect to the feature. Let $f_b : R^n \rightarrow R$ be the black-box model, and x is the instance for which the prediction $f_b(x)$ has to be explained. GPX returns a local surrogate model f^* which

is easy to interpret and also mimics the model f_b in the neighbourhood around x. An open-source implementation is present at https://github.com/leauferreira/GpX. The algorithm's key components are outlined below.

Local Neighbourhood Generation

Since all the features are numerical, the local neighbourhood is generated by using a multivariate Gaussian distribution centred at x. It assumes all the features are independent. The covariance matrix is $\Sigma = \sigma I_n$, where σ is calculated over the training data.

Objective function

The objective of GPX is to generate a function f^* such that the distance between $f^*(N(x))$ and $f_b(N(x))$ is minimum.

$$f^* = \arg\min_f d(f(N(x)), f_b(N(x))) \tag{23}$$

Where d could be L_2 norm (Eq. 7). $N(x)$ is the generated neighbourhood of size H around the point x. $f(N(x)) = [f(s_1), f(s_2)...f(s_H)]^\top$ such that $s_i \in N(x)$ $\forall i \in \{1, 2, 3..H\}$.

Interpretability

Given a neighbourhood around an instance, GPX creates a local surrogate model which mimics the black-box model in the neighbourhood. The surrogate model is a simple non-linear expression of features. The complexity could be controlled by limiting the depth of the generated tree. The interpretability comes from the fact that the local surrogate model is a simple non-linear, differentiable mathematical expression.

Experimental Simulations

The authors have used twenty different datasets for performing experiments. These datasets are extracted from popular repositories such as the UCI Machine Learning repository, OpenML, Kaggle and scikit-learn. The experiments are done in two parts, i) A comparative study is done with LIME and Locally Generated Decision Trees to measure the mimicking capability of the surrogate models. ii) A case study is done to interpret a Random Forest Regressor on the Boston and Diabetes dataset using GPX.

For the comparative study, the accuracy (classification task) or the mean square error (regression task) is used. For a particular instance x, a neighbourhood is generated around x then the local model is trained. The accuracy or the mean square error of the local model on the generated neighbourhood is evaluated by taking the black-box model predictions as the target values. This is performed on all the instances in the test dataset and the average over all the test instances is returned.

3.3 Transparent Models

Transparent models are those that are easy to understand and interpret, and their decision-making process is clear and explicit, making them useful in contexts where understanding the model's decision-making process is important.

3.3.1 Decision Tree

Decision trees [35] are constructed using a set of simple rules that help to partition the data into smaller subsets. At each step of the decision tree construction, the algorithm selects the best feature or attribute that divides the data into subsets. This is done by evaluating various criteria such as information gain. Once the best feature is selected, a value is chosen to split the data into two or more subsets. The process is repeated recursively until a stopping criterion is satisfied, which could be a maximum tree depth or a minimum number of training examples in each leaf node. The leaves are assigned a class label or a regression value, depending on the task.

Because decision trees are constructed using a series of simple rules that are based on the values of the input features, they are easy to interpret and explain to humans. It is possible to visualize a decision tree and see the decisions made at each node, which makes it clear how the model is making its predictions.

A lot of work has been done on applying evolutionary algorithms for decision tree generation. While any optimization technique could be implemented to tune the hyper-parameters of the tree, our interest lies in algorithms that are specifically used in the tree-building process. The primary motivation for adopting such algorithms is that the traditional tree-building algorithms are greedy. Once a node is created there is no way of backtracking and changing the attribute split condition later in the tree-building process. In contrast, evolutionary algorithms provide a global optimization approach for tree construction that the traditional methods lack.

Simple Genetic Algorithm in Tree-Building

Evo-Tree [19] is an example where an evolutionary algorithm (genetic algorithm) is used in the tree-building process. The trees start with a random population and evolve using a multi-objective GA. To be accurate, it works with a single objective

which is a weighted aggregate of two objectives. The two objectives are validation set accuracy and tree size. It is to be noted that balancing complexity and accuracy as competing objectives and employing multi-objective optimization to solve is a powerful idea. And despite its potential benefits, a single objective optimization is performed.

Components of the Evo-Tree GA

1. Fitness function: The fitness function is a combination of both objectives mentioned earlier, with weights assigned to each.

$$fitness = \alpha_1 f_1 + \alpha_2 f_2 \tag{24}$$

$$f_1 = 1 - acc \tag{25}$$

$$f_2 = \frac{Tree\ current\ depth}{Tree\ target\ depth} \tag{26}$$

   Where f_1 is the measure of inaccuracy on a validation set, and f_2, is the measure of tree complexity in terms of tree depth.
2. Genetic Representation: A Decision tree is represented as a chromosome. There are several ways to encode the tree into a chromosome, the easiest and the most natural way to do it is by using tree encoding. A decision tree is represented by a binary tree, where each node has an attribute and also a threshold value. It represents the splitting condition based on an attribute value threshold at the corresponding node of the decision tree. It works under the assumption that all attributes are numeric in nature, and the categorical features are to be converted into numeric beforehand.
3. Crossover: A child is created by performing a crossover between two parent trees. A node is randomly selected for both trees as the crossover point. The subtree rooted at the crossover point for the first parent tree is replaced by the subtree rooted at the crossover point of the second parent tree, creating a new child tree.
4. Mutation: A node condition mutation is implemented. First, a node is randomly selected as the mutation point. The attribute and the threshold value present at the node are replaced by some arbitrary attribute and value.

Additional Works

There are a lot of various implementations of evolutionary algorithms in building decision trees. Tree-GA hybrid [26, 7] has a two-stage training scheme. First, a D-tree is created by using a traditional algorithm, and then a population of all the paths from the root to the distinct leaves are treated as tree rules. To be more specific, the conjunction of the split conditions satisfied along a particular path makes a rule. Then the population of rules is evolved using GA. Genetic programming could also

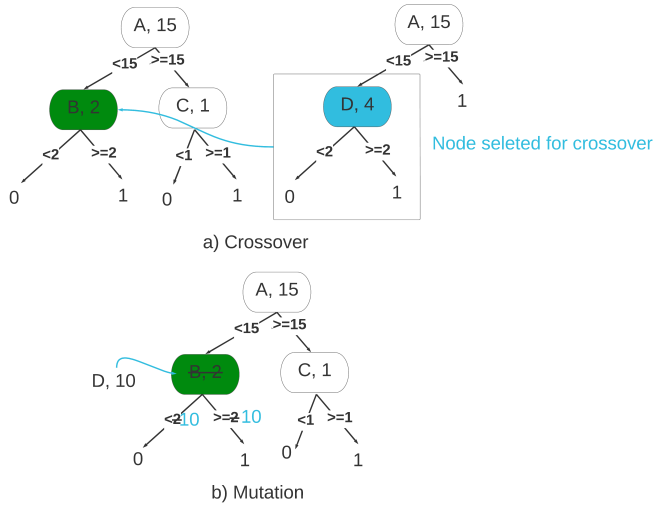

Fig. 9 Genetic operations in Evo-Tree algorithm

be used for creating decision trees [12]. In the work of Evans et al. [13], a multi-objective genetic programming approach is suggested. The objectives are improving accuracy and reducing tree complexity. The implementation is a hybrid between NSGA-II and GP.

3.3.2 Learning Classifier System

A Learning Classifier System (LCS) [46] is a rule-based approach which maintains a population of rules or classifiers. These rules are evaluated on the basis of their fitness in solving a particular problem and the population evolves over time through a process of selection, recombination, and mutation. It is important to note that LCS is more of a programming paradigm rather than a specific method. It is made up of several components which could be modified as per the task. LCS has been widely used in both supervised as well as in reinforcement learning.

Components of a generic LCS

The components may vary based on the specification of the task. The basic components of a generic LCS are listed below.

1. Current Knowledge: The finite population of the classifier or rules that represents the system's overall body of collective knowledge.
2. Performance Component: It involves the interplay between the environment and the classifier population.

3. Credit Assignment (Reinforcement): It distributes the reward (updates fitness) to the classifiers after interaction with the environment.
4. Discovery: Discovers or modifies the population of classifiers using Genetic Algorithm.

Key Elements of LCS

The elements may vary based on the task specification. To be more specific, we are considering the Michigan-style implementation for a binary classification task [45].

- Environment: The environment is a source of experience. It is simply the training data for the classification task. In the Michigan-style system, where the learning is incremental, only one training instance is used for one learning cycle.
- Classifier: A classifier or a rule is an ”if condition then action” statement. In the context of classification, it could be viewed as the condition being a conjunction of conditions on feature values and action being the class label. Note that a classifier itself is not a complete ML model. For a given test example, it may be possible that a particular rule is not applicable. It is more appropriate to treat it as a local model while the entire population of classifiers collectively forms the ML model.
- Matching set: While training or predicting a single instance is queried. The matching set is a subset of the population of classifiers whose conditions are satisfied (matched) by the instance.
- Covering mechanism: In the training cycle, if no classifier belongs to the matching set, then new classifiers are introduced which match the instance by a covering mechanism.

Rule Matching and Covering

A classifier or rule has a condition which is a conjunction of conditions on the individual feature values. For continuous features, it could be an interval of values and for discrete features, it could be a particular value that it could take. Wildcard ”#” is the condition on a feature which allows it to take any possible value. One instance from the training set is chosen for a learning cycle. If the feature values of the selected instance satisfy all the feature conditions of a rule, then the rule is said to ”match” and is moved to the matching set. See Figure 10.

A covering mechanism is triggered if none of the population’s rules matches the training instance. New rules are derived based on the feature values of the training instance. For example, if all the features are discrete, some of the features are arbitrarily selected and assigned the feature condition ”#” wildcard. For the rest of the features, the condition allows them to only take the same value as the training instance (Eq. 27). If the population is already at capacity, the new rules replace some of the rules present in the population.

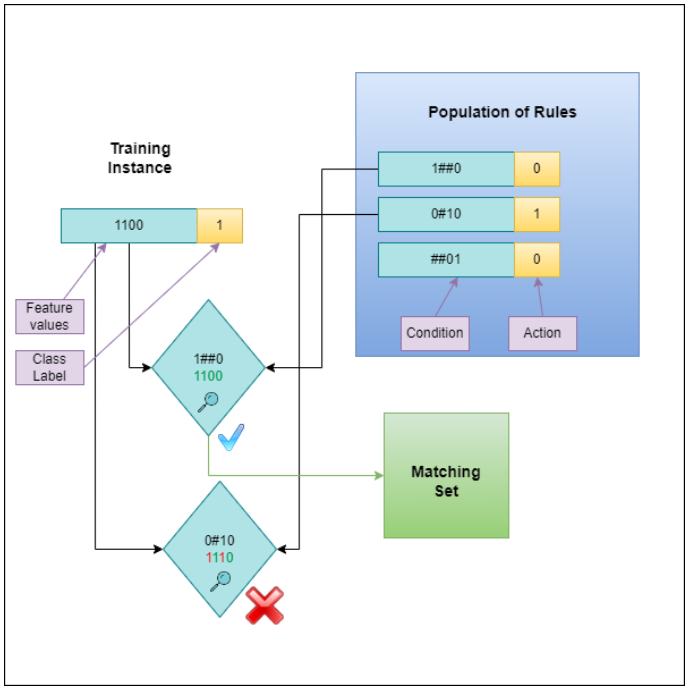

Fig. 10 The matching process in LCS. The training instance is shown on the left with its attributes taking boolean values, belonging to class 1. The classifiers in the population set are compared one by one. If the non-wildcard attributes of the classifier and the instance are identical then the classifier is included in the matching set.

$$\textit{Instance}: [1, 2, 2, 0, 1] \rightarrow 1; \textit{New Rule}: [\#, 2, \#, 0, 1] \rightarrow 1 \tag{27}$$

A Learning Cycle

UCS [6, 33] is an LCS framework, which is used specifically in the supervised learning setting. Following is the description of a learning cycle for UCS for the binary classification task.

There are certain parameters that are associated with a classifier. These parameters are updated in every training cycle. In UCS, the main parameters are i. accuracy (acc), ii. fitness (F), iii. numerosity (num) and iv. experience (exp).

$$acc = \frac{number\ of\ correct\ classifications}{experience} \tag{28}$$

$$F = (acc)^c \tag{29}$$

Where c is a user-defined constant. The experience is incremented whenever the classifier participates in a Matching Set. The numerosity is the count of the number

of replicas of the classifier in the classifier population. Fitness is used in the rule discovery which is done by the GA. It is also used to regulate the population size of the classifier population.

First a training instance x is selected from the training set. LCS begins with an empty population [P] of classifiers, in contrast to other population-based evolutionary algorithms. In the first cycle, the population is automatically filled by the covering mechanism. In a usual cycle, the training instance x is compared one by one with the condition of the classifiers in [P] to create a matching set [M]. The set [M] is further divided into two subsets, correct [C] and incorrect classifiers [IC]. It is done by comparing the action of the classifiers with the class label associated with x. At this point, if [C] is empty, then the new rules, created using the covering mechanism, replace some of the rules in [P]. So far, the performance and the credit assignment components of the LCS are only involved. Now new rules are generated by the discovery component by using the genetic algorithm. GA selects two classifiers from the population as parents based on fitness values. These parents create two offspring by applying recombination and mutation operations. The current population is updated by inserting these new classifiers, while some classifiers are removed to preserve the population size.

Additionally, a subsumption mechanism could be applied in every cycle. It removes the rules which are overly specific and redundant, in presence of a more general rule which has similar accuracy. In this process, the numerosity of the general rule, which covers the feature space of the redundant rule, is incremented.

Prediction and Interpretability

At the time of prediction, a matching set [M] of rules is created. Then based on fitness and numerosity, a weighted vote of all the classifiers in [M] is used in order to make a prediction. The list of the rules used to generate a prediction is a list of simple ”IF:THEN” statements which are human readable. Investigating too many rules at a time in a complex system may become incomprehensible for a human reader. In these cases, the transparency of the model may be reduced, making it more challenging to comprehend the decisions being made by the system. To tackle this, several visualization techniques have also been implemented [27, 47].

Overall, whether LCS can be considered as a transparent model depends on the specific implementation, and the complexity of the rules learned. But LCS can be considered to be more interpretable than some other machine learning approaches. Additionally, because LCS uses an evolutionary process to learn the rules, it is possible to trace the evolution of the rules over time and understand how they have changed and adapted to different situations. This can provide insights into the decision-making process and make the model more transparent.

4 Concluding Remarks

The importance of explainability in artificial intelligence (AI) systems has been highlighted in this book chapter, and the use of evolutionary and metaheuristic algorithms to achieve it has been discussed. Various research efforts in this area have been outlined, including counterfactual explanations, local surrogate modeling, and transparent models. Counterfactual explanations involve generating alternative scenarios that could have led to a different outcome, while local surrogate modeling aims to simplify complex black-box models. Transparent models, in contrast, are inherently interpretable and can provide insights into how they reached a specific decision or prediction.

Given the increasing role of AI in our lives, it is crucial that we prioritize transparency and accountability in these systems. To achieve this, evolutionary and metaheuristic algorithms are well-suited and flexible for some of the optimization tasks involved in explainability. These algorithms can handle high-dimensional and nonlinear search spaces effectively and can handle multiple competing objectives and constraints. Furthermore, their ability to explore a wide range of potential solutions makes them useful. We conclude that evolutionary and metaheuristic algorithms are valuable tools for developing trustworthy and responsible AI systems.

References

[1] Andersen H, Lensen A, Browne WN, Mei Y (2022) Evolving counterfactual explanations with particle swarm optimization and differential evolution. In: 2022 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp 01–08

[2] Arrieta AB, Díaz-Rodríguez N, Del Ser J, Bennetot A, Tabik S, Barbado A, García S, Gil-López S, Molina D, Benjamins R, et al (2020) Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. Information fusion 58:82–115

[3] Bacardit J, Brownlee AE, Cagnoni S, Iacca G, McCall J, Walker D (2022) The intersection of evolutionary computation and explainable ai. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp 1757–1762

[4] Barredo-Arrieta A, Del Ser J (2020) Plausible counterfactuals: Auditing deep learning classifiers with realistic adversarial examples. In: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, pp 1–7

[5] Van den Bergh F (2002) An analysis of particle swarm optimizers [ph. d. dissertation]. Department of Computer Science, University of Pretoria, Pretoria, South Africa

[6] Bernadó-Mansilla E, Garrell-Guiu JM (2003) Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. Evolutionary computation 11(3):209–238

[7] Carvalho DR, Freitas AA (2000) A hybrid decision tree/genetic algorithm for coping with the problem of small disjuncts in data mining. In: Proceedings of the 2nd annual conference on genetic and evolutionary computation, pp 1061–1068
[8] Dandl S, Molnar C, Binder M, Bischl B (2020) Multi-objective counterfactual explanations. In: Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I, Springer, pp 448–469
[9] Dastile X, Celik T, Vandierendonck H (2022) Model-agnostic counterfactual explanations in credit scoring. IEEE Access 10:69543–69554
[10] Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation 6(2):182–197
[11] Duong TD, Li Q, Xu G (2021) Prototype-based counterfactual explanation for causal classification. 2105.00703
[12] Espejo PG, Ventura S, Herrera F (2009) A survey on the application of genetic programming to classification. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 40(2):121–144
[13] Evans BP, Xue B, Zhang M (2019) What's inside the black-box? a genetic programming method for interpreting complex machine learning models. In: Proceedings of the genetic and evolutionary computation conference, pp 1012–1020
[14] Ferreira LA, Guimarães FG, Silva R (2020) Applying genetic programming to improve interpretability in machine learning models. In: 2020 IEEE congress on evolutionary computation (CEC), IEEE, pp 1–8
[15] Guidotti R, Monreale A, Ruggieri S, Pedreschi D, Turini F, Giannotti F (2018) Local rule-based explanations of black box decision systems. arXiv preprint arXiv:180510820
[16] Hashemi M, Fathi A (2020) Permuteattack: Counterfactual explanation of machine learning credit scorecards. arXiv preprint arXiv:200810138
[17] Hassan R, Cohanim B, De Weck O, Venter G (2005) A comparison of particle swarm optimization and the genetic algorithm. In: 46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference, p 1897
[18] Holland JH (1992) Genetic algorithms. Scientific american 267(1):66–73
[19] Jankowski D, Jackowski K (2014) Evolutionary algorithm for decision tree induction. In: Computer Information Systems and Industrial Management: 13th IFIP TC8 International Conference, CISIM 2014, Ho Chi Minh City, Vietnam, November 5-7, 2014. Proceedings 14, Springer, pp 23–32
[20] Katoch S, Chauhan SS, Kumar V (2021) A review on genetic algorithm: past, present, and future. Multimedia Tools and Applications 80:8091–8126
[21] Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks, IEEE, vol 4, pp 1942–1948

[22] Koza J (1992) Genetic programming, on programming of computer by natural selection
[23] Kumar A (2013) Encoding schemes in genetic algorithm. International Journal of Advanced Research in IT and Engineering 2(3):1–7
[24] Lash MT, Lin Q, Street N, Robinson JG, Ohlmann J (2017) Generalized inverse classification. In: Proceedings of the 2017 SIAM International Conference on Data Mining, SIAM, pp 162–170
[25] Li R, Emmerich MT, Eggermont J, Bäck T (2013) Mixed integer evolution strategies for parameter optimization. Evolutionary computation 21(1):29–64
[26] Liu Ds, Fan Sj (2014) A modified decision tree algorithm based on genetic algorithm for mobile user classification problem. The Scientific World Journal 2014
[27] Liu Y, Browne WN, Xue B (2021) Visualizations for rule-based machine learning. Natural Computing pp 1–22
[28] Mehrabi N, Morstatter F, Saxena N, Lerman K, Galstyan A (2021) A survey on bias and fairness in machine learning. ACM Computing Surveys (CSUR) 54(6):1–35
[29] Montavon G, Samek W, Müller KR (2018) Methods for interpreting and understanding deep neural networks. Digital signal processing 73:1–15
[30] Monteiro WR, Reynoso-Meza G (2022) Counterfactual generation through multi-objective constrained optimisation
[31] Mothilal RK, Sharma A, Tan C (2020) Explaining machine learning classifiers through diverse counterfactual explanations. In: Proceedings of the 2020 conference on fairness, accountability, and transparency, pp 607–617
[32] Navas-Palencia G (2021) Optimal counterfactual explanations for scorecard modelling. arXiv preprint arXiv:210408619
[33] Orriols-Puig A, Bernadó-Mansilla E (2006) A further look at ucs classifier system. GECCO'06 pp 8–12
[34] Preece A, Harborne D, Braines D, Tomsett R, Chakraborty S (2018) Stakeholders in explainable ai. arXiv preprint arXiv:181000184
[35] Quinlan JR (1987) Simplifying decision trees. International journal of man-machine studies 27(3):221–234
[36] Rasheed K, Hirsh H, Gelsey A (1997) A genetic algorithm for continuous design space search. Artificial Intelligence in Engineering 11(3):295–305
[37] Rasouli P, Chieh Yu I (2022) Care: Coherent actionable recourse based on sound counterfactual explanations. International Journal of Data Science and Analytics pp 1–26
[38] Ribeiro MT, Singh S, Guestrin C (2016) " why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 1135–1144
[39] Ribeiro MT, Singh S, Guestrin C (2018) Anchors: High-precision model-agnostic explanations. In: AAAI Conference on Artificial Intelligence (AAAI)
[40] Schleich M, Geng Z, Zhang Y, Suciu D (????) Geco: Quality counterfactual explanations in real time. arXiv preprint arXiv:210101292

[41] Sharma S, Henderson J, Ghosh J (2020) Certifai: A common framework to provide explanations and analyse the fairness and robustness of black-box models. In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, pp 166–172
[42] Slack D, Hilgard S, Jia E, Singh S, Lakkaraju H (2020) Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, pp 180–186
[43] Storn R, Price K (1997) Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11(4):341
[44] Tolomei G, Silvestri F, Haines A, Lalmas M (2017) Interpretable predictions of tree-based ensembles via actionable feature tweaking. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp 465–474
[45] Urbanowicz R, Browne W (2015) Introducing rule-based machine learning: a practical guide. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp 263–292
[46] Urbanowicz RJ, Moore JH (2009) Learning classifier systems: a complete introduction, review, and roadmap. Journal of Artificial Evolution and Applications 2009
[47] Urbanowicz RJ, Granizo-Mackenzie A, Moore JH (2012) An analysis pipeline with statistical and visualization-guided knowledge discovery for michigan-style learning classifier systems. IEEE computational intelligence magazine 7(4):35–45
[48] Ustun B, Spangher A, Liu Y (2019) Actionable recourse in linear classification. In: Proceedings of the conference on fairness, accountability, and transparency, pp 10–19
[49] Wachter S, Mittelstadt B, Russell C (2017) Counterfactual explanations without opening the black box: Automated decisions and the gdpr. Harv JL & Tech 31:841
[50] Zitzler E, Thiele L (1998) Multiobjective optimization using evolutionary algorithms—a comparative case study. In: Parallel Problem Solving from Nature—PPSN V: 5th International Conference Amsterdam, The Netherlands September 27–30, 1998 Proceedings 5, Springer, pp 292–301