**Computing Rule-Based Explanations by Leveraging Counterfactuals**

Zixuan Geng, Maximilian Schleich, Dan Suciu – Fatima Vahora, Haeun Suh, Prashant Ravic summary

**ABSTRACT**

We introduce a new method to efficiently compute rule-based explanations for automated high-stakes decisions, by leveraging counterfactual explanations, for which many systems are already in place. To validate our approach, we present a Duality Theorem that establishes a relationship between rule-based and counterfactual explanations. Through comprehensive experiments, we demonstrate that our system outperforms or matches the performance of previous systems like MinSetCover and Anchor.

1. **INTRODUCTION**

Due to the increasing adoption of machine learning in high-stakes decisions, there is an urgent need for more explainable and debuggable models. As a result, explainable machine learning has become a crucial research topic.

The Counterfactual Explanation (also known as Actionable Recourse) is a form of local explanation that focuses on individual instances, informs users what features must change for a machine learning model to predict a positive outcome from a previously negative one. On the other hand, the Rule-based Explanation is conjunctions of predicates on features consistently leading to certain outcomes and descriptively provide core reasons for decisions. The counterfactual explanation can potentially mislead by not reflecting all influential features, whereas the rule-based explanation is more challenging and often requires complex solutions such as converting the issue into a minimum set-cover problem.

This paper attempts to improve the existing counterfactual system through a new version of a rule-based explanation system. Assume that the two systems are dual, three core algorithms (GeneticRule, GeneticRuleCF, and GreedyRuleCF) are introduced to compute rule-based explanations by employing counterfactual explanations as a black box. Assuming proof that the two systems are dual, three core algorithms (GeneticRule, GeneticRuleCF, and GreedyRuleCF) are introduced to compute rule-based explanations by employing counterfactual explanations as a black box as well as verified their efficiency experimental evaluation comparing with MinSetCover and Anchor.

In summary, the following contributions will be made in this paper:

(1) Prove the Duality Theorem between counterfactual and rule-based explanations. (Section 3.1.)

(2) Demonstrate applying the Duality Theorem to compute rule-based explanations by using a counterfactual-based explanation system. (Section 3.2.)

(3) Introduce the three core algorithms: GeneticRule, GeneticRuleCF, and GreedyRuleCF (Section 4.)

(4) Verify the algorithms by experimental evaluation and comparison. (Section 5.)

1. **DEFINITIONS**

|  |
| --- |
| * 𝐹1, . . ., 𝐹𝑛 be n features, with ordered domains 𝑑𝑜𝑚(𝐹1), . . ., 𝑑𝑜𝑚(𝐹𝑛). * Define INST = 𝑑𝑜𝑚(𝐹1) × · · · × 𝑑𝑜𝑚(𝐹𝑛). * Call an element 𝑥 ∈ INST an *instance*. * Given a black box classifier 𝐶, on any instance 𝑥 ∈ INST, returns a prediction 𝐶(𝑥) within range [0, 1] * Assume that C(x) ≤ 0.5 is an “undesired” or “bad”, while C(x) > 0.5 is “desired” or “good”. For the binary classifier, replace the C(x) with each case as {0, 1} respectively. * Assume a database D = {x1, . . ., xm} with m instances. Can be a training or test set, or data from historical decisions derived from the system. * For every instance xi in D, its feature values are given as xi = (fi1, . . ., fin). |

* 1. **Rule-based Explanation**

For a given instance xi = (fi1, …, fin) ∈ 𝐷, a rule component RC(xi), or RC relevant to xi, is a predicate of the form F𝑗 ≤ 𝑓𝑖𝑗 or F𝑗 ≥ 𝑓𝑖𝑗 for some feature Fj. A rule relevant to 𝑥𝑖 is a set of rule components, represented as 𝑅 = {𝑅𝐶1, . . ., 𝑅𝐶𝑐} where cardinality of the rule is c. R(x) is a predicate that is the conjunction of the R, that is, R(x) = RC1 ∧ · · · ∧ -RCC. To indicate that a feature value 𝐹𝑗 = 𝑓𝑖𝑗, it is essential to incorporate both ≤ and ≥ rule components, meaning the cardinality constraint is 0 ≤ 𝑐 ≤ 2𝑛. Denote INST𝑅 the set of all instances that satisfy R, thus, INST𝑅 = {𝑥|𝑥 ∈ INST𝑅, R(x) = 1}.

If an instance xi is classified as "undesired", then C(xi) ≤ 0.5, and let R be some rule. Suggested by Rudin and Shaposhnik, a rule R can be used as explanation for the instance xi when satisfy below:

1. **Relevance**: Ensure xi ∈ INST𝑅

\* The paper inherently satisfies this requirement as it only considers rules relevant to instance xi.

1. **Global Consistency**: ∀𝑥 ∈ INST𝑅, 𝐶(𝑥) ≤ 0.5.
2. **Interpretability**: The rule must be simple with minimal cardinality.

The paper identifies a trivial rule, Rtriv, that guarantees global consistency. However, its cardinality, being 2n, compromises its interpretability. For computational feasibility, the study emphasizes the need to derive a minimal set of rule components that maintain global consistency. Recognizing the computational demands of exhaustive global consistency checks, the paper advocates "Data Consistency". This approach evaluates consistency relative to a predefined database D = {x1, …, xm}, ensuring ∀𝑥𝑘 ∈ 𝐷 ∩ INST𝑅, 𝐶 (𝑥𝑘) ≤ 0.5.

It is noted that certain methods, such as MinSetCover and Anchor, might not always uphold global consistency.

* 1. **Counterfactual Explanation**

Counterfactual explanations identify changes in specific features that could lead to a desired outcome. Given an instance xi with and undesired outcome, a counterfactual explanation xcf specifies how xi could be modified to achieve a desired outcome where C(xi) > 0.5.

xcf must satisfy two main properties:

(1) Feasibility and Plausibility:

- Feasibility sets limits on potential feature values (e.g., income constraints).

- Plausibility dictates how new values in xcf can differ from the values in xi (e.g., gender shouldn't change).

- These criteria are encompassed by the PLAF (plausibility/feasibility) predicates:

P(xcf) = Φ1 ∧ · · · ∧ Φ𝑚 ⇒ Φ0, where Φi is a predicate over the features of xi and xcf.  
- Examples:

e.g., Feasibility: genderCF = genderi

e.g., Plausibility: education𝐶𝐹 > education𝑖 ⇒ age𝐶𝐹 ≥ age𝑖 +4

(2) Magnitude of Changes:

- Changes between xi and xcf can be quantified using a distance function dist(x, x’)

* A counterfactual explanation system takes as input an instance xi, a PLAF constraint P(x), and a distance function dist(x, x’). It outputs a ranked list of counterfactuals based on their proximity to xi.
  1. **Discussion**

Different explanations offer varied insights:

SHAP Score assigns significance percentages to features, helping rank them but not providing actionable guidance. In the case of Counterfactual Explanation, it suggests specific feature changes for desired outcomes but might raise fairness questions. In the other hand, rule-based Explanation outlines decision-making criteria, ensuring consistency and fairness but without actionable advice.

1. **DUALITY**

Rule-based explanations and counterfactual explanations offer distinct information. Though both prioritize brevity — a relevant rule should have few components, and a counterfactual should modify minimal features of 𝑥𝑖. Efficient counterfactual systems are available, but current rule-based systems compromise global consistency for speed.

When comparing counterfactual and rule-based explanations, differing complexities arise due to the nature of their constructions. Using a small set of features, F, counterfactual explanations require 𝑁 𝑘 calls to the oracle, assuming domains of size N, to ascertain possible changes. Rule-based explanations, on the other hand, necessitate N n−k calls to the classifier to verify global consistency, which can become extensive given the difference in feature counts. An innovative approach proposes using a counterfactual explanation system as a black box to compute rule-based explanations, tapping into an advantageous duality that exists between the two types of explanations.

* 1. **The Duality Theorem**

Lemma 3.1. If 𝑅 is a globally consistent rule, and 𝑥𝑐𝑓 is any counterfactual, then 𝑅(𝑥𝑐𝑓) is false.

Proof. Given Q: ∀𝑥’(𝑅(𝑥′) = T -> 𝐶(𝑥′) ≤ 0.5). Also Given R(xcf) = ⌝𝑅(𝑥′).

Contrapositive of Q’: ∀𝑥𝐶(𝑥′)> 0.5) = F -> ⌝ 𝑅(𝑥′) = R(xcf)

Dual rules: Given two instances xi and x, the dual rule for x (denoted as Rx) consists of a union (logical OR) of rule components that represent the disparities or “conflicts” between feature values of x and xi.

e.g., Given xi = (F1 = 10, F2 = 20, F3 = 30) and x = (F1 = 5, F2 = 90, F3 = 30).

Then dual rule Rx = (F1 ≥ 10) ∪ (F2 ≤ 20)

Theorem 3.2 (Duality). Fix a globally consistent rule 𝑅 relevant to 𝑥𝑖, let 𝑥𝑐𝑓 ,1, . . ., 𝑥𝑐𝑓, 𝑘 be counterfactual instances, and let 𝑅𝑥𝑐𝑓 ,1, . . ., 𝑅𝑥𝑐𝑓, 𝑘 be their duals. Then 𝑅 is a set cover of {𝑅𝑥𝑐 𝑓 ,1, . . ., 𝑅𝑥𝑐𝑓, 𝑘}. In other words, for every counterfactual 𝑥𝑐f, 𝑚 the rule 𝑅 contains at least one rule component that conflicts with 𝑥𝑐𝑓, 𝑚. Conversely, fix any counterfactual 𝑥𝑐𝑓, and let 𝑅1, . . ., 𝑅𝑘 be globally consistent rules. Then the dual 𝑅𝑥𝑐 𝑓 is a set cover of {𝑅1, . . ., 𝑅𝑘}.

Proof. If R and Rxcf, m does not share any common rule component, then Rxcf, m is would be true, contradicting the Lemma 3.1

e.g., Given xi = (F1 = 10, F2 = 20, F3 = 30) [Good] and x = (F1 = 15, F2 = 19, F3 = 30) [BAD]

And dual rule Rx have been (F1 ≥ 10) ∪ (F2 ≤ 20)

If there is no conflict, then x should be Good which is contradictory.

Therefore, at least one confliction is needed.

* 1. **Using the Duality Theorem**

Theorem 3.2 provides a naive algorithm. However, this can't be used because counterfactual systems often don't produce a comprehensive list of counterfactuals. Our approach builds the rule R incrementally, starting from R = ∅, adding one rule component at a time until achieving global consistency. The steps are as follows:

Step 1: Construct the predicate R (x ′) associated with the rule R.

Step 2: Utilize the counterfactual explanation system to identify a list of counterfactuals 𝑥𝑐𝑓 ,1, . . ., 𝑥𝑐𝑓, 𝑘 for 𝑥𝑖 that meet the PLAF predicate. If no counterfactual matches, then R is deemed globally consistent, and the process stops.

Step 3: For each 𝑗 = 1 to 𝑘, compute the dual Rxcf, j for each counterfactual 𝑥𝑐𝑓, 𝑗.

Step 4: For every minimal set that encompasses 𝑅0 of 𝑆, create the augmented rule 𝑅 ∪ 𝑅0 and start again from Step 1.

e.g., xi = (Age = 50, AccNum = 4, Income = 500, Debt = 10k) => “DENIED”

Step1. (Current) R = (Age <= 50) ∧ (AccNum >= 4)

Step2. 𝑥𝑐𝑓 ,1 = (Age = 50, **AccNum = 5, Income = 900**, Debt = 10𝑘) => “APPROVE”

𝑥𝑐𝑓 ,2 = (Age = 50, AccNum = 4, **Income = 600**, **Debt = 2𝑘**) => “APPROVE”

Step3. Rxcf, 1 = (AccNum ≤ 4) ∨ (Income ≤ 500)

Rxcf, 2 = (Income ≤ 500) ∨ (Debt ≥ 10𝑘)

Step4. R1 = (Age ≤ 50) ∧ (AccNum ≥ 4) ∧ (Income ≤ 500) Cardinality = 3

𝑅2 = (Age ≤ 50) ∧ (AccNum = 4) ∧ (Debt ≥ 10𝑘) Cardinality = 4

\* ‘=’ represents two rule component AccNum≤4 and AccNum≥4

1. ALGORITHMS

GeneticRule: This is a base-line algorithm, which explores the space of rule-based explanations using a genetic algorithm. It does not use counterfactuals.

GeneticRuleCF: This algorithm extends GeneticRule by using an oracle call to a counterfactual explanation system to generate and validate the rule-based explanations.

GreedyRuleCF This algorithm replaces the genetic search with a greedy search: we greedily expand only the rule with the smallest cardinality in the population, using the counterfactual explanation system as an oracle.

* 1. A white paper with black text

     Description automatically generatedGeneticRule

GeneticRule is a “naïve” algorithm that generates rules by a genetic algorithm, shown in Algorithm 1.

* Input: Instance x, classifier C, dataset D.
* Output: a set of rules explaining instance x for classifier C.
* Five integer hyperparameters
  + q > 0: The number of rules that kept after each iteration.
  + k <= q: The number of rules that the algorithm returns to the user.
  + s: The number of samples taken from INST to check for global consistency.
  + m: The number of new candidates that are generated during mutation.
  + c: The number of new candidates that are generated during crossover.

e.g., q = 50, k = 5, s = 1000, m = 3, c = 2. Used in most of the experiments. (Why? See sec. 5)

1. Computes the initial population of rule candidates.
   1. Initial population has all possible rule candidates with exactly one rule component.
   2. Initial candidates are likely not valid and consistent rules.
2. Repeatedly applies below:
   1. Mutate and crossover to generate new candidates.
   2. Compute the fitness score vis selectFittest for each candidate.
   3. Selects the q fittest candidates for the next generation.
   4. Repeat until finding K candidates that are consistent on both the dataset D and s samples from the more general INST space.
3. Check the top-k candidates are not in the set of new generated candidates CAND to guarantee their stability for at least one generation of the algorithm.

* Mutate operator: Generates m new rule candidates for each candidate 𝑅 ∈ POP.
  + Find the set S of all rule components that are not part of R.
  + Generates each new candidate by sampling without replacing a single component from S and adding it to R. (For keeping the low cardinality of the rules and avoiding redundant rule components as possible)
* Crossover operator: Generates c new candidates for each pair of candidates, Ri and Rj.
  + Compute the set 𝑆 = 𝑅𝑖 ∪𝑅𝑗 of all rule components in Ri and Rj.
  + Randomly sample t components from S to form a new candidate, as 𝑡 = 𝑚𝑎𝑥 (|𝑅𝑖|, |𝑅𝑗|) + 1 to keeping the low cardinality of the rules.
  + Repeat the sampling process c times to generate c candidates for every pair of candidates in the population.
* The selectFittest operator: Calculate the fitness score of each candidate in POP, sorts the candidates by in descending order of their fitness scores, and returns the top q candidates.
  + See sec 4.4 for detail.
* GeneticRule only guarantees of the returned rules are global consistency for a sample of INST, thus only guaranteed the rules are data consistent.
  1. A white text with black text

     Description automatically generatedGeneticRuleCF

GeneticRuleCF applies a counterfactual explanation to GeneticRule to generate and verify rule candidates, shown in Algorithm 2.

* CFRules function: This function takes as input a set of rule candidates and generates new candidates by computing the counterfactual explanations for each input candidate.
  + Consisted as multiple processes like as below (Also in Section 3.2):
    - Compute the PLAF predicates for a given input candidate.
    - Compute the counterfactual explanation for this candidate.
    - Return the dual of the counterfactual explanation as a new candidate.
  + It extends both initial population and the candidate pool in the main loop of the genetic algorithm.
* consistentCF function: This function verifies global consistency of the top rule candidates using a counterfactual model.
  + By checking if any counterfactual example exists for a given candidate, this function guarantees higher global consistency than GeneticRule.
* Performance optimizations: To maximize performance at minimal effects:
  + Reduced the calling of the counterfactual explanation model, by only running CFRules once for every three iteration or when all top-k candidates are marked as data consistent.
  + Cache to check availability of generating counterfactuals for each rule candidate, to ensure the single run of the counterfactual model per candidate.
* An optional post process stage – not shown in the pseudo code: Ensures that the returned rules have no redundant components.
  + Remove one rule component at a time for each returned rule.
  + Check if the rule without this component is still verified by consistentCF.
  + If so, remove that redundant component.
  + Repeat the above processes until the returned rules do not have any redundant component.
  1. A screenshot of a computer code

     Description automatically generatedGreedyRuleCF

GreedyRuleCF greedily finds rule candidates with small cardinality by repeatedly utilizing the underlying counterfactual explanation, shown in Algorithm 3.

* Generate the initial population by running CFRules, keep them sorted as ascending order of their cardinalities.
* Choose the candidate with the smallest cardinality, generate the new candidate towards this candidate by CFRules, and replace the secondary candidate with the new candidate from the population.
* Repeat until the candidate with the smallest cardinality is found to be consistent by consistentCF.
* The cardinality of the rules in the population are monotonically increasing while repeatedly replacing the inconsistent rule candidate with the smallest cardinality.
* The algorithm is guaranteed to terminate with a consistent rule by having at most 2n rule components, where n is the number of variables in D.
  1. Fitness Score Function
* selectFittest function: Computes the fitness score for ranking the rule candidates.
  + The fitness score is based on its degree of consistency and its cardinality.
  + Three degrees of consistency:
    - The rule failed data consistency (𝐹𝐷𝐶): it violates instances in the database 𝐷.
    - The rule failed global consistency (FGC): it is data consistent (satisfies all instances in the dataset D), but fails for some instances in Inst.
    - The rule is globally consistent (𝐺𝐶): The rule is consistent for both the dataset 𝐷 as well as the instances from Inst.
  + The fitness score of a Rule R):
    - Define m as the cardinality of D (a.k.a |D|).
    - Define n is the number of features in D.
    - Define VD the number of instances in D that violates R.
    - If VD = 0, use sample s instances from INST, and define VS the number of samples the violates R.
    - A math equations with numbers

      Description automatically generatedThe fitness score - score(R):
    - The expressions were chosen to ensure that the score function always ranks candidates in each level higher than the candidates in the levels below, to prioritize candidates having better consistency.

e.g., GC > FGC > FDC for different level, min(cardinality) for the same level.

1. EXPERIMENTS
   1. Experiment Setup

* DataSets and Classfiers: Considers four real dataset.

(1) Credit Dataset [28]: used to predict the default of the customers on credit card payments in Taiwan.

(2) Adult Dataset [9]: used to predict whether the income of

adults exceed $50K/year using US census data from 1994.

(3) FICO Dataset [6]: used to predict the credit risk assessments.

(4) Yelp Dataset [29]: used to predict review ratings that users

give to businesses.

* + A table with text and numbers

    Description automatically generatedTable 1 illustrates the key statistics and the classifier type used for each data set.
  + Credit and Adult are from the UCI repository and are commonly used in the machine learning explanation fields.
  + FICO is from the public FICO challenge, which is an Explainable Machine Learning Challenge, using two-layer neural networks classifier.
  + Yelp uses a complex MLPClassifier with 10 layers.
  + Create synthetic classifiers for the Credit dataset to proof if the systems can recover the rules when ground truth is known.
  + The algorithms access the classifier as a block box.
  + Expecting relatively small number of the rule components is in real rule-based explanations, by 10 to be interpretable for the typical user, 2, 4, 6 and 8 rule components included classifiers are created.
  + To ensure fairness, the same preprocessing for all the systems applied.
* Underlying Counterfactual Explanation Model.
  + Benchmarked 13 different counterfactual explanation models and reported their evaluation to the public GitHub repository(<https://github.com/GibbsG/GeneticCF>)
  + A counterfactual explanation model GeCo has been chosen for the algorithms because it applies a flexible PLAF constraint and has no redundant feature changes.
* Considered Algorithms.
  + Benchmarks the algorithms against two existing system Anchor and MinSetCover.
    - Anchor: Generates rule-based explanations (i.e., anchors) by the beam-searched version of pure-exploration multi-armed bandit problem.
      * Procedure of Anchor: Starts with an empty rule. Then, iterate each rule like as follow:

1. Randomly select m possible rule components and add each of the possible components to the rule to create m new rules.
2. Evaluates all the rules.
3. Selects top n rule to keep for the next iteration.

It stops when the rules converge.

* + - * Assess whether a rule is consistent by sampling k instances to see if all instances return “unwanted” results for the rule.
    - MinSetCover: Generates rule-based exploration using the minimum set cover problem.
      * Consider the m instances in the database as elements and the binary predicates (≤ 𝐹𝑖 and ≥ 𝐹𝑖), to reduce the finding a minimum rule with data consistency to finding the minimum number of binary predicates that covers all those “undesired” instance.
      * Applies linear Programming for the above.
      * Only can be applied on the historical database D.
* Parameter Choice.
  + Five hyperparameters (ref. Sec 4.1):
    - k: The number of rules to be returned by the algorithm.
    - q: The number of rules to be kept in each iteration.
    - m: The number of new candidates generated during mutation.
    - c: The number of new candidates generated during crossover.
    - s: The number of samples from INST during evaluation.
  + Except k which differs f
  + The following combinations, obtained because of carrying out various pilot experiments within a time frame that allows for reanalysis with acceptable quality, are used in the experiments:
    - Adult, Credit and FICO: 𝑞 = 50, 𝑠 = 1000, 𝑚 = 3, 𝑐 = 2
    - Yelp: 𝑞 = 20, 𝑠 = 5000, 𝑚 = 3, 𝑐 = 2
    - The hyperparameter k is not considered as it depends on user requirements.
  + For GeneticRuleCF, we enable the optional post reduction stage, but limit it to reduce only the top rule to limit the overhead.
* Experimental pipeline.
  + The data is pre-processed as required by the classifiers:
    - Encoding categorical variables:
      * Credit and Yelp: Integer encoded.
      * Adult: Apply one-hot encoded.
    - Applied classifiers:
      * Credit and Adult: Decision tree.
      * Fico and Yelp: Multi-layer neuron network.
  + The post-processed dataset retains the same number of instances (tuples) as the original data. (ref. Table 1)
  + One explanation is for one single user, yet the system requires to examine at least the entire dataset D or the entire space of instance INST to provide this explanation to one instance. Therefore, if the system returns multiple explanations, retain the top-ranked rule only.
  + Measure the run time needed to find the explanation then evaluate its quality.
  + Repeat above process for 10,000 users (i.e., 10,000 explanations). Thus, in our experiments each system returns 10,000 rules.
* Evaluation Metrics.
  + Following two metrics applied to evaluate the quality of the generated rules:

(1) Global consistency: None of “desired” rules has been found by the classifier.

(2) Interpretability: Check the cardinality of the rule to determine whether the rule returned is truly minimal.

* + Only evaluates the top one rule-based explanation for GeneticRule and GeneticRuleCF.
* Setup.
  + Implemented the algorithms in Julia 1.5.2.
  + All experiments are run on an Intel Core i7 CPU Quad-Core/2.90GHz/64bit with 16GB RAM running on macOS Big Sur 11.6.
  1. Quality in terms of Consistency and Interpretability
* Compare all considered algorithms in terms of the quality of generated rules on the datasets.
* First, consider synthetic classifiers and then evaluate the considered systems on real classifiers.
* Synthetic Classifiers
  + The classifier is a rule itself and the task of the system is to find an explanation that is precisely that rule.
  + The rule-based explanation is categorized as follows:

(1) The rule exactly matches the classifier. That is, the rule is consistent and minimal.

(2) The rule is consistent but possesses redundant components. That is, it is a strict superset of the classifier.

(3) The rule is inconsistent with the classifier. That is, at least one rule component of the classifier has been missed.

* + A graph of different colored bars

    Description automatically generated with medium confidenceThe ratio of the three categories for each algorithm is illustrated in Figure 1.
    - GeneticRuleCF and GreedyRuleCF: Their rules are always consistent and minimal regardless of the cardinality of classifiers.
    - GeneticRule: Returns partial inconsistent when the cardinality of classifiers increased as 8. Suggests the need to include a counterfactual explanation system.
    - Anchor and MinSetCover: Even with the lowest cardinality as 2, both fail to find minimal rules though partially finds consistent rules. When cardinality increases, both systems even fail to find consistent rules.
* Real Classifiers
  + For the real classifiers, we categorize each rule 𝑅 in one of the following five categories:

(1) Failed data consistency (𝐹𝐷𝐶): there is an instance in the dataset 𝐷 where the rule fails.

(2) Failed global consistency (FGC): all instances in 𝐷 satisfy the rule, but it fails on some instances in Inst.

(3) Globally Consistent (𝐺𝐶) but redundant: the rule holds on all instances in Inst but has some redundant rule components.

(4) Globally Consistent (𝐺𝐶), non-redundant, but not minimal: the rule is globally consistent and non-redundant but is not of minimum size: there exists a strictly smaller globally consistent rule.

(5) Globally Consistent (𝐺𝐶) and minimal: has the smallest number of rule components.

* + It is difficult to determine the correct rule for this classifier directly, so instead, uses a method to check whether consistency is maintained even if components of the rule are removed as well as checking all possible rule sorted by the cardinality until finding a consistence one. If there are no more redundancies yet persisting consistency, then the rules are considered as minimal.
  + Uses Geco as a proxy for testing global consistency via searching of any existence of counterfactual models.
  + A close-up of a graph

    Description automatically generatedFigure 2 illustrates evaluation for each dataset with their specific classifiers like as below:
    - GeneticRuleCF and GreedyRuleCF: The two algorithms always find globally consistent rules, except in cases where Geco's limitations for a particular algorithm cause it to return partially inconsistent rules. Among them, GreedyRuleCF is the only algorithm that can achieve the most ideal steps, GC and minimal.
    - GeneticRule: GeneticRule does not guarantee to find globally consistent rules and has the limitation of only utilizing samples from INST. Therefore, this algorithm requires a counterfactual system for expansion.
  + The algorithms are superior to both in that they contain fewer redundant components than Anchor and in that they guarantee extensive consistency over MinSetCover's limited data consistency.
  1. Runtime Comparison
* Measures the runtime of all considered algorithms for the synthetic and real classifiers, especially in terms of impacts of the different cardinalities and size of datasets on the synthetic classifiers.
* Synthetic classifier.
  + A graph of different colored squares

    Description automatically generatedThe comparison of runtime for the synthetic classifiers is like as below Figure 3:
  + GeneticRule, GeneticRuleCF, and GreedyRuleCF usually consume less time than Anchor and MinSetCover, regardless of the cardinality of rules behind the classifier.
  + Since it needs more effort to generate the complicated rule, the runtime takes longer for the larger cardinality, especially for the traditional approach which adds the one rule component per one rule.
* Real Classifiers.
  + A close-up of a graph

    Description automatically generatedThe comparison of runtime for the real classifiers is like as below Figure 4:
  + Credit dataset: The runtimes of all algorithms are similar for the dataset, which suggests that the algorithms can efficiently generate consistent rule-based descriptions without any additional cost for moderately complex datasets and classifiers.
  + Adult dataset: For this dataset, the algorithms designed are much faster than existing algorithms, and this difference in performance is due to the characteristics of the algorithms that can handle one-hot encoding as a single function.
  + Fico and Yelp datasets: In this case, GeneticRuleCF and GreedyRuleCF took more time due to their internal strong verification mechanism.
  + Also, it is suggested that the algorithm is making classifier calls more frequently, which can affect performance, and is structured to depend on Geco, which also has a significant impact on performance, resulting in a delay.
  + A table with numbers and text

    Description automatically generatedTable 2 shows the contribution of the classifier and Geco to the runtime of each dataset as follows. In particular, the impact of Geco is further discussed in Sec 5.4.
  + Yelp dataset: The Yelp dataset has a relatively large number of instances and uses more complex classifiers, making rule creation challenging. Nevertheless, the Genetic algorithm shows superior performance to existing algorithms in terms of performance.
  + In summary, GeneticRuleCF can always finish in reasonable run-time to generate high quality rules regardless of the classifier speed or the size of the dataset. GreedyRuleCF typically generates rules with the highest quality, and it is fast when the classifier and dataset are moderate size. For complex classifiers over large datasets, however, GeneticRuleCF is more efficient than GreedyRuleCF.
  1. Microbenchmarks
* A graph of a number of datasets

  Description automatically generatedThe summary of microbenchmarks is illustrated in from Figure 5 to Figure 7 like as below:
* Breakdown of Runtime (Figure 5):
  + Note: Excluded GreedyRuleCF since it uses the counterfactual system as GECO only.
  + Prep (i.e., Preparation) spends its runtime computing the rule space and to build the initial population, while Selection (i.e., selectFittest), Crossover, Mutation and CFRules spends their runtime accumulated by all relevant iterations.
  + Since there are no running counterfactual explanations in the selectFittest(ref. sec 4.2.), it is the CFRule that captures the time used by the counterfactual system in building the rules.
  + Therefore, the time used in terms of the counterfactual systemis the sume of the runtime in CFRules and Reduction, which are occupying the most runtime for overall algorithm.
  + As such, most of the runtime is consumed by counterfactual operations, suggesting the need to improve the runtime of existing counterfactual systems.
* Number of Candidate Rules Explored (Figure 6):
  + According to the results, optimization of Genetic Rule CF and Greedy Rule CF leads to searching for candidates with a high probability of consistency while minimizing redundancy when the classifier is moderately complex (Credit and Adult), and when the classifier is complex, it leads to searching for candidates that are likely to be consistent. It guides you toward exploring many rules while preventing you from being stopped prematurely by them.
* The Number of GeCo Runs (Figure 7):
  + Since Geco takes up a significant portion of the runtime, the results demonstrate that GeneticRuleCF is more efficient than GreedyRuleCF for large datasets with complex classifiers.

1. LIMITATIONS AND FUTURE WORK

**Bound of Rule Components.** To simplify the search space, we've confined rule components to directly relate to input instance values. Thus, they can only be larger, smaller, or equal to the feature value, even if the actual rule might be broader. While this narrows the rule, it's effective for analyzing classifier behavior. We aim to generalize rules in future work.

**Realistic Feature Value Distributions**. While GeCo and other advanced Counterfactual Explanation models utilize perturbation distributions as the instance search space for interpretability, estimating these distributions accurately remains challenging, especially when representing causal dependencies between features. Improving this will benefit various explanation methods.

**Underlying Counterfactual Explanation System.** Our algorithm's stability and run time are greatly influenced by the underlying counterfactual system, GeCo. While GeCo is among the best we've found, it can be costly and unstable in extreme scenarios, impacting our system's performance. A more efficient counterfactual system would significantly enhance our systems.

**Better Counterfactual Explanation Model.** Our paper uses a counterfactual explanation model to generate rule-based explanations. A more advanced rule-based model could enhance our approach and potentially improve the counterfactual explanation process.

**Static Data and Classifier.** Our algorithms currently assume a constant data and classifier setup. Given their susceptibility to changes in either, we aim to develop explanations that can adapt to minor shifts in data or classifiers, aligning with the broader challenge of robust machine learning.

1. **CONCLUSION**

We've developed a method using counterfactual explanations to generate and ensure global consistency of rule-based explanations. We introduced a base algorithm, GeneticRule, and two advanced ones, GeneticRuleCF and GreedyRuleCF. Extensive tests validate our system's effectiveness.