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 such as 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 and 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, rule-based explanation is conjunctions of predicates on features consistently leading to certain outcomes and descriptively provides 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 and verified their efficiency experimental evaluation comparing with MinSetCover and Anchor.

2. DEFINITIONS

- $F_1, \ldots, Fn$ be n features, with ordered domains $dom(F_1), \ldots, dom(F_n)$.
- Define $\mathrm{I}_{\mathrm{NST}} = dom(F_1) \times \cdots \times dom(F_n)$.
- Call an element $x \in \mathrm{I}_{\mathrm{NST}}$ an *instance*.
- Given a black box classifier C, on any instance $x \in \mathrm{I}_{\mathrm{NST}}$, returns a prediction $C(x)$ within range $[0, 1]$
- Assume that $\mathrm{C}(\mathrm{x}) \leq 0.5$ is an "undesired" or "bad", while $\mathrm{C}(\mathrm{x}) > 0.5$ is "desired" or "good". For the binary classifier, replace the $\mathrm{C}(\mathrm{x})$ with each case as $\{0, 1\}$ respectively.

- Assume that a database $D = \{x_1, \ldots, x_m\}$ with m instances. It can be training or test set or data from historical decisions derived from the system.
- For every instance x_i in D, its feature values are given as $x_i = (f_{i1}, \ldots, f_{in})$.

2.1. Rule-Based Explanation

For a given instance xi = (fi1, ..., fin) $\in D$, a rule component $RC(xi)$, or RC relevant to xi, is a predicate of the form $Fj \leq fij$ or $Fj \geq fij$ for some feature Fj. A rule relevant to xi is a set of rule components, represented as $R = \{RC_1, \ldots, RC_c\}$ where cardinality of the rule is c. $R(x)$ is a predicate that is the conjunction of R, that is, $R(x) = RC_1 \wedge \cdots \wedge$ -RC_C. To indicate that a feature value $F_j = f_{ij}$, it is essential to incorporate both ≤ and ≥ rule components, meaning the cardinality constraint is $0 \leq c \leq 2n$. Denote I_{NSTR} the set of all instances that satisfy R, thus, $I_{NSTR} = \{x | x \in I_{NSTR}, R(x) = 1\}$.

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

(1) **Relevance**: Ensure $x_i \in I_{NSTR}$

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

(2) **Global Consistency**: $\forall x \in I_{NSTR}$, $C(x) \leq 0.5$.
(3) **Interpretability**: The rule must be simple with minimal cardinality.

Recognizing the computational demands of exhaustive global consistency checks, the paper advocates "Data Consistency". This approach evaluates consistency relative to a predefined database D = {x_1, …, x_m}, ensuring $\forall x_k \in D \cap I_{NSTR}$, $C(x_k) \leq 0.5$.

2.2. Counterfactual Explanation

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

x_{cf} must satisfy two main properties:

(1) Feasibility and Plausibility: Feasibility sets limits on potential feature values (e.g., income constraints), while plausibility dictates how new values in x_{cf} can differ from the values in x_i (e.g., gender shouldn't change). These criteria are encompassed by the PLAF (plausibility/feasibility) predicates:

$P(x_{cf}) = \Phi_1 \wedge \cdots \wedge \Phi_m \Rightarrow \Phi 0$, where Φ_i is a predicate over the features of x_i and x_{cf}.

(2) Magnitude of changes: The degree of changes between x_i and x_{cf} can be quantified using a distance function $dist(x, x')$.

A counterfactual explanation system takes as input an instance x_i, a PLAF constraint $P(x)$, and a distance function $dist(x, x')$. It produces a ranked list of counterfactuals based on their proximity to xi.

3. DUALITY

Rule-based explanations and counterfactual explanations offer distinct information. Although both prioritize brevity, a relevant rule should have few components, and a counterfactual should modify minimal features of x_i. 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. 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 explanation.

3.1. The Duality Theorem

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

Theorem 3.2 (Duality). Fix a globally consistent rule R relevant to x_i, let $x_{cf,1}, \ldots, x_{cf,k}$ be counterfactual instances, and let $R_{xcf,1}, \ldots, R_{xcf,k}$ be their duals. Then R is a set cover of $\{R_{xcf,1}, \ldots, R_{xcf,k}\}$. In other words, for every counterfactual xcf, m the rule R contains at least one rule component that conflicts with $x_{cf,m}$. Conversely, fix any counterfactual x_{cf}, and let $R_1, \ldots, R_k$ be globally consistent rules. Then the dual $Rxc\,f$ is a set cover of $\{R_1, \ldots, R_k\}$.

3.2. Using the Duality Theorem

The steps are as follows. First, construct the predicate $\mathrm{R}(\mathrm{x}')$ associated with the rule R. Next, utilize the counterfactual explanation system to identify a list of counterfactuals $x_{cf,1}, \ldots, x_{cf,k}$ for x_i that meet the PLAF predicate. If there are no counterfactual matches, then R is deemed globally consistent, and the process stops. Then, for each $j = 1$ to k, compute the dual $\mathrm{R}_{\mathrm{xcf,j}}$ for each counterfactual $x_{cf,j}$. Lastly, for every minimal set that encompasses R_0 of S, create the augmented rule $R \cup R_0$ and start again from Step 1.

4. ALGORITHMS

GeneticRule is a baseline algorithm that explores the space of rule-based explanations using a genetic algorithm. It does not use counterfactuals. GeneticRuleCF is an algorithm that extends GeneticRule by using an oracle call to a counterfactual explanation system to generate and validate rule-based explanations. GreedyRuleCF is an algorithm that 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.

4.1. GeneticRule

Algorithm 1: Pseudo-code of *GeneticRule*:
explain(instance x, classifier C, dataset D)

POP = $[\{F_1 \leq f_{i1}\}, \{F_1 \geq f_{i1}\} \ldots \{F_n \leq f_{in}\}, \{F_n \leq f_{in}\}]$
do
  CAND = **crossover**(POP, c) ∪ **mutate**(POP, m)
  POP = **selectFittest**(x, POP ∪ CAND, C, D, q, s)
  TOPK = POP[1 : k]
while $(\exists R \in TOPK : !\textbf{\textit{consistent}}(R, D, s)) \vee (TOPK \cap CAND \neq \emptyset)$;
return TOPK

GeneticRule is a "naïve" algorithm that generates rules using a genetic algorithm, shown in Algorithm 1. Inputs are Instance x, classifier C, and dataset D and output is a set of rules explaining instance x for classifier C. Five integer hyperparameters exist as follows: q is the positive number of rules that are kept after each iteration. k is the number of rules that the algorithm returns to the user, which is $k \leq q$. s is the number of samples taken from I_{NST} to check global consistency. m and c are the number of new candidates that are generated during mutation and crossover, respectively.

The steps of GeneticRule are as follows. First, it computes the initial population of rule candidates. The initial population has all possible rule candidates with exactly one rule component, and the initial candidates are likely not valid and consistent rules. Next, repeatedly applies follow: Mutate and crossover to generate new candidates, compute the fitness score via selectFittest for each candidate, then select the q fittest candidates for the next generation. Repeat until it finds K candidates that are consistent on both the dataset D and s samples from the more general I_{NST} space. Finally, check that 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.

The description of the internal operator is as follows. The mutate operator generates m new rule candidates for each candidate $R \in$ POP. Meanwhile, the crossover operator generates c new candidates for each pair of candidates, R_i and R_j. The selectFittest operator calculates the fitness score of each candidate in POP, sorts the candidates in descending order of their fitness scores, and returns the top q candidates.

GeneticRule only guarantees of the returned rules are global consistency for a sample of I_{NST}, thus only guaranteed the rules are data consistent.

4.2. GeneticRuleCF

Algorithm 2: Pseudo-code of *GeneticRuleCF*:
explain(instance x, classifier C, dataset D)

POP = $[\{F_1 \leq f_{i1}\}, \{F_1 \geq f_{i1}\} \ldots \{F_n \leq f_{in}\}, \{F_n \leq f_{in}\}]$
POP = POP ∪ **CFRules**($[\{\}], x, C, D$)
do
  CAND = **crossover**(POP, c) ∪ **mutate**(POP, m)
  CAND = CAND ∪ **CFRules**(POP, x, M, D)
  POP = **selectFittest**(x, POP ∪ CAND, C, D, q, s)
  TOPK = POP$[1 : k]$
  topk_consistent = ($\forall R \in$ TOPK :
    consistent(R, D, s) ∧ **consistentCF**(R, x, C, D))
while !*topk_consistent* ∨ ($TOPK \cap CAND \neq \emptyset$);
return TOPK

GeneticRuleCF applies a counterfactual explanation to GeneticRule to generate and verify rule candidates, shown in Algorithm 2.
Two functions involve a counterfactual explanation as follows. First, CFRules is a function that takes as input a set of rule candidates and generates new candidates by computing the counterfactual explanations for each input candidate. Second, consistentCF is a function that verifies global consistency of the top-rule candidates using a counterfactual model.
Performance optimizations exist to maximize performance with minimal effects. The algorithms have reduced the calling of the counterfactual explanation model, by only running CFRules once for every three iterations 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, which is not shown in the pseudo code, ensures that the returned rules have no redundant components like as follows.

4.3. GreedyRuleCF

Algorithm 3: Pseudo-code of *GreedyRuleCF*:
explain (instance x, classifier C, dataset D)

POP = **CFRules**($[\{\}], x, C, D$)
POP = **sortByCardinality**(POP)
while (!***consistentCF***(*POP[1]*, x, C, D)) **do**
  `// get and remove the top candidate from POP`
  top_cand = **pop**(POP)
  `// generate new candidates only for top_cand`
  CAND = **CFRules**([top_cand], x, C, D)
  POP = **sortByCardinality**(POP ∪ CAND)
  POP = POP[1 : q]
end
return POP[1]

GreedyRuleCF greedily finds rule candidates with small cardinality by repeatedly utilizing the underlying counterfactual explanation, shown in Algorithm 3.
It generates the initial population by running CFRules, keeping them sorted in ascending order of their cardinalities. Then, 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 end with a consistent rule by having at most 2n rule components, where n is the number of variables in D.

4.4. Fitness Score Function

selectFittest is a function that computes the fitness score to rank the rule candidates. The fitness score is based on its degree of consistency and its cardinality. There exist 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 assumption for fitness score of a rule is like the following:

- 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 violate R.
- If VD = 0, use sample s instances from INST, and define VS the number of samples that violate R.

The fitness score - score(R):

$$score(R) = \begin{cases} 0.25 \times (1 - \frac{|R|}{2 \cdot n}) + 0.25 \times (1 - \frac{VD}{m}), & FDC \\ 0.25 \times (1 - \frac{|R|}{2 \cdot n}) + 0.25 \times (1 - \frac{VS}{s}) + 0.25, & FGC \\ 0.25 \times (1 - \frac{|R|}{2 \cdot n}) + 0.75. & GC \end{cases}$$

5. EXPERIMENTS

5.1. Experiment Setup

DataSets and Classfiers. Consider four real datasets.

(1) Credit Dataset: used to predict the default of the customers on credit card payments in Taiwan.
(2) Adult Dataset: used to predict whether the income of
adults exceed $50K/year using US census data from 1994.
(3) FICO Dataset: used to predict the credit risk assessments.

(4) Yelp Dataset: used to predict review ratings that users give to businesses.

	Credit	Adult	Fico	Yelp
Number of Instances	30K	45K	10.5K	22.4M
Features	14	12	23	34
Classifier Type	Decision Tree	Decision Tree	Neural Network	Neural Network

Table 1: Key Characteristics for each Real Dataset.

Table 1 illustrates the key statistics and the classifier type used for each dataset.

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 test if the systems can recover the rules when the 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 is applied for all systems.

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. The algorithms benchmark the algorithms against two existing systems, Anchor and MinSetCover. Anchor is a system that generates rule-based explanations (i.e., anchors) by the beam-searched version of the pure-exploration multi-armed bandit problem. MinSetCover is a system that generates rule-based exploration using the minimum set cover problem.

Parameter Choice. The five hyperparameters are as follows. K is the number of rules that the algorithm will return. Q is the number of rules to be kept in each iteration. M is the number of new candidates generated during mutation. c is the number of new candidates generated during the crossover. S is the number of samples from I_{NST} during evaluation.

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 are pre-processed as required by the classifiers. 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 I_{NST} to provide this explanation to one instance. Therefore, if the system returns multiple explanations, retain the top-ranked rule only. It is needed to measure the run time to find the explanation and then evaluate its quality. The test repeats the above process for 10,000 users (i.e., 10,000 explanations). Thus, in our experiments, each system returns 10,000 rules.

Evaluation Metrics. Two metrics were applied to evaluate the quality of the global consistency and interpretability of the generated rules. Global consistency means that the classifier has not found any of the "desired" rules. To ensure the interpretability, the evaluation checks 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.

5.2. Quality in terms of Consistency and Interpretability

Compare all algorithms considered in terms of the quality of the rules generated on the datasets. First, consider synthetic classifiers, and then evaluate the systems considered on real classifiers.

Synthetic Classifiers. This form of 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.

The ratio of the three categories for each algorithm is illustrated in Figure like below:

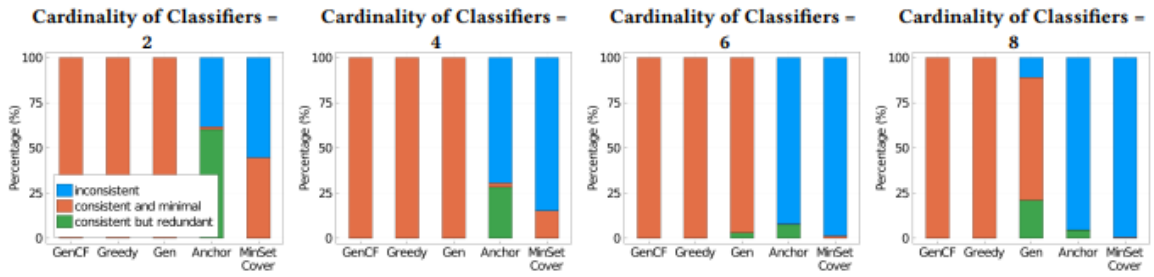

Figure 1: The break down of the percentage of rules that are consistent and minimal, consistent with redundant components, and inconsistent for *GeneticRule*(Gen), *GeneticRuleCF*(GenCF), *GreedyRuleCF*(Greedy), Anchor, and MinSetCover over 1000 synthetic classifiers with 2, 4, 6, and 8 rule components for the Credit Dataset.

The rules of GeneticRuleCF and GreedyRuleCF are always consistent and minimal regardless of the cardinality of the classifiers. GeneticRule returns partial inconsistent when the cardinality of classifiers increases to 8. Suggests the need to include a counterfactual explanation system. In case of 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 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 minimal. We used Geco as a proxy for testing global consistency via searching of any existence of counterfactual models.

Figure 2 illustrates the evaluation for each dataset with their specific classifiers as follows:

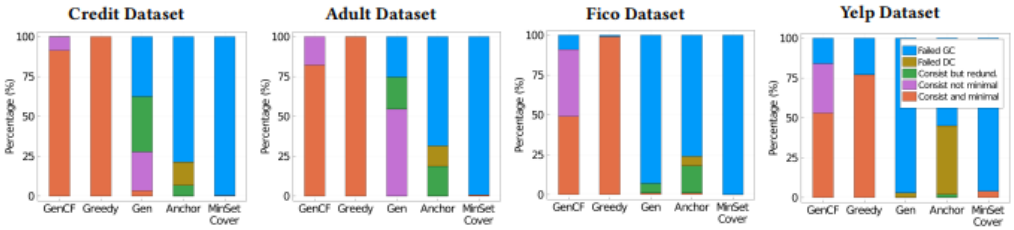

Figure 2: The break down of the percentage of rules that are not global consistency (Failed GC), not data consistency (Failed DC), consistent with redundant components, consistent with redundant components but not minimal (consist not minimal), and consistent and minimal for *GeneticRule* (Gen), *GeneticRuleCF* (GenCF), *GreedyRuleCF* (Greedy), Anchor, and MinSetCover. We explain 10000 instances for the Credit, Adult, and Fico dataset, and 100 instances for the Yelp dataset.

GeneticRuleCF and GreedyRuleCF 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 does not guarantee the search for globally consistent rules and has the limitation of only utilizing samples from INST. Therefore, this algorithm requires a counterfactual expansion system. Thease algorithms are superior both in that they contain fewer redundant components than Anchor and in that they guarantee extensive consistency over MinSetCover's limited data consistency.

5.3. Runtime Comparison

Measures the runtime of all algorithms considered 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. The comparison of runtime for the synthetic classifiers is as shown in Figure 3:

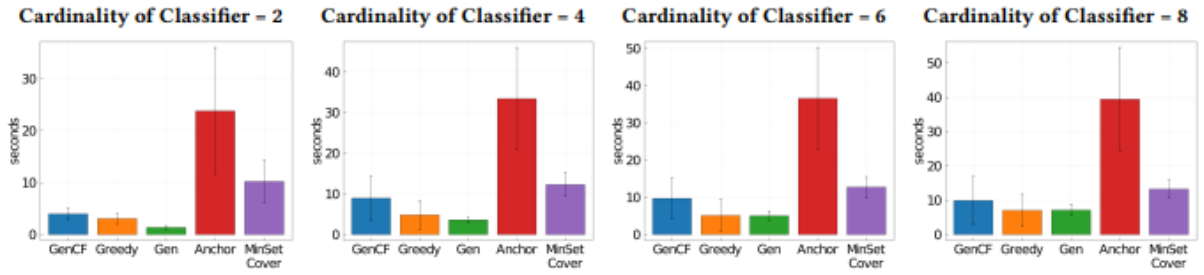

Figure 3: Comparison of runtime for *GeneticRule* (Gen), *GeneticRuleCF* (GenCF), *GreedyRuleCF* (Greedy), Anchor, and MinSet-Cover over 1000 synthetic classifiers with 2, 4, 6, and 8 rule components for the Credit dataset

GeneticRule, GeneticRuleCF, and GreedyRuleCF usually consume less time than Anchor and MinSetCover, regardless of the cardinality of the 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. The comparison of runtime for the real classifiers is as shown in Figure 4:

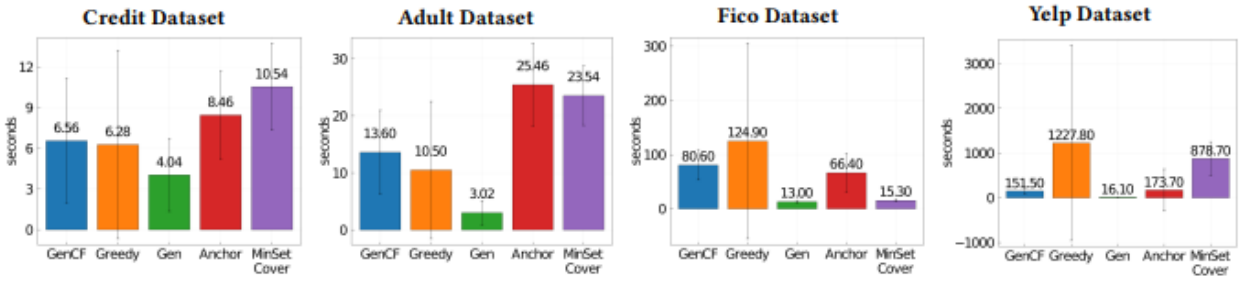

Figure 4: Comparison of the average runtime (in seconds) of rules for *GeneticRule*, *GeneticRuleCF*, *GreedyRuleCF*, Anchor, and MinSetCover. We explain 10000 instances for the Credit, Adult, and Fico dataset, and 100 instances for the Yelp dataset.

For the Credit dataset, the runtimes of all algorithms are similar to the dataset, suggesting that the algorithms can efficiently generate consistent rule-based descriptions without any additional cost for moderately complex datasets and classifiers. For the Adult dataset, the algorithms designed are much faster than the existing algorithms, and this difference in performance is due to the characteristics of the algorithms that can handle hot encoding as a single function. In the case for Fico and Yelp datasets, 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.

Table 2 shows the contribution of the classifier and Geco to the runtime of each dataset as follows:

	Credit	Adult	Fico	Yelp
Classifier Runtime	0.0041	0.0079	0.1079	0.02081
GeCo Runtime	0.0854	0.1294	1.9050	2.0793

Table 2: Run time of the classifiers to predict 10,000 instances, and for GeCo to explain a single instance on each dataset.

The Yelp dataset has a relatively large number of instances and uses more complex classifiers, making the creation of rules 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 is fast when the classifier and the dataset are moderate in size. For complex classifiers over large datasets, however, GeneticRuleCF is more efficient than GreedyRuleCF.

5.4. Microbenchmarks

The summary of microbenchmarks is illustrated in Figure 5 to Figure 7 as follows:

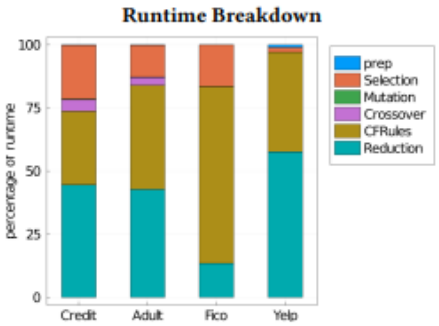

Figure 5: The break down of average run time into the main operators for *GeneticRuleCF* algorithm.

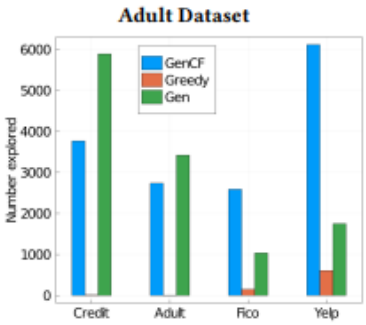

Figure 6: The number of rule candidates explored for *GeneticRuleCF*, *GeneticRule* and *GreedyRuleCF*.

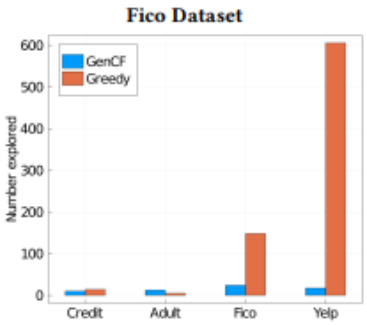

Figure 7: The number of rule candidates explored by GeCo for *GeneticRuleCF*and *GreedyRuleCF*.

Breakdown of Runtime (Figure 5): Note that the GreedyRuleCF excluded since it uses the counterfactual system as GECO only. Prep (i.e., Preparation) spends its runtime computing the rule space and building 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 system is the sum of the runtime in CFRules and Reduction, which occupy 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 the search 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 the search 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 occupies a significant portion of the runtime, the results demonstrate that GeneticRuleCF is more efficient than GreedyRuleCF for large datasets with complex classifiers.

6. LIMITATIONS AND FUTURE WORK

Bound of Rule Components. To simplify the search space, we have confined rule components to directly relate to input instance values, but actual rule might differ.

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.

Underlying Counterfactual Explanation System. Our algorithm's stability and run time are greatly influenced by the underlying counterfactual system.

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. The algorithms currently assume only a constant classifier and classifier setup.

7. CONCLUSION

We have 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 the effectiveness of our system.