AutoAction: Automated Generation of Actionable Query Explanations

ABSTRACT

The vision to enable automated generation of *actionable* insights from data is intriguing but also challenging. This paper seeks to pursue this vision in the query explanation setting. Given an unexpected query result (e.g., the average grade of students is too low), the query explanation problem aims to automatically generate an explanation to resolve the unexpected query result. Existing provenance-based explanation methods identify a group of records to delete from an input table. However, this deletion-based output is often not actionable in practice (e.g., asking a group of students to leave the school is not possible). Inspired by the counterfactual explanation from the ML community, we propose AutoAction, an actionable query explanation framework, which aims to automatically generate an actionable explanation by identifying the updates to a group of records (e.g., encouraging a group of international students to change their study habits). There are a number of challenges to develop AutoAction: i) how to design an effective objective function to quantify different explanations, ii) how to devise an efficient algorithm to search for the best explanation, and iii) how to make sure the generated explanations satisfy real-world constraints. Novel solutions are proposed to address these challenges. We conduct extensive experiments with real-world and synthetic datasets and did a user study with 20 participants. The results show that AutoAction can automatically generate useful actionable explanations more efficiently than baselines.

1 INTRODUCTION

Recent years have witnessed a growing interest in developing automated solutions for a variety of data-science tasks, such as AutoML for automatically building machine learning models [10], AutoInsight for automatically discovering interesting data patterns (e.g., outlier, trend) [7], AutoEDA for automating exploratory data analysis [21], and AutoEM for automating entity matching model development [34]. These automated solutions are appealing because they not only make data science more accessible for non-technical people but also enable data scientists to work more efficiently [33].

Despite that automated solutions are available for many data-science tasks, there is still a lack of automation in generating *actionable* query explanations. Query explanation is a well-studied topic in the database community and plays an important role in data-driven decision making [19]. To explain an unexpected query result, existing work typically adopts a method called provenance-based explanation. It returns a group of records such that if they were removed from the input table, the query result would meet the expectation. In many situations, however, a user cannot easily translate a group of deleted records into a practical action. To overcome this limitation, this paper proposes actionable query explanations and studies how to automatically generate them.

	Provenance-based Explanation	Counterfactual Explanation	Actionable Query Explanation
Input	Query	X	Query
	Table	**Single record**	Table
	X	Model	Model
Output	**Delete**	Update	Update
	A group	**Single record**	A group

Figure 1: Comparison of different explanation frameworks

Example 1 (Provenance vs Actionable Explanations). *Consider a table in Table 1. Suppose that a data scientist runs an aggregation query to compute the average grade of primary school students and finds it lower than expectation. Using provenance-based explanation, she can identify a group of students (e.g., all international students from Class 1) that have low grades such that if they were removed from the table, the low-grade issue would be resolved. However, this is not an actionable explanation since in reality, she cannot ask these students to leave the school. A possible actionable explanation is to encourage these students to change their study habits (e.g., visit course content and raise hands more frequently) such that if this action was taken, the low-grade issue would be resolved.*

Generating actionable explanations is challenging because it requires estimating the impact of updating some columns on another column. For the above example, we need to estimate the impact of updating Visited and Raised on Grade. Fortunately, a similar problem, named *counterfactual explanation*, has been recently studied in the ML explanation community [13, 22, 29, 32]. Given an ML model and a single record, if the model has an unexpected predication on the record, counterfactual explanation can generate an action (i.e., recommending some changes to input features) such that if the changes were made, the updated record would have a desired prediction.

Example 2 (Actionable Explanation on Single Record). *Consider an ML model which predicts whether a student has a high grade or not based on features like how many times a student raises his/her hand in class (Raised), how many times a student visits course content (Raised), etc. Suppose that a student (Raised = 2, Visited = 0) is predicated to have a grade of 60, and she would like to seek a recommendation on what action to take in order to improve the grade from 60 to 70. Counterfactual explanation can return an action like increasing Raised by 3 and Visited by 2. That is, if the student had raised hands by 2+3=5 times and visited course content by 0+2 = 2 times, she would receive a grade of 70 according to the ML model.*

Figure 1 summarizes the differences between three explanation frameworks. Provenance-based explanation takes a query and a table as input, and uses a group of deleted records to explain the query. However, this deletion-based output is typically not actionable. Counterfactual explanation takes a model and a single record as input and uses the updates to the single record to explain the model. While this update-based output is actionable, it does not explain a query. Our actionable query explanation aims to explain a query and return the updates to a group of records.

Example 3 (Actional Query Explanation). *Given an unexpected query result in Example1, and a model in Example 2, actionable query explanation can return a group of students (e.g., From = "International" and Class = 1) and the updates (e.g., increase Raised by 3 and Visited by 3). It means that if the updates were made to the group, this group of students would receive higher grades according to the model, leading to an expected query result.*

We propose AutoAction, a novel actionable query explanation framework. Due to the differences from existing frameworks, AutoAction faces a number of new challenges.

How to design an effective objective function? We need to design an objective function to rank different explanations in order to identify the best one. Our objective function considers three factors: cost, effect, fairness. The cost quantifies the difficulty to take an action. For example, asking 1 students to increase Raised by 10 is harder than by 2. Thus, the former has a higher cost than the latter. The effect quantifies the impact of action. For example, it can quantify the increase of the average grade after asking a group of students to increase Raised by 2. The fairness quantifies how fair an action is. We adopt a common definition of fairness and incorporate it into our objective function.

How to develop an efficient search algorithm? The counterfactual explanation problem is to find the best updates on a single given record. This is already a challenging and computationally expensive problem. Our problem faces even more challenges. First, the group is not given and it needs to search for the best group (e.g., From = "International" and Class = 1). Second, once a group is identified, it needs to search for the best updates on that group (e.g., increase Raised by 2 and Visited by 3). We model this problem as a multi-armed bandit problem, where each group is an arm. We first implement the state-of-the-art approach, Hyperband [16], for our problem. However, it does not work well since different groups have different size and directly applying Hyperband suffers from unable running time. To overcome this limitation, we propose AutoAction, a simple but highly effective search algorithm by treating Hyperband as a group selection component and running it for multiple times in the whole search process.

How to support different types of constraints? We observe that AutoAction sometimes generate an action that is not valid because it violates real-world constraints (e.g., asking a student to raise hands by a negative number is impossible). We identify three types of constraints observed in the real world: i) Action-based Constraint, which takes an action as input and checks whether it is valid, ii) Instance-based Constraint, which takes an action and a single record as input and check whether applying the action to the tuple is valid, iii) Approximate Instance-based Constraint, which takes an instance-based constraint, a set of records, and a threshold as input, and requires the percentage of the records that satisfy the constraint is smaller than the threshold. We formally define these constraints, and propose two approaches in AutoAction to support them.

In summary, we make the following contributions:

Table 1: Sample rows of the running example

Student	From	Class	Raised	Visited	Grade
Jack	Domestic	1	3	4	70
Mark	International	1	2 -> 3	3 -> 4	60 -> 70
Emma	International	1	3 -> 4	3 -> 4	70 -> 90
Sophia	International	1	5 -> 6	3 -> 4	80 -> 80
James	Domestic	2	1	1	50
Olivia	Domestic	2	2	1	60
Lucas	International	2	4	1	70
Luna	Domestic	3	5	4	80
...	...	...	...	...	...

- We are the first to study how to automatically generate actionable query explanations. We design an objective function and define the actionable query explanation problem.
- We model our problem as a multi-armed bandit problem, and identify the limitation of the state-of-the-art solution. To overcome the limitation, we propose AutoAction, a new search algorithm to solve our problem.
- We identify three types of constraints (action-based, instance-based, approximate instance-based) and discuss how to extend AutoAction to support them.
- We conduct experiments using both real-world and synthetic datasets, and did a user study with 20 participants. The results show that AutoAction is both efficient and effective compared to baselines and the automatically generated explanations are useful to users.

2 PROBLEM DEFINITION

We formally define the actionable query explanation problem.

Supported Query. Let R denote a table with an attribute set of A. The supported queries are in the following form:

```
SELECT agg(a) FROM R
WHERE P_1 AND/OR ... AND/OR P_p
```

where a is an aggregated numeric attribute, agg is SUM, COUNT, or AVG, and P_i (for each i $\in [1, p]$) is a predicate. We assume that for any P_i, P_i is not defined on attribute a. For remaining of the paper, we assume that agg is SUM, and it is trivial to extend the framework to other aggregate functions.

Example 4. *Back to Example 1, the data scientist runs the following SQL query to get the average grade of primary school students:*

```
SELECT AVG(Grade) FROM Students
WHERE school = 'primary'
```

Given a supported query and a table R, let D denote the subset of records from R that satisfy the query's predicate. For the above example, D is the subset of all primary school students. Table 1 shows some sample rows from D.

Attribute Categories. We classify attributes into four categories, which are aggregation attribute a, group attributes A_g, actionable attributes A_a and the other attributes A_o. These four subsets of attributes are disjoint, which yields that $A = \{a\} \bigcup A_g \bigcup A_a \bigcup A_o$.

(a) Aggregation Attribute. It is the attribute that the aggregate function of input query is performed on. For our problem setting, it should be an attribute with numeric values (discrete or continuous).

Users aim to improve the values of this attribute. In Table 1, "Grade" is the aggregation attribute.
(b) Group Attributes. To explain a query, a predicate is often returned as an explanation [27, 35]. Group attributes are the attributes involved in the predicate. Formally, we define a set of m discrete attributes $a_g^1, ..., a_g^m$ as group attributes. An instance of m values over group attributes indicates a predicate $\{a_g^1 = v_g^1, ..., a_g^m = v_g^m\}$ for a subgroup of records in D. Group attributes determine a search space, in which we search a group to apply an action on. We will discuss it in details later.

Example 5. *In Table 1, suppose that the group attributes are "From" and "Class". An instance of group attributes is {From = 'international', Class = '1'}, which indicates all international students from Class 1 (i.e., the red records in Table 1).*

(c) Actionable Attributes. Actionable attributes are defined as a set of continuous numeric attributes $a_a^1, ..., a_a^n$ that can take an action. We assume that for any predicate P_i in input query and actionable attribute a_a^j, P_i is not defined on a_a^j.

Definition 1 (Action). *Given a group g which is a subset of rows and n actionable attributes $a_a^1, ..., a_a^n$, an action is defined as a tuple (g, s), where s is a n-dimension vector $s = (s_1, ..., s_n)$. The application of an action on group g is defined that for each record $r \in g$, r is updated to r^s that for each value of actionable attribute $r.a_a^i$, we have $r^s.a_a^i = r.a_a^i + s_i$.*

Example 6. *In Table 1, suppose that the actionable attributes are "Raised" and "Visited", which count the number of times a student raises hands and visits online courses per day. Consider an action (g, s) where g is {From = 'International', Class = '1'}, and $s = (1, 1)$. We update "Raised" and "Visited" by 1, respectively, for the international student from Class 1. The arrows in Table 1) indicate the updates on "Raised" and "Visited".*

Objective Function. In order to identify the best action, we need to design an objective function to quantify an action. We consider three factors in our design: effect, cost, and fairness.
(a) Effect. Effect indicates the change of the aggregation value after an action is applied on a particular group. When actionable attributes are updated, due to causal relationship between variables, the aggregation attribute will be changed as well. Typically, the counterfactual value of an aggregation value is not accessible in real scenarios. Thus, for our method, we train a model over D to get its estimated value.

Definition 2 (Inference Model). *An inference model $M(\cdot)$ is fit on D, which accepts an individual record r from D, and predicts the value of the aggregation attribute a of this record, i.e., $r.a$.*

Definition 3 (Effect). *Given an action (g, s) and an inference model M then the estimated effect of the action on g is defined as*

$$\Delta E_g^s = \sum_{r \in g} M(r^s) - \sum_{r \in g} r.a$$

The ground-truth effect is represented as $\Delta E_g^s = \sum_{r \in g} r^s.a - \sum_{r \in g} r.a$, where $r^s.a$ can not be gotten directly.

When $\Delta E_g^s > 0$, it implies that after the action is applied on group g, the aggregation value over a will be improved according to estimated result of model M. As an indicator for action effect, E_g^s denotes the change on output of the aggregate query.

Example 7. *For Table 1, the arrows in Grade indicate the change over the aggregation attribute after the action is applied. The effect value can be calculated as in Definition 3, which yields $\Delta E_g^s = (70 + 90 + 80) - (60 + 70 + 80) = 30$.*

(b) Cost. Cost can be regarded as a balance to effect, since the action with too much cost is not applicable and not insightful, although it can provide much effect. For counterfactual explanation, l_p norm has been studied and identified as a feasible metric to quantify the cost of an action [12]. Adopting a similar idea, we use the l_2 norm as the cost measure.

Definition 4 (Cost). *Given an action (g, s), its cost is defined as*

$$C_g^s = |g| \cdot || \sum_{i=1}^{n} s_i ||_2$$

We multiply the l_2 norm of an action vector by the group size, since this action will be applied to every record in the group.

Example 8. *For Table 1, as the action vector is $s = (1, 1)$ and it is applied to a group of three records, the cost is calculated as $C_g^s = 3 \cdot \sqrt{1^2 + 1^2} = 4.24$.*

(c) Fairness. ML fairness has been extensively investigated recently to avoid bias in AI systems [18]. Since we consider the fairness of applying an action to a particular group, it is different from the focus of ML fairness which reduces the bias of a predictive model. Intuitively, we tend to give the action to the group who needs the action the most. For our running example, this is the group whose average grade is the lowest.

Definition 5 (Fairness). *Given an action (g, s), the population D, and the aggregation attribute a, the fairness of the action is defined as*

$$\delta = \frac{1}{|D|} \sum_{r \in D} r.a - \frac{1}{|g|} \sum_{r \in g} r.a$$

Example 9. *For Table 1, the fairness for the highlighted group can be computed as $\delta = \frac{70+60+70+80+50+60+70+80}{8} - \frac{60+70+80}{3} = -2.50$, which is the difference between the average grades of the selected group and the population.*

We justify the effectiveness of this fairness definition in our user study. There are certainly other ways to define fairness, which we defer to future work.
(d) Combing Cost, Effect, and Fairness. We construct an objective function to quantify the quality of an action by combining cost, effect, and fairness together.

Definition 6 (Objective Function). *Given an action (g, s), the objective function $f(g, s)$ is given as*

$$f(g, s) = \lambda \cdot min(-\Delta E_g^s, 0) + (1 - \lambda) \cdot C_g^s - \beta \cdot \delta$$

In the objective function, λ and β are user-specified parameters, allowing the user to balance the trade-off between the three factors. According to previous discussion, for all three components in the objective function, when its value gets smaller, the action will have better quality. So the overall target would be minimizing the objective value, which yields our optimization problem.

Actionable Query Explanation. The following formally defines the actionable query explanation problem.

Definition 7 (Actionable Query Explanation). *Given a table D with an aggregation set of $A = \{a\} \bigcup A_g \bigcup A_a \bigcup A_o$, a supported query, and an inference model M fit on D with attribute a as target, the goal of actionable query explanation is to find the best action (g^*, s^*) that minimizes the objective function:*

$$(g^*, s^*) = \arg\min_{(g,s)} f(g, s)$$

3 SEARCH ALGORITHM

In this section, we first discuss how to solve our problem when a group is given, and then propose AutoAction in a general setting.

3.1 Optimization on Single Group

Suppose that there is only one group involved, which means that we do not need to decide which group should take an action.

Derivative-free Optimization. Recall that the definition of the objective function accepts an inference model M, which is not necessarily differentiable over the action variable. General gradient-based approaches do not work for such setting [5, 25]. Thus, we seek for a derivative-free optimization approach.

COBYLA. We adopted a popular derivative-free optimization technique, called Constrained Optimization BY Linear Approximations (COBYLA) [23, 24]. COBYLA utilizes polynomial linear interpolation to approximate the objective function value in a trust region, and does not require the objective function to be differentiable. A bench-marking work has shown that COBYLA can usually achieve the best performance compared with other local solvers when the target is easy, it even outperforms the virtual best solver under some small budget setting. [30] It has been successfully applied to solving a variety of derivative-free optimization problems [6, 9, 31]. If we execute COBYLA for one run, it may step into a local minimum after a few iterations and keep optimizing it until the budget is exhausted. This issue can be mitigated by running COBYLA for multiple times (3 by default).

3.2 Optimization on Multiple Groups

Brute-force Solution. In a general setting, there will be over than one groups. A brute-force solution is to iterate each group one by one, apply the above optimization to each group, and then return the action with the best objective function value. If there are hundreds of groups, this method will run COBYLA for hundreds of times, which is highly time-consuming.

Note that a query explanation system should be able to respond interactively (e.g., in less than 30 s). This is because that there could be different inputs to the system (e.g., different queries, different parameters) and the user would like to interactively see the output actions by trying different inputs. Obviously, adopting the above brute-force approach will not meet this need.

Model as Multi-Armed Bandit. Multi-armed bandit problem has been well studied as a reinforcement learning problem [3, 4]. It has

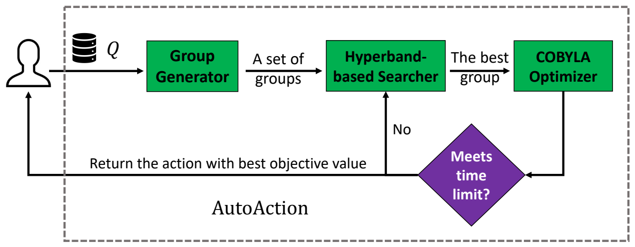

Figure 2: AutoAction Workflow

multiple *arm*, where each arm is associated with a random reward. Pulling an arm collects a reward from the arm. The objective is to come up with a pulling strategy to maximize the sum of rewards. It is necessary to keep a good balance between exploration and exploitation, which is about the decision whether to pull more arms or stick to the current best arm.

For our problem, each group can be considered as an arm. Once the action vector and the group of an action are fixed, the reward of the action is independent with the choice of our pulling strategy. Thus, this bandit problem is non-stochastic. The identification of the best arm under non-stochastic setting has been established in the ML community [11].

(a) Hyperband. Hyperband [16] is the state-of-the-art algorithm from the multi-armed bandit literature that can solve our problem. It is a bandit-based approach to conduct hyper-parameter selection for ML models. It denotes the computation resource as a total budget R, and the number of tried configurations (i.e., pulled arms) as n. There is a balance between n and the average budget for each configuration $\frac{R}{n}$. When n increases (decreases), it will try more (less) configurations with less (more) budget allocated to each configuration. Intuitively, Hyperband uses the early stopping idea to quickly filter out those configurations which are less likely to be the best choice. It uses an efficient selection subroutine called Successive Halving [11]. For the subroutine, at every iteration, it runs each configuration for a small number of iterations and early terminates bad configurations. Finally, the best result seen so far is returned.

(b) Why not Hyperband? We first present how to apply Hyperband to solve our problem and then discuss its limitation.

As every group is a configuration choice, we can directly apply Hyperband to search for a group g in D. By the definition of Hyperband, a candidate subset of groups is selected at first. Then, for each iteration of Hyperband, we iterate through those groups, run the

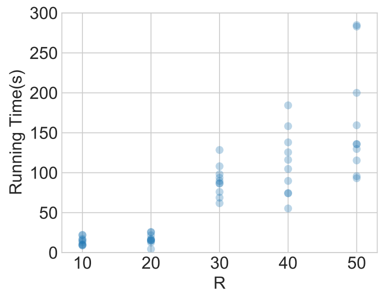

Figure 3: Running time of different trials of Hyperband by varying R.

COBYLA based optimizer on individual group with early stopping, where the maximum number of iteration of COBYLA is dependent on the budget allocated for this iteration. Before starting next iteration, we discard the groups with worst intermediate objective value. When all iterations of Hyperband are finished, we output the group g with best intermediate objective value seen so far. For next step, we apply COBYLA based optimizer on g to get the action vector a, then return the action (g, s) as the final output.

Hyperband has proved to be effective in the hyper-parameter tuning setting. However, unlike hyper-parameter tuning, the search process of our problem is dependent on the size of group. The size of different groups may vary a lot. According to our experiments, some groups could be orders of magnitude larger than other groups. As a result, given a time budget (e.g., 100s), it is hard to figure out how to set the budget R for Hyperband. To illustrate this point, we varied R from 10 to 50 at a step size of 10, and executed Hyperband on the Adult dataset for 10 times. The result of each trial is represented as a dot in Figure 3. We can see that when setting $R = 40$, the running time of Hyperband varied from 50s to 200s, which is highly unstable. In our experiment, we have tried to build a relationship between R and the average running time of 10 runs, and set the budget R based on this relationship. Due to the high variance in running time, this approach still did not perform well (see Section 5.4 for detail).

AutoAction Search Algorithm. A simple way to overcome the limitation of Hyperband is to adopt the *random search* algorithm. Random Search supports the user-specified time limit naturally. It randomly selects a group and computes its objective value using COBYLA. This process repeats until the time limit is reached. Finally, the action with the best objective value is returned. While random search can overcome Hyperband's limitation, it does not leverage the early stopping idea to speed up the search process. If a randomly selected group is not good, it will waste a lot of time on that group.

Both Hyperband and random search have advantages and limitations. To this end, we propose our AutoAction search algorithm, which combines Hyperband and random random to achieve the best of both worlds.

(a) Main Idea. The main idea of AutoAction is to follow the random search framework but treat Hyperband as a group-selection component in the framework. Compared to Hyperband, AutoAction runs Hyperband for multiple times, where each time is allocated with a small budget. Compared to random search, AutoAction uses Hyperband (rather than random selection) to avoid selecting a bad group.

(b) Algorithm Description. The workflow of the AutoAction algorithm is presented in Figure 2. It takes a query, a table, and a time limit as input, and returns the best action under the time limit. It first generates a set of groups and then iteratively selects a group using Hyperband and computes its objective value using COBYLA. We set a very relatively small budget for Hyperband since its purpose is for group selection. Finally, when the time limit is met, we return the action with the best objective value.

AutoAction preserves the nice property of single-round Hyperband algorithm to incorporate early stopping. At the same time, by allocating the total budget of single-round Hyperband to multiple cheaper executions of Hyperband, it avoids relying on just one single Hyperband, which would lead to unstable running time.

Algorithm 1: AutoAction

```
1  AutoAction $(R, Q, A, T)$;
   Input  : Table $R$, query $Q$, attribute set $A$, time budget $T$
   Output : action $(g, s)$
2  Train a prediction model $M$ on $D$;
3  Initialize population $D$, total running time $t = 0$, initial
     action $(g, s) = (g_0, s_0)$, objective value $v = f(g_0, s_0)$,
     iteration index $i = 1$;
4  while $t < T$ do
5      $g_i = Hyperband\_Search(D, A, f, M)$;
6      $s_i = COBYLA\_Optimize(D, A, f, M, g_i)$;
7      $i = i + 1$;
8      $t = t + running\_time(i)$;
9      if $f(g_i, s_i) < v$ AND $t <= T$ then
10         $v = f(g_i, s_i)$;
11         $(g, s) = (g_i, s_i)$;
12     end
13 end
14 return $(g, s)$;
```

4 SUPPORTING CONSTRAINTS

AutoAction may generate an action that violates domain constraints, thus not providing enough insights to users. Back to Example 6, if the action which minimizes the objective function requires a student to visit online courses for 100 times a day, it does not apply to real-world scenarios. In this section, we discuss how we support real world constraints in AutoAction.

4.1 Different Types of Constraints

We propose a general framework to cover different types of constraints in real world. The formal definition is given below.

Definition 8 (Constraint). *Given an action (g, s), a constraint c is identified as a function $F_c = f_c(g, s)$, when $F_c <= 0$, the action (g, s) does not violate the constraint c, otherwise this action is illegal.*

According to the definition, the violation to a constraint just depend on the action after the group is fixed.

Action-based Constraint. Action-based constraints are defined on value of actions directly. For example, the improvement of the salary of a person can not be more than \$1000 a month.

Definition 9 (Action-based Constraints). *Given an action (g, s) and n values of actionable attributes $s.a_a^1, ..., s.a_a^n$, an action-based constraint c is defined as an $F_c = \alpha_1(s.a_a^1)^{c_1} + ... + \alpha_n(s.a_a^n)^{c_n} + b$, where $\alpha_1, ..., \alpha_n, c_1, ..., c_n$ are integer values and b is a constant.*

Instance-based Constraint. Instance-based constraints check the legality of tuples in a group after the given action is applied. For example, a student should at least watch 10 online course videos per week, after some action takes place. It is also common in real world that the limit over some attributes is dependent on the condition over other attributes. For example, if a student reads more than 50 pages of textbooks per week, she can watch at least 5 online course videos per week.

Definition 10 (Instance-based Constraints). *Given n values of actionable attributes of a record r after applying action (g, s), which is split to $(r^s.a_a^1, ..., r^s.a_a^m)$ and $(r^s.a_a^{m+1}, ..., r^s.a_a^n)$. An instance-based constraint c is defined as $F_c = \sum_{r\in g} max(f_c(r^s), 0)$, where*

$$f_c(r^s) = \begin{cases} f_m(r^s.a_a^1, ..., r^s.a_a^m), & \text{if } f_{n-m}(r^s.a_a^{m+1}, ..., r^s.a_a^n) <= 0 \\ 0, & \text{otherwise} \end{cases}$$

Both f_m and f_{n-m} have the same form with f in Definition 9 that they are polynomial of $(r^s.a_a^1, ..., r^s.a_a^m)$ and $(r^s.a_a^{m+1}, ..., r^s.a_a^n)$. When m is equal to n, there is no condition and $f_c(r^s) = f_m$.

This type of constraint implies that for every tuple t in group g, after applying action, t should satisfy $f_m <= 0$ when $f_{n-m} <= 0$, otherwise the constraint will be violated. When the m is equal to n, the instance-based constraint is reduced to similar form as the action-based constraints.

Approximate Instance-based Constraint. In practice, it is very common that if a little fractions of tuples in a group violate an instance-based constraint, it can be tolerated. As long as the majority do not violate the constraint, we consider the action is legal. Thus, we further define approximate instance-based constraint to indicate those situations.

Definition 11 (Approximate Instance-based Constraints). *Given an action (g, s), an instance-based constraint c and $f_c(r^s)$, an approximate instance-based constraint c^{appro} is defined as $F_c = \frac{\sum_{r\in g} n(r^s) - \sum_{r\in g} n(r)}{|g|} - \epsilon <= 0$, where*

$$n(r) = \begin{cases} 1, & \text{if } f_c(r^s) > 0 \\ 0, & \text{otherwise} \end{cases}$$

ϵ is a user defined threshold parameter, which indicates the tolerance how many more tuples violating the constraint can be generated, after the action is applied. In practice, ϵ is usually set to a small value like 0.01.

Examples. Several examples of constraints we discussed are listed in Table 2, where we show how to transform a constraint to a formal functional form.

4.2 Supporting Constraints

We present two approaches to support constraints.

Post Verification. A straightforward approach to support constraints is called *post verification.* For this approach, after applying AutoAction, a checker accepts the generated action and the group as input, then filters out actions which violate at least one constraint. While this two-step approach is easy to implement, it relies on the original optimization problem. If COBYLA fails to generate actions observing constraints, post verification will not work well.

Penalty-based Incorporation. For our approach, we adjust the original optimization problem to incorporate constraints, by modifying the objective function and adding penalty to it. It is a common practice to transform a constrained optimization to an unconstrained optimization.

Definition 12 (Incorporating constraints as penalty). *Given the objective function $f(g, s)$ and a set of n constraints $c_1, c_2, ..., c_n$,*

Table 2: Example Transformation of Constraints

Type	Constraint	Transformation
Action	$s.a >= s.b$	$F(c) = -s.a + s.b$
Instance	$(r^s.a)^2 + (r^s.b)^2 >= 5$	$f_m = -(r^s.a)^2 - (r^s.b)^2 + 5$
Instance	$r^s.b <= 10$ if $r^s.a >= 5$	$f_m = r^s.b - 10$, $f_{n-m} = -r^s.a + 5$

the objective functions will become

$$f'(g, s) = f(g, s) + \alpha \cdot \sum_{k=1}^{n} max(F_{c_k}, 0),$$

where α is an user defined parameter.

According to the definition of constraints in Section 4.1, when $F_{c_k} > 0$, the action is considered to violate the constraint c, then penalty is to be added to the objective value. In practice, the penalty factor α is usually set to a very large value like 1e8.

Ideally, the penalty component in the new objective value is equal to zero, which implies that each constraint is not violated for the given action. Compared with the post verification approach, the penalty-based approach adjusts the original optimization to make the algorithm more effective in supporting different types of constraints. We conduct experiments to show it in Section 5.3.

5 EXPERIMENTS

Our experiments aim to answer the following questions. (1) How do parameters affect AutoAction's performance? (Section 5.2); (2) How well can AutoAction support real-world constraints? (Section 5.3); (3) How well does AutoAction perform compared to baseline? (Section 5.4); (4) Are the actionable query explanations generated by AutoAction meaningful to real users? (Section 5.5)

5.1 Experimental Setup

In this section, we present our experimental setup.

Datasets. We evaluated AutoAction using two real world and two synthetic datasets. (1) *Adult* [15] is a real-world dataset, which indicates whether a person's income is above 50K/year from census income data. (2) *Student Academic Performance* [1, 2] is a real-world dataset, which indicates the academic performance of a group of students. For the synthetic datasets, we adopted prior work in [13] which assumes a credit card application scenario. The causal graph of the synthetic dataset is shown in the Figure 4. Besides the original setup, we added two group attributes g_1 and g_2, and assume that $X_1 := U_1, X_2 := 3/10 \cdot X_1 + U_2$, and $Y = sgn(X_1 + 5 \cdot X_2 - 22.5)$. For each value of attribute g_1 and g_2, we randomly generates two parameter $0 < \mu_1 < 1, -0.5 < \mu_2 < 0.5$, then assumes that $U_1 \sim Poission(\mu_1 \cdot 20), U_2 \sim 0.25 \cdot \mathcal{N}(\mu_2, 1)$. We generated two datasets randomly which contains 100K and 1M instances, and regarded them as (3) Credit-100K, which contains 1,000 groups and (4) Credit-1M, which contains 10K groups. For both synthetic datasets, we assigned the distribution of group size randomly.

Query, Attributes, Constraints. We consider an SUM query without a predicate since supporting predicates is trivial. The aggregation attributes are the grade for Student dataset, salary for Adult dataset, and the credit card application decision for two synthetic

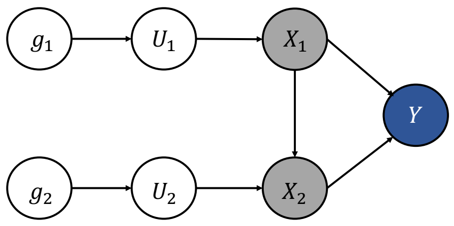
$$\begin{tikzcd} g_1 \arrow[r] & U_1 \arrow[r] & X_1 \arrow[d] \arrow[dr] & \\ & & & Y \\ g_2 \arrow[r] & U_2 \arrow[r] & X_2 \arrow[ur] & \end{tikzcd}$$

Figure 4: Causal graph of synthetic datasets used in experiment

datasets. We chose two group attributes and two actionable attributes for each dataset. For each dataset, we selected two action based constraints and one approximate instance-based constraint. The details of attribute selection is shown in Table 3 and the details of constraints used in this experiment is shown in Table 4.

Evaluation Metrics. We used the objective value of the generated action (g, s) as the metric for performance evaluation. When it is larger than zero, we recorded it as zero, implying that no feasible solution is successfully generated. Otherwise we recorded the absolute value of an objective value. For two real world datasets, no ground truth for the action effect can be obtained, so we used the value inferred by the model. For synthetic datasets, we used the ground truth.

Inference Model. For each dataset, we trained a random forest classifier whose maximum depth is 9 and the number of decision trees is 500, as the inference model M. We picked all numeric attributes in each dataset to fit the model.

Implementation and Environment. All methods were implemented in Python 3.7 and all experiments were conducted on a single-threaded 1.4 GHz Intel Core i5/8GB 2133 MHz LPDDR3. The operating system version was macOS 10.14.6. For the objective function, we set $\lambda = 0.99, \beta = 20, \alpha = 1e8$, and the threshold for approximate constraints $\epsilon = 0.01$. For optimization method COBYLA, we set its maximum iterations to be 200.

5.2 Parameter Analysis

AutoAction has Hyperband as a subroutine. We study how the budget parameter R for Hyperband affects AutoAction's performance. We set the time limit to 1 minute, which is a reasonable limit for real users to wait. We then run AutoAction 5 times for each value of R, and record their average objective values.

Figure 5 shows the result. We can see that when the budget R was set to 30, AutoAction failed to find a feasible solution under 1 minute time limit. This is because that the single round Hyperband would exceed the given time. When the budget was set to a small value like 1, the performance of AutoAction was not optimal as well. This is due to that the coverage of the subset of groups is small, so the optimization will terminate earlier, which decreases the quality of the generated action.

For the figure, we can see that $R = 10$ is a reasonable value balancing the running time and the depth of search process. We uses this parameter for all the remaining experiments.

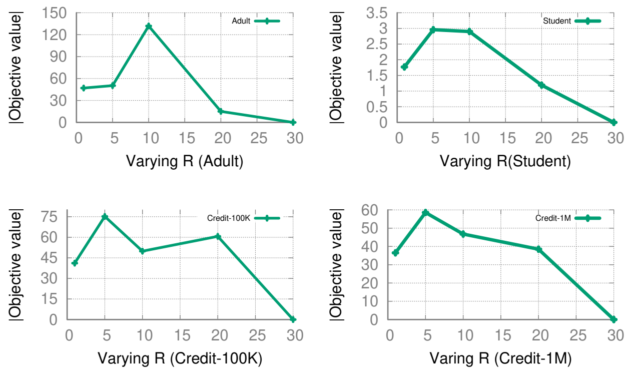

Figure 5: Analyzing the relationship between R and objective value for AutoAction (The higher, the better).

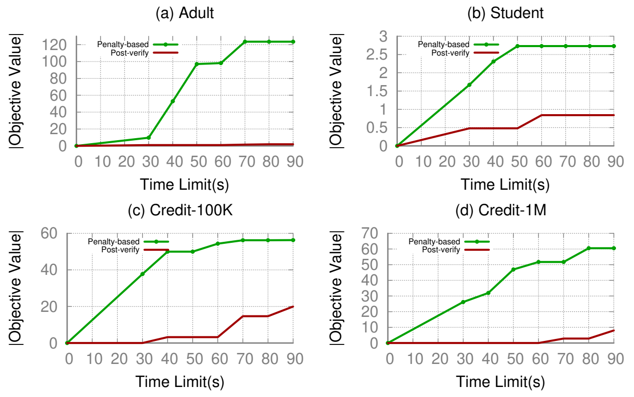

Figure 6: Comparing penalty-based and post verification approaches (The higher, the better).

5.3 Effectiveness of Supporting Constraints

We evaluate the effectiveness of supporting constraints of our method AutoAction, with penalty-based constraint supporting mechanism discussed in Section 4.2. For the baseline method, we applied the AutoAction with post verification. Figure 6 presents the experimental result where we set the time limit every 10 seconds from 30s to 90s. For each dataset we input three constraints. From the result, the penalty-based AutoAction outperformed post verification approach on all four datasets.

The quality of action generated by our method on four datasets kept increasing between the time limits are 30s and 1 minute. After 1 minute, its increase slowed down, which means that AutoAction can generate a relatively good action before it executes for around 1 minute. For the post verification approach, under 90s time limit, it also could generate some feasible solution for all four datasets. This is due to the coverage of the search algorithm with relatively small budget for each run, which guarantees the diversity of the action it produces. However, the quality of result is not comparable with that generated by penalty-based AutoAction.

Besides the default setting for constraints, we also conduct the experiment when there are one or two constraints, which are selected from three constraints in default setting. We ran both algorithms 5

Table 3: Selection of Attributes in Experiments

Dataset	Group Attributes	Actionable Attributes	Aggregation Attribute
Adult	work-class & occupation	working hour & education year	salary
Student	stage & section	# of raising hands & # of visiting courses	grade
Credit-100K & Credit-1M	g_1 & g_2	X_1 (salary) & X_2 (balance)	Y (application decision)

Table 4: Constraints used in Experiments

Dataset	Constraint
Adult	$s.year <= 2$
Adult	$s.year^2 + s.hour^2 >= 1$
Adult	$when\, s.year >= 12, s.hour <= 75$
Student	$s.raised^2 + s.visited^2 >= 5$
Student	$s.raised^2 + s.visited^2 <= 20$
Student	$when\, s.visited <= 10, s.raised >= 10$
Credit-100K & Credit-1M	$s.X_2 <= 1$
Credit-100K & Credit-1M	$s.X_2{}^2 + s.X_1{}^2 >= 0.5$
Credit-100K & Credit-1M	$when\, s.X_1 >= 15, s.X_2 >= 10$

times and took the average objective value for each time limit. The result of improvement on objective value is shown in Table 5.

When the number of constraints is increased to 3, the improvement of our method becomes much larger, except for the Student dataset. It is due to that when more constraints are input, it is harder for post verification approach to generate a feasible solution. For Student dataset, the setting of one constraint introduces a relatively strong condition, which makes the addition of other two constraints making less difference to the framework.

We also note that for both synthetic datasets, when the number of constraints is increased from 1 to 2, the improvement rate keeps similar and even decreases a little bit. The reason is that when the number of constraints increases, the quality of the actions generated by penalty-based AutoAction will also decrease. So the improvement rate may not necessarily increase under such cases.

5.4 Comparing with Baselines

We compare the effectiveness of our method with two baselines under each given time limit. Two baselines we implemented are introduced in Section 3.2, which are random search and single round Hyperband. For single round Hyperband, as its running time is not stable, according to Figure 3, we could not directly assign parameters based on the time limit. So, we picked candidate values of input budget R of Hyperband, then run Hyperband 10 times, take both the average of both running time objective score. The result of average running time when varying R is shown in Table 6. For each time limit T, we chose the input budget R to maximize $\bar{f}_R$, which was the average objective value of 10 runs of single round Hyperband, and the average running time $\bar{t}_R$ satisfies $\bar{t}_R <= T$. Then, we record $\bar{f}_R$ as the objective value under T. For random search and AutoAction, same as Section 5.3, we ran them 5 times and report the average objective value.

Table 5: The improvement of penalty-based AutoAction over post verification approach, on four datasets.

	# of Constraints		
Datasets	1	2	3
Adult	×6.94	×44.68	×101.80
Student	×3.34	×2.24	×3.24
Credit-100K	×3.62	×2.84	×16.96
Credit-1M	×2.65	×2.13	× Infinity

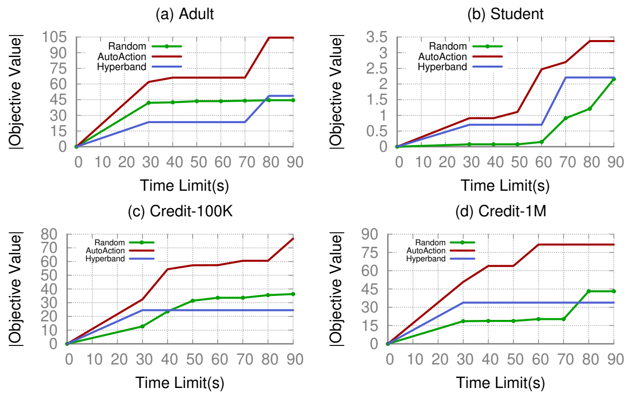

Figure 7: The comparison of AutoAction, random search and Hyperband, on four datasets (The higher, the better).

The result is shown in Figure 7. AutoAction generated better actions compared with two other baselines under every time limit for all four datasets. For two synthetic datasets, when the time limit was relatively small, the single round Hyperband outperformed random search, however when time limit became larger, the random search would behave better. It shows that single round Hyperband is not flexible for different time budget compared with random search. Along with the increase of budget R, the running time of single round Hyperband will increase a lot due to the postpone of early stopping, while the random search could keep discovering

Table 6: Average running time (s) of 10 executions of single round Hyperband, varying R

	R			
Datasets	20	25	30	35
Adult	19.74	74.24	>90s	>90s
Student	21.29	64.04	80.17	>90s
Credit-100K	29.75	>90s	>90s	>90s
Credit-1M	30.54	>90s	>90s	>90s

other groups in relatively short time. For the Adult dataset, the performance of single round Hyperband is worse than random search before 70s time limit. This is due to that there is very high imbalance among subgroups of this dataset, so the single round Hyperband with small budget in this experiment may wrongly discard groups with high potential, decreasing its performance.

5.5 User Study

Finally, as there is no authoritative ground truth on the "actionability" of actionable query explanation, we conducted a user study to validate the real-world utility of our system.

We focus on two tasks: (1) Do users prefer explanations generated by our system or baseline models, and (2) Does our approach adequately generalize to a variety of real-world, practical scenarios?

Methodology. We recruited 20 graduate students in computer science and information science as well as other graduate students who had taken four high level technical courses on related topics. 12 participants identified as male, 8 as female, and 0 as non-binary. All user studies were conducted online using video and screen share on Zoom as a result of pandemic restrictions. After consenting to the study, participants received a 5-minutes tutorial introducing basic concepts of counterfactual query, action, cost, and fairness preservation. This also offered the opportunity to inform participants of tool-specific terminologies such as action, the cost of the action, and effectiveness.

In the main task, participants analyzed two datasets in two 20 minute periods (datasets were counterbalanced for order effects). Participants had access to all raw data as well as pivot table visualizations and a Python notebook. Each participant was asked to choose which of four candidate actions would best achieve a hypothetical goal (e.g., "Which action would best improve the scores of primary school students?"). Goals were developed for two dataset scenarios as mentioned in the experimental setup session, Adult and a student performance dataset. Each dataset contains outcome measures such as salary and test scores. Participants' goal was to improve these outcomes. We used both our system and baselines to generate possible actions to provide to participants: 2 were generated from our constraint-based algorithms (the top-2 based on its rankings), and 2 candidates were generated using a baseline objective function that didn't consider fairness constraints and trade-off between cost and effectiveness. We made sure that both baseline results in the survey could achieve high objective function scores. In this way, we can have a better clear picture of whether AutoAction generates actions which better map to participants'

Examining AutoAction's effectiveness. As there is no ground truth best action for each task, we employed a majority vote metric. For each task we counted the percentage of participants who chose options provided by AutoAction versus those provided by the baselines. We averaged scores across all questions from both datasets to create a general preference measure. We found that the average score of AutoAction (mean=0.817) is 4.3 times($\rho < .001$) higher than the baseline model (mean=0.19), demonstrating that participants generally favored actions provided by AutoAction over those from the baseline. Interestingly, participants' preference was not identical across datasets. In the Student dataset, which is simpler (fewer columns [adult:15 columns, student:7 columns] and rows [adult: 32561 rows, student: 480 rows] compared to Adult), participants' preferred AutoAction suggestions at a rate 7.3 times higher than that of the baseline. On the other hand, in the more complex Adult dataset, participants preferred AutoAction 3 times over the baseline. We believe that this may be due to participants' difficulty in judging smaller differences between actions for a more complex case, though more study is necessary.

Qualitative Response. To provide additional insight into participants' responses to the different tasks and actions they saw during the study, we also asked them to provide a written justification for the specific action they picked. Later, we also asked them for general feedback.

17 participants specifically mentioned that they were balancing trade-offs between effectiveness and costs. For example, P5 mentioned that "The effectiveness is good, but the cost is higher because of 705 records". Participants also expressed concern over preserving the statistical properties of data as they took actions (much as in the fairness constraints). P5 specifically mentioned a need to preserve statistical disparities as they took actions. This is an encouraging sign that AutoAction suggestions encouraged participants to think holistically about which actions would be best for their population.

Domain expertise and life experience also played a role in their decision-making. 16 participants referenced real world issues in their responses. P12 mentioned that since some middle schools have limited access to resources, "it would be better to encourage them." P11 continued this line of thought, expressing, "But there will be some fairness issue related since section A and B [which differ by race] should be treated fairly". Some considered real-world scenarios such as "...the one with local-gov seems more practical than self-employing" (P9). Others thought about future implications ("encouraging more investment in education is beneficial for the long term" [P9]). Participants also desired more accountability, context, and justification in the actions they were provided. For example, P13 said, "My intuition is that without fully controlling for confounding factors, it is not possible to make decisions based on a point estimate of an effective score." and P9 said "Without knowing the cost metric..., it's hard to evaluate the effect-cost tradeoff". This suggests that in the future AutoAction ought to provide more avenues for participants to leverage their expertise and more insight into the specific optimization that led to its suggested action.

6 RELATED WORK

We review the related work in this section.

Query Explanation. The output of an aggregation query can be explained by data insights generated by query explanation frameworks. A common type of query explanation returns a predicate that represents a group of records. Under this setting, the aggregation result of the group of records contributes much to the overall query result, thus serving as an explanation. It implies that the removal of those records will change the output of the original query to an opposite direction [26, 27, 35]. Another type of explanation framework is implemented by generating individual records that help people better understand the data pattern [20].

While our framework also gives explanations to SQL queries, unlike prior work, our framework generates actionable explanations rather than provenance-based explanations.

Machine Learning Explanation. Machine learning explanation helps people to understand black-box ML models by generating data insights. Particularly, the counterfactual explanation problem has the most closed setting to our problem, where *actionable recourse* over instances are generated as explanations [13, 22, 29, 32]. To improve the feasibility of generated recourse under real-world settings, the explanation frameworks are made to support causal knowledge between variables [17] or work under imperfect causal knowledge [14]. Framework also have been made to introduce other type of counterfactual explanations in recourse settings [8] or construct an interactive system to generate such explanations [28].

For our problem, we use a given inference model to help to identify the effect of actionable explanations. Unlike prior work, our explanations are at a group level w.r.t. an unexpected query result rather than at an instance level w.r.t. a model misprediction.

7 CONCLUSION

This paper studied how to enable automated generation of actionable query explanations. To the best of our knowledge, this is the first study on this important topic. We defined an objective function by considering three aspects of an action (cost, effect, and fairness) and formalized the actionable query explanation problem as an optimization problem. We discussed how to solve this problem in a single-group setting using COBYLA. For multi-group settings, we modeled it as a multi-armed bandit problem. Hyperband failed to solve the problem effectively because its running time under the same budget was highly unstable. To overcome this limitation, we proposed AutoAction, whose main idea is to treat Hyperband as a group-selection component and run it for multiple times through the whole search process. We defined three types of constrains inspired by real-world scenarios and proposed two methods to extend AutoAction to support them. The experimental and user study results show that i) AutoAction significantly outperformed random search and Hyperband in terms of efficiency, and ii) users preferred actionable explanations generated by AutoAction compared to baseline models.

REFERENCES

[1] Elaf Abu Amrieh, Thair Hamtini, and Ibrahim Aljarah. 2015. Preprocessing and analyzing educational data set using X-API for improving student's performance. In *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*. IEEE, 1–5.

[2] Elaf Abu Amrieh, Thair Hamtini, and Ibrahim Aljarah. 2016. Mining educational data to predict student's academic performance using ensemble methods. *International Journal of Database Theory and Application* 9, 8 (2016), 119–136.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2 (2002), 235–256.

[4] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing* 32, 1 (2002), 48–77.

[5] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. 2009. *Introduction to derivative-free optimization.* SIAM.

[6] Diego Desani, Veronica Gil-Costa, Cesar AC Marcondes, and Hermes Senger. 2016. Black-box optimization of Hadoop parameters using derivative-free optimization. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. IEEE, 43–50.

[7] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. 2019. Quickinsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the 2019 International Conference on Management of Data*. 317–332.

[8] Sainyam Galhotra, Romila Pradhan, and Babak Salimi. 2021. Explaining black-box algorithms using probabilistic contrastive counterfactuals. In *Proceedings of the 2021 International Conference on Management of Data*. 577–590.

[9] Daeyoung Hong, Woohwan Jung, and Kyuseok Shim. 2021. Collecting Geospatial Data with Local Differential Privacy for Personalized Services. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2237–2242.

[10] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated machine learning: methods, systems, challenges.* Springer Nature.

[11] Kevin Jamieson and Ameet Talwalkar. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*. 240–248.

[12] Amir-Hossein Karimi, Gilles Barthe, Bernhard Schölkopf, and Isabel Valera. 2020. A survey of algorithmic recourse: definitions, formulations, solutions, and prospects. *arXiv preprint arXiv:2010.04050* (2020).

[13] Amir-Hossein Karimi, Bernhard Schölkopf, and Isabel Valera. 2021. Algorithmic recourse: from counterfactual explanations to interventions. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. 353–362.

[14] Amir-Hossein Karimi, Julius Von Kügelgen, Bernhard Schölkopf, and Isabel Valera. 2020. Algorithmic recourse under imperfect causal knowledge: a probabilistic approach. *arXiv preprint arXiv:2006.06831* (2020).

[15] Ron Kohavi. 1996. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid.. In *Kdd*, Vol. 96. 202–207.

[16] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research* 18, 1 (2017), 6765–6816.

[17] Divyat Mahajan, Chenhao Tan, and Amit Sharma. 2019. Preserving causal constraints in counterfactual explanations for machine learning classifiers. *arXiv preprint arXiv:1912.03277* (2019).

[18] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2019. A survey on bias and fairness in machine learning. *arXiv preprint arXiv:1908.09635* (2019).

[19] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. 2014. Causality and explanations in databases. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1715–1716.

[20] Zhengjie Miao, Qitian Zeng, Chenjie Li, Boris Glavic, Oliver Kennedy, and Sudeepa Roy. 2019. CAPE: explaining outliers by counterbalancing. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1806–1809.

[21] Tova Milo and Amit Somech. 2020. Automating exploratory data analysis via machine learning: An overview. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2617–2622.

[22] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. 607–617.

[23] Michael JD Powell. 1994. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*. Springer, 51–67.

[24] Michael JD Powell. 1998. Direct search algorithms for optimization calculations. *Acta numerica* (1998), 287–336.

[25] Luis Miguel Rios and Nikolaos V Sahinidis. 2013. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization* 56, 3 (2013), 1247–1293.

[26] Sudeepa Roy, Laurel Orr, and Dan Suciu. 2015. Explaining query answers with explanation-ready databases. *Proceedings of the VLDB Endowment* 9, 4 (2015), 348–359.

[27] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1579–1590.

[28] Maximilian Schleich, Zixuan Geng, Yihong Zhang, and Dan Suciu. 2021. GeCo: Quality counterfactual explanations in real time. *arXiv preprint arXiv:2101.01292* (2021).

[29] Berk Ustun, Alexander Spangher, and Yang Liu. 2019. Actionable recourse in linear classification. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*. 10–19.

[30] Konstantinos Varelas and Marie-Ange Dahito. 2019. Benchmarking multivariate solvers of SciPy on the noiseless testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1946–1954.

[31] Hoang Dung Vu, Kok Soon Chai, Bryan Keating, Nurislam Tursynbek, Boyan Xu, Kaige Yang, Xiaoyan Yang, and Zhenjie Zhang. 2017. Data driven chiller plant energy optimization with domain knowledge. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1309–1317.

[32] Sandra Wachter, Brent Mittelstadt, and Chris Russell. 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.* 31 (2017), 841.

[33] Dakuo Wang, Justin D Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. 2019. Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–24.

[34] Pei Wang, Weiling Zheng, Jiannan Wang, and Jian Pei. 2021. Automating Entity Matching Model Development. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1296–1307.

[35] Eugene Wu and Samuel Madden. 2013. Scorpion: explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment* 6, 8 (2013), 553–564.