DataPrism: Exposing Disconnect between Data and Systems

Sainyam Galhotra
University of Chicago
sainyam@uchicago.edu

Anna Fariha
Microsoft
annafariha@microsoft.com

Raoni Lourenço
New York University
raoni@nyu.edu

Juliana Freire
New York University
juliana.freire@nyu.edu

Alexandra Meliou
University of Massachusetts Amherst
ameli@cs.umass.edu

Divesh Srivastava
AT&T Chief Data Office
divesh@research.att.com

ABSTRACT

As data is a central component of many modern systems, the cause of a system malfunction may reside in the data, and, specifically, particular properties of data. E.g., a health-monitoring system that is designed under the assumption that weight is reported in lbs will malfunction when encountering weight reported in kilograms. Like software debugging, which aims to find bugs in the source code or runtime conditions, our goal is to debug data to identify potential sources of *disconnect* between the assumptions about some data and systems that operate on that data. We propose DataPrism, a framework to identify data properties (*profiles*) that are the root causes of performance degradation or failure of a data-driven system. Such identification is necessary to repair data and resolve the disconnect between data and systems. Our technique is based on *causal reasoning* through *interventions*: when a system malfunctions for a dataset, DataPrism alters the data profiles and observes changes in the system's behavior due to the alteration. Unlike statistical observational analysis that reports mere correlations, DataPrism reports causally verified root causes—in terms of data profiles—of the system malfunction. We empirically evaluate DataPrism on seven real-world and several synthetic data-driven systems that fail on certain datasets due to a diverse set of reasons. In all cases, DataPrism identifies the root causes precisely while requiring orders of magnitude fewer interventions than prior techniques.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Information systems** → *Data cleaning*; • **Computing methodologies** → **Causal reasoning and diagnostics**.

KEYWORDS

Debugging, root-cause identification, data profiles, causal testing

ACM Reference Format:
Sainyam Galhotra, Anna Fariha, Raoni Lourenço, Juliana Freire, Alexandra Meliou, and Divesh Srivastava. 2022. DataPrism: Exposing Disconnect between Data and Systems. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3514221.3517864

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517864

1 INTRODUCTION

Traditional software debugging aims to identify errors and bugs in the mechanism—such as source code, configuration files, and runtime conditions—that may cause a system to malfunction [26, 36, 50]. However, in modern systems, data has become a central component that itself can cause a system to fail. Data-driven systems comprise complex pipelines that rely on data to solve a target task. Prior work addressed the problem of debugging machine learning models [14] and finding root causes of failures in computational pipelines [52], where certain values of the pipeline parameters—such as a specific model and/or a specific dataset—cause the pipeline failure. However, just knowing that a pipeline fails for a certain dataset is not enough; naturally, we ask: what *properties* of a dataset caused the failure?

Two common reasons for malfunctions in data-driven systems are: (1) incorrect data, and (2) *disconnect* between the assumptions about the data and the design of the system that operates on the data. Such disconnects may happen when the system is not robust, i.e., it makes strict assumptions about metadata (e.g., data format, domains, and distributions), and when new data drifts away from the data over which the system was tested on before deployment [60] (e.g., when a system expects a data stream to have a weekly frequency, but the data provider suddenly switches to daily data).

Therefore, in light of a failure, one should investigate potential issues in the data. Some specific examples of commonly observed system malfunctions caused by data include: (1) decline of a machine learning model's accuracy (due to out-of-distribution data), (2) unfairness in model predictions (due to imbalanced training data), (3) excessive processing time (due to a system's failure to scale to large data), and (4) system crash (due to invalid input combination in the data tuples beyond what the system was designed to handle). These examples indicate a common problem: *disconnect* or *mismatch* between the data and the system design. Once the mismatch is identified, possible fixes can be either to repair or reformat the data to comply with the system design, or to adjust the system design (i.e. modify source code) to accommodate data with different properties.

A naïve approach to deal with potential issues in the data is to identify outliers: report tuples as potentially problematic based on how atypical they are with respect to the rest of the tuples in the dataset. However, without verifying whether the outliers actually cause unexpected outcomes, we can never be certain about the actual root causes. As pointed out in prior work [9]: *"With respect to a computation, whether an error is an outlier in the program's input distribution is not necessarily relevant. Rather, potential errors can be* *spotted by their effect on a program's output distribution."* To motivate our work, we start with an example inspired from a real-world incident, where Amazon's delivery service was found to be racist [44].

EXAMPLE 1 (BIASED CLASSIFIER). *An e-commerce company wants to build an automated system to offer customer discounts. To this end, they organize data of customers' purchases over a year into a dataset with attributes* `name`*,* `gender`*,* `age`*,* `race`*,* `zip_code`*,* `phone`*,* `products_purchased`*, etc. Anita, a data scientist, is asked to develop a machine learning (ML) pipeline over this dataset to predict whether a customer will spend over a certain amount, and, subsequently, should be offered discounts. Anita decides to use an off-the-shelf ML algorithm that is trained over historical data. To avoid discrimination over any group and to ensure that the classifier trained on this dataset is fair, Anita decides to drop the sensitive attributes—*`race` *and* `gender`*—during the pre-processing step of the ML pipeline, before feeding it to the classifier. However, despite this effort, the trained classifier turns out to be highly biased against African Americans and women. After a close investigation, Anita discovers that: (1) in the training data,* `race` *is highly correlated with* `zip_code`*, and (2) the training dataset is imbalanced: a larger fraction of customers who purchase expensive products are male. Now she wonders: if these two properties did not hold in the dataset, would the learned classifier be fair? Have either (or both) of these properties caused the observed unfairness?*

Existing tools [9] that blame individual cells (values) cannot help here, as no single cell in the training data is responsible for the observed discrimination, rather, global statistical properties (e.g., correlation) that involve multiple attributes over the entire data are the actual culprits. Also, Anita only identified two potential, merely *correlated* data issues that may not be the actual cause of the unfairness. To distinguish mere correlation from true causation, we need to dig deeper. Example 1 is one among many incidents in real-world applications where issues in the data caused systems to malfunction [11, 34]. A recent study of 112 high-severity incidents in Microsoft Azure services showed that 21% of the bugs were due to inconsistent assumptions about data format by different software components or versions [51]. The study further found that 83% of the data-format bugs were due to inconsistencies between data producers and data consumers, while 17% were due to mismatch between interpretations of the same data by different data consumers. Similar incidents happened due to misspelling and incorrect datetime format [63], and issues pertaining to data fusion where schema assumptions break for a new data source [21, 75]. We provide another example where a system times out when the distribution of the data, over which the system operates, exhibits significant skew.

EXAMPLE 2 (PROCESS TIMEOUT). *A toll collection software EZGO checks if vehicles passing through a gate have electronic toll passes installed. If it fails to detect a toll pass, it uses external OCR software to extract the registration number from images of the vehicle license plates. At midnight, EZGO processes all images in a batch mode and is expected to finish processing within 4 hours and generate a report. However, on certain days, EZGO fails to produce the report after 4 hours. A close investigation reveals that the external OCR software uses an algorithm that is extremely slow for images of black license plates captured in low illumination. As a result, when a batch contains a large number of such cases (significantly skewed distribution), EZGO fails.*

The aforementioned examples bring forth two key challenges. First, we need to correctly identify potential causes of unexpected outcomes and generate *hypotheses* that are expressive enough to capture the candidate root causes. For example, "outliers cause unexpected outcomes" is just one of the many possible hypotheses, which offers very limited expressivity. Second, we need to *verify* the hypotheses to confirm or refute them, which enables us to pinpoint the actual root causes, eliminating false positives.

Data profile as root cause. Towards solving the first challenge, our observation is that data-driven systems often function properly for certain datasets, but malfunction for others. Such malfunction is often rooted in certain *properties* of the data, called *data profiles* [1], that distinguish passing and failing datasets. Examples include size of a dataset, domains and ranges of attribute values, correlations between attribute pairs, conditional independence [77], functional dependencies and their variants [16, 25, 41, 46, 56], and other more complex data profiles [20, 27, 49, 57, 72].

Oracle-guided root cause identification. Our second observation is that an *oracle* that indicates whether the system functions desirably or not, can help verify our hypotheses, and allows us to precisely isolate the correct root causes of the undesirable malfunction from a set of candidate causes. Here, an oracle is a mechanism that can characterize whether the system functions properly over the input data. The definition of proper functioning is application-specific; for example, processing the input dataset under a certain time constraint may indicate proper functioning for streaming applications and achieving a certain accuracy may indicate proper functioning for an ML pipeline. Such oracles are often available in many practical settings, and have been considered in prior work [26, 52].

Solution sketch. We propose DATAPRISM, a framework that identifies and exposes data profiles that cause an opaque data-driven system (i.e., a system whose internal mechanisms are unknown) to malfunction. Our framework involves two main components: (1) an intervention-based mechanism that alters the profiles of a dataset, and (2) a mechanism that speeds up analysis by carefully selecting appropriate interventions. Given a scenario where an opaque system malfunctions (fails) over a dataset but functions properly (passes) over another, DATAPRISM focuses on the *discriminative* profiles, i.e., data profiles that significantly differ between the two datasets. DATAPRISM's intervention mechanism modifies the "failing" dataset to alter one of the discriminative profiles; it then observes whether this intervention causes the system to perform desirably, or the malfunction persists. DATAPRISM speeds up this analysis by favoring interventions on profiles that are more likely causes of the malfunction. To estimate this likelihood, we leverage three properties of a profile: (1) *coverage*: the more tuples an intervention affects, the more likely it is to change the system behavior, (2) *discriminating power*: the bigger the difference between the failing and the passing datasets over a profile, the more likely that the profile is a cause of the malfunction, and (3) *attribute association*: if a profile involves an attribute that is also involved with a large number of other discriminative profiles, then that profile has high likelihood to be a root cause. This is because altering such a profile is likely to passively repair other discriminative profiles as a side-effect (through the associated attribute). While an intervention may involve a large number of tuples, it is conceptually succinct (e.g., gender=male).

Scope. In this work, we only focus on the cases where system malfunction is due to some holistic profile(s) over the input dataset. Note that prior data-debugging approaches target different classes of data issues [8, 63] and assume access to the internal mechanisms of the system. In contrast, DataPrism is completely agnostic to the type of data-driven system and can support any system ranging from machine learning prediction models (e.g., binary classifiers, regression models, deep neural networks, etc.) that learn from data (Example 1) to other general software that just operate on data (Example 2). In Section 5, we demonstrate the efficacy of DataPrism over a diverse set of opaque systems that internally deploy different ML models such as logistic regression, neural network, AdaBoost, and random forest classifier. However, DataPrism cannot be used to prevent malfunctions from happening in the first place as it requires the knowledge of contrasting scenarios where the system under consideration malfunctions (fails) over a dataset versus functions properly (passes) over another dataset.

DataPrism requires knowledge of the classes of (domain-specific) data profiles that encompass the potential root causes. E.g., in Example 1, we assume the knowledge that correlation between attribute pairs and disparity between the conditional probability distributions (the probability of belonging to a certain gender, given price of items bought) are potential causes of malfunction. This assumption is realistic because: (1) For a number of tasks there exists a well-known set of relevant profiles: e.g., class imbalance and correlation between sensitive and non-sensitive attributes are common causes of unfairness in classification [10]; and violation of conformance constraints [27], missing values, and out-of-distribution tuples are well-known causes of ML model's performance degradation. (2) Domain experts are typically aware of the likely class of data profiles for the specific task at hand and can easily provide this additional knowledge as a *conservative* approximation, i.e., they can include extra profiles just to err on the side of caution. Notably, this assumption is also extremely common in software debugging techniques [26, 50, 79], which rely on the assumption that the "predicates" (traps to extract certain runtime conditions) are expressive enough to encode the root causes, and software testing [53], validation [48], and verification [38] approaches, which rely on the assumption that the test cases, specifications, and invariants reasonably cover the codebase and correctness constraints.

Data profiles. While we use data profiles to explain the cause of malfunction, developing data profiling techniques [1] is orthogonal to our work. A number of data profiling primitives exist in the literature along with corresponding techniques to extract them from a dataset. DataPrism assumes access to a suite of data-profiling techniques and uses them to extract profiles from the data. DataPrism then examines these profiles to identify the causes of system malfunction. To support a new data profile, DataPrism needs the corresponding mechanisms for their discovery and intervention. We discuss some common classes of data profiles as representative ones, which are currently supported in the implementation of DataPrism, and the corresponding discovery and intervention techniques. For data-profile discovery, we rely on prior work on pattern discovery [58], statistical-constraint discovery [77], data-distribution learning [37], knowledge-graph-based concept identification [31], conformance-constraint discovery [27], etc. While our evaluation covers specific data profiles (for which efficient discovery techniques exist), DataPrism is generic and works for any class of data profiles, as long as the corresponding discovery and intervention techniques are available.

Limitations of prior work. To find potential issues in data, Dagger [62, 63] provides data debugging primitives for human-in-the-loop interactions with data-driven computational pipelines. Other explanation-centric efforts [7, 19, 24, 75] report salient properties of historical data based only on observations. In contrast with observational techniques, the presence of an oracle allows for interventional techniques [52] that can query the oracle with additional, system-generated test cases to identify root causes of system malfunction more accurately. One such approach is CheckCell [9], which presents a ranked list of cells of data rows that unusually affect output of a given target function. CheckCell uses a fine-grained approach: it removes one cell of the data at a time, and observes changes in the output distribution. While it is suitable for small datasets, where it is reasonable to expect a human-in-the-loop paradigm to fix cells one by one, it is not suitable for large datasets, where no individual cell is significantly responsible, rather, a holistic property of the entire dataset (profile) causes the problem. Capuchin [69] is an interventional approach that, similar to our approach, alters the dataset to remove attribute correlations, a prime cause for prediction bias in ML algorithms. However, Capuchin does not verify if there is indeed a causal connection between attribute correlation and prediction bias. In contrast, DataPrism offers a general solution to verify and isolate the true root cause of system malfunction for a diverse set of systems.

Interpretable machine learning is related to our problem, where the goal is to explain behavior of machine learning models. However, prior work on interpretable machine learning [65, 66] typically provide *local* (tuple-level) explanations, as opposed to *global* (dataset-level) explanations. While some approaches provide feature importance as a global explanation for model behavior [17], they do not model feature interactions as possible explanations.

Contributions. In this paper, we make the following contributions:

- We formalize the novel problem of identifying root causes (and fixes) of the disconnect between data and data-driven systems in terms of data profiles (and interventions). (Section 2)
- We design a set of data profiles that are common root causes of system malfunctions, and discuss their discovery and intervention techniques based on available technology. (Section 3)
- We design and develop a novel interventional approach to pinpoint causally verified root causes. The approach leverages a few properties of the data profiles to efficiently explore the space of candidate root causes with a small number of interventions.
- We evaluate DataPrism on seven real-world applications, where data profiles are responsible for causing system malfunction, and demonstrate that DataPrism successfully explains the root causes with very few interventions (fewer than 10). Furthermore, DataPrism requires 10–1000× fewer interventions compared to two state-of-the-art techniques for root-cause analysis: BugDoc [52] and Anchors [66]. Over synthetic pipelines, we further show that the number of required interventions by DataPrism increases sub-linearly with the number of discriminative profiles.

2 PRELIMINARIES & PROBLEM DEFINITION

In this section, we first formalize the notions of system malfunction and data profile, its violation, and transformation function used for intervention. We then proceed to define explanation (cause and corresponding fix) of system malfunction and formulate the problem of data-profile-centric explanation of system malfunction.

Basic notations. We use $\mathcal{R}(A_1, A_2, \ldots, A_m)$ to denote a relation schema over m attributes, where A_i denotes the i^{th} attribute. We use $\texttt{Dom}_i$ to denote the domain of attribute A_i. Then the set $\mathbf{Dom}^m = \texttt{Dom}_1 \times \cdots \times \texttt{Dom}_m$ specifies the domain of all possible tuples. A dataset $D \subseteq \mathbf{Dom}^m$ is a specific instance of the schema $\mathcal{R}$. We use $t \in \mathbf{Dom}^m$ to denote a tuple in the schema $\mathcal{R}$. We use $t.A_i \in \texttt{Dom}_i$ to denote the value of the attribute A_i of the tuple t and use $D.A_j$ to denote the multiset of values all tuples in D take for attribute A_j. We use $\vec{A}$ to denote an ordered list of all attributes.

2.1 Quantifying System Malfunction

To measure how much the system malfunctions over a dataset, we use the *malfunction score.*

Definition 3 (Malfunction score). *Let $D \subseteq \mathbf{Dom}^m$ be a dataset, and S be a system operating on D. The malfunction score $m_S(D) \in [0, 1]$ is a real value that quantifies how much S* malfunctions *when operating on D.*

The malfunction score $m_S(D) = 0$ indicates that S functions properly over D and a higher value indicates a higher degree of malfunction, with 1 indicating extreme malfunction. A threshold τ defines the acceptable degree of malfunction and translates the continuous notion of malfunction to a Boolean value. If $m_S(D) \leq \tau$, then D is considered to pass w.r.t S; otherwise, a mismatch between D and S exists, whose cause (and fix) we aim to expose.

Example 4. *For a binary classifier, its misclassification rate (additive inverse of accuracy) over a dataset can be used as a malfunction score. Given a dataset D, if a classifier S makes correct predictions for tuples in $D' \subseteq D$, and incorrect predictions for the remaining tuples, then S achieves accuracy $\frac{|D'|}{|D|}$, and, thus, $m_S(D) = 1 - \frac{|D'|}{|D|}$.*

Example 5. *In fair classification, we can use disparate impact [40], the ratio between the fraction of tuples with favorable outcomes within the unprivileged and the privileged groups, to measure malfunction.*

2.2 Profile-Violation-Transformation (PVT)

Once we detect a mismatch, the next step is to investigate its cause. We use *data profiles* to model the possible causes of mismatch. The schema of a data profile is given as a template that can be parameterized with different values. Populating a profile template with a particular set of values produces an instantiation of the profile (P). Given a dataset D, we use existing data-profiling techniques to discover parameter values, such that D satisfies the corresponding profile instances. To measure how much a dataset D satisfies or violates a data profile, we need a violation function (V) that gives *semantics* to the data profiles. Finally, to repair a dataset D, with respect to a data profile and its corresponding violation function, we need a transformation function (T). Transformation functions provide an intervention mechanism to alter data and suggest repairs to remove the cause of malfunction. DataPrism requires the following three components over the schema ⟨**P**rofile, **V**iolation function, **T**ransformation function⟩, PVT in short:

(1) P: an instantiated profile, which follows the schema ⟨profile type, parameters⟩.
(2) $V(D, P)$: a violation function that computes how much the dataset D violates the profile P and returns a violation score.
(3) $T(D, P, V)$: a transformation function that transforms the dataset D to another dataset D' such that D' no longer violates the profile P with respect to the violation function V. (When clear from the context, we omit the parameters P and V in the notation of transformation functions.)

For a PVT triplet X, we use X_P, X_V, and X_T to denote its profile, violation function, and transformation function, respectively. As an example, consider a profile P in a PVT $\langle P, V, T\rangle$ over a dataset D: P may correspond to the ideal domain of an attribute A in D, specified by $\texttt{Dom}(A)$, $V(D, P)$ may correspond to the fraction of out-of-domain values in $D.A$, and $T(D, P, V)$ can be to remove all tuples $t \in D$ that contain out-of-domain values (i.e., $t.A \notin Dom(A)$). We provide detailed examples and additional discussion on data profiles, violation functions, and transformation functions in Section 3. We proceed to formalize these notions.

2.2.1 Data Profile. Intuitively, data profiles encode dataset characteristics. They can refer to a single attribute (e.g., mean of an attribute) or multiple attributes (e.g., correlation between a pair of attributes, functional dependencies, etc.).

Definition 6 (Data Profile). *Given a dataset D, a data profile P denotes properties or constraints that tuples in D (collectively) satisfy.*

2.2.2 Profile Violation Function. To quantify the degree of violation a dataset incurs with respect to a data profile, we use a *profile violation function* that returns a numerical violation score.

Definition 7 (Profile violation function). *Given a dataset D and a data profile P, a* profile violation function $V(D, P) \mapsto [0, 1]$ *returns a real value that quantifies how much D violates P.*

$V(D, P) = 0$ implies that D fully complies with P (does not violate it at all). In contrast, $V(D, P) > 0$ implies that D violates P. The higher the value of $V(D, P)$, the higher the profile violation.

2.2.3 Transformation Function. In this work, we assume knowledge of a passing dataset for which the system functions properly, and a failing dataset for which the system malfunctions. Our goal is to identify which profiles of the failing dataset caused the malfunction. We seek answer to the question: how to "fix" the issues within the failing dataset such that the system no longer malfunctions on it (mismatch is resolved)? To this end, we apply *interventional causal reasoning*: we intervene on the failing dataset by altering its attributes such that the profile of the altered dataset matches the corresponding correct profile of the passing dataset. To perform intervention, we need *transformation functions* with the property that it should push the failing dataset "closer" to the passing dataset in terms of the profile that we are interested to alter. More formally, after the transformation, the profile violation score should decrease.

Definition 8 (Transformation function). *Given a dataset D, a data profile P, a selection predicate $\mathbb{S}$ and a violation function V, a transformation function $T(D, P, V, \mathbb{S}) \mapsto 2^{\mathbf{Dom}^m}$ alters tuples in $\sigma_{\mathbb{S}}(D)$ to produce D' such that $V(D', P) < V(D, P)$.*

The selection predicate $\mathbb{S}$ characterizes the subset of tuples in D that can be altered during transformation, and helps limit transformation to within a certain subset of the data. An empty $\mathbb{S}$ indicates that any tuple can be transformed. A dataset can be transformed by applying a series of transformation functions, for which we use the composition operator ($\circ$).

Definition 9 (Composition of transformations). *Given a dataset D, and two PVT triplets X and Y, $(X_T \circ Y_T)(D) = X_T(Y_T(D))$. Further, if $D'' = (X_T \circ Y_T)(D)$, then $X_V(D'', X_P) = Y_V(D'', Y_P) = 0$.*

2.3 Problem Definition

We expose a set of PVT triplets for explaining the system malfunction. The explanation contains both the *cause* and the corresponding *fix*: profile within a PVT triplet indicates the cause of system malfunction with respect to the corresponding transformation function, which suggests the fix.

Definition 10 (Explanation of system malfunction). *Given*
(1) a system S with a mechanism to compute $m_S(D)$ $\forall D \subseteq \mathbf{Dom}^m$,
(2) an allowable malfunction threshold τ,
(3) a passing dataset D_{pass} for which $m_S(D_{pass}) \leq \tau$,
(4) a failing dataset D_{fail} for which $m_S(D_{fail}) > \tau$, and
(5) a set of candidate PVT triplets $\mathcal{X}$ such that $\forall X \in \mathcal{X}$ $X_V(D_{pass}, X_P) = 0 \wedge X_V(D_{fail}, X_P) > 0$,
the explanation of the malfunction of S for D_{fail}, but not for D_{pass}, is a set of PVT triplets $\mathcal{X}^* \subseteq \mathcal{X}$ such that $m_S((\circ_{X \in \mathcal{X}^*} X_T)(D_{fail})) \leq \tau$.

Informally, $\mathcal{X}^*$ explains the cause: why S malfunctions for D_{fail}, but not for D_{pass}. More specifically, *failing to satisfy* the profiles of the PVT triplets in $\mathcal{X}^*$ are the causes of malfunction. Furthermore, the transformation functions of the PVT triplets in $\mathcal{X}^*$ suggest the fix: how can we repair D_{fail} to eliminate system malfunction. However, there could be many possible such $\mathcal{X}^*$ and we seek a *minimal* set $\mathcal{X}^*$ such that a transformation for every $X \in \mathcal{X}^*$ is necessary to bring down the malfunction score below the threshold τ.

Definition 11 (Minimal explanation of system malfunction). *Given a system S that malfunctions for D_{fail} and an allowable malfunction threshold τ, an explanation $\mathcal{X}^*$ of S's malfunction for D_{fail} is minimal if $\forall \mathcal{X}' \subset \mathcal{X}^*$ $m_S((\circ_{X \in \mathcal{X}'} X_T)(D_{fail})) > \tau$.*

Note that there could be multiple such minimal explanations and we seek any one of them, as any minimal explanation exposes the causes of mismatch and suggests minimal fixes.

DataPrism requires knowledge of a passing dataset to guide the search for the cause of mismatch between D_{fail} and S. Assuming availability of such knowledge is realistic in many real-world scenarios and has been considered in prior work [26, 50]. For instance, in ML, the training set can be considered as a passing dataset.

Problem 12 (Discovering explanation of mismatch between data and system). *Given a system S that malfunctions for D_{fail} but functions properly for D_{pass}, the problem of discovering the explanation of mismatch between D_{fail} and S is to find a minimal explanation that captures (1) the cause why S malfunctions for D_{fail} but not for D_{pass} and (2) how to repair D_{fail} to remove the malfunction.*

Certain applications allow access to multiple passing datasets or multiple malfunction metrics. DataPrism extends to this setting as well (details are in our technical report [30]).

3 DATA PROFILES, VIOLATION FUNCTIONS, AND TRANSFORMATION FUNCTIONS

We now provide an overview of the data profiles we consider, how we discover them, how we compute the violation scores for a dataset w.r.t. a data profile, and how we apply transformation functions to alter profiles of a dataset. While a multitude of data-profiling primitives exist in the literature, we consider a carefully chosen subset of them that are particularly suitable for modeling issues in data that commonly cause malfunction or failure of a system (Figure 1). We focus on profiles that, by design, can better "discriminate" a pair of datasets as opposed to "generative" profiles (e.g., data distribution) that can profile the data better, but nonetheless are less useful for the task of discriminating between two datasets. However, the DataPrism framework is generic, and other profiles can be plugged into it. In this work, we just pick these profiles to show the efficacy of DataPrism over a set of real-world use cases.

As discussed in Section 2, a PVT triplet encapsulates a profile, and corresponding violation and transformation functions. Figure 1 provides a list of profiles (not exhaustive) along with the data types they support, how to learn their parameters from a given dataset, how to interpret them intuitively, and the corresponding violation and transformation functions. The PVTs are specified according to the data types of the attributes (numerical, categorical, dates, etc.) to reduce the effort of defining PVTs individually for each attribute[1]. In this work, we assume that a profile can be associated with multiple transformation functions (e.g., row 3), but each transformation function can be associated with at most one profile. This assumption helps us to blame a unique profile as the cause of the system malfunction when any of its transformation functions is verified to be a fix. When the assumption does not hold, DataPrism may blame multiple profiles, which are potentially related via a disjunctive relationship, as possible cause. In such cases, some of the reported profiles may be false positives, but the true cause will never be missed (no false negatives).

PVT triplets can be classified in different ways. Based on the strictness of the violation function, they can be classified as follows:

- *Strict*: *All* tuples are expected to satisfy the profile (rows 1 and 2).
- *Thresholded by data coverage:* Certain fraction (θ) of data tuples are allowed to violate the profile (rows 3–6).
- *Thresholded by a parameter:* Some degree of violation is allowed with respect to a specific parameter (α) (rows 7 and 8).

Further, PVT triplets can be classified in two categories based on the nature of the transformation functions:

- *Local* transformation functions can transform a tuple in isolation without the knowledge of how other tuples are being transformed (e.g., row 1). Some local transformation functions only transform the violating tuples (e.g., row 3, transformation (2)), while others transform all values.
- *Global* transformation functions are holistic, as they need the knowledge of how other tuples are being transformed while transforming a tuple (e.g., row 4).

Many transformation functions may exist, including some extreme ones such as removing all tuples from the dataset. Prior data-cleaning techniques can be used as transformations to obey

[1] Specifying different PVTs for each attribute and their combination (not just data types) requires significant user effort but DataPrism is flexible to consider such PVTs.

		Profile	Data type	Discovery over X	Interpretation	Violation by Y	Transformation function
strict	1	$\langle$ Domain, A_j, $\mathbb{S}\rangle$	Categorical	$\mathbb{S} = \bigcup_{t \in X} \{t.A_j\}$	Values are drawn from a specific domain.	$\frac{\sum_{t \in Y} [\![t.A_j \notin \mathbb{S}]\!]}{\lvert Y \rvert}$	Map values outside $\mathbb{S}$ to values in $\mathbb{S}$ using domain knowledge.
	2	$\langle$ DataSize $\rangle$	All	$\lvert D \rvert$	Dataset size.	$\frac{\lvert Y \rvert}{\lvert D \rvert}$	Remove tuples (or add duplicates) to satisfy the size requirements.
thresholded by data coverage	3	$\langle$ Outlier, A_j, O, $\theta\rangle$	All	$\theta = \frac{\sum_{t \in X} [\![O(X.A_j, t.A_j)]\!]}{\lvert X \rvert}$, where O is learned from $X.A_j$'s distribution [37]	Fraction of outliers within an attribute does not exceed a threshold.	$\max\left(0, \frac{\sum_{t \in Y} [\![O(Y.A_j, t.A_j)]\!] - \theta \cdot \lvert Y \rvert}{\lvert Y \rvert \cdot (1-\theta)}\right)$	(1) Replace outliers with the expected value (mean, median, mode) of the attribute. (2) Map all values above (below) the maximum (minimum) limit with highest (lowest) valid value.
	4	$\langle$ Missing, A_j, $\theta\rangle$	All	$\theta = \frac{\sum_{t \in X} [\![t.A_j = \text{NULL}]\!]}{\lvert X \rvert}$	Fraction of missing values within an attribute does not exceed a threshold.	$\max\left(0, \frac{\sum_{t \in Y} [\![t.A_j = \text{NULL}]\!] - \theta \cdot \lvert Y \rvert}{\lvert Y \rvert \cdot (1-\theta)}\right)$	Use missing value imputation techniques.
	5	$\langle$ Selectivity, $\mathbb{P}$, $\theta\rangle$	All	$\theta = \frac{\lvert \sigma_{\mathbb{P}}(X) \rvert}{\lvert X \rvert}$	Fraction of tuples satisfying a given constraint (selection predicate) does not exceed a threshold.	$\max\left(0, \frac{\lvert \sigma_{\mathbb{P}}(Y) \rvert - \theta \cdot \lvert D \rvert}{\lvert D \rvert \cdot (1-\theta)}\right)$	Undersample tuples that satisfy the predicate $\mathbb{P}$.
	6	$\langle$ Functional, A_i, A_j, $\theta\rangle$	All	θ is the fraction of tuples that satisfy $A_i \rightarrow A_j$	Fraction of tuples violating a functional dependency does not exceed a threshold.	$\max\left(0, \frac{\sum_{t \in Y} [\![t.A_i \not\rightarrow t.A_j]\!]}{\lvert Y \rvert}\right)$	Modify the violating tuples using data cleaning techniques [61].
thresholded by parameter	7	$\langle$ Indep, A_j, A_k, $\alpha\rangle$	Categorical	α denotes Chi-squared statistic between $X.A_j$ and $X.A_k$	χ^2 statistic between a pair of attributes is below a threshold with a p-value ≤ 0.05.	$1 - e^{-\max(0, \chi^2(Y.A_j, Y.A_k) - \alpha)}$	Modify attribute values to remove/reduce dependence.
	8	$\langle$ ConfCons, $\vec{A}$, $\vec{w}$, $\alpha\rangle$	Numerical	Coefficients in $\vec{w}$ and threshold α are learned using [27] such that $\forall t \in X \; \lvert \vec{w} \cdot t.\vec{A} \rvert \leq \alpha$	There is a linear arithmetic relationship among the numerical attributes, captured by low-variance (α) projections ($\vec{w}$).	$\frac{\sum_{t \in Y} \max(0, \vec{w} \cdot t.\vec{A} - \alpha, -\alpha - \vec{w} \cdot t.\vec{A})}{\sigma(\vec{w} \cdot X.\vec{A})}$	Perform linear transformation over the numerical attributes to satisfy the conformance constraint.

Figure 1: A list of PVT triplets that we consider in this paper, their syntax, and semantics.

corresponding profiles (e.g., [69] to break correlation). While transformation functions should be designed to *minimally* alter the data, the design of minimal transformation functions is orthogonal to DataPrism. However, the choice of transformation function for a specific profile has no impact on the correctness of DataPrism. Nevertheless, with respect to a dataset, some transformations may incur less cost (e.g., fewer data alterations) or offer greater benefits (e.g., fewer number of interventions). When multiple transformations are available, DataPrism estimates their likelihood to reduce system malfunction using heuristics, which we discuss in Section 4.

Example 13. ***Domain*** *requires two parameters: (1) an attribute $A_j \in \mathcal{R}(D)$, and (2) a set $\mathbb{S}$ specifying its domain. A dataset D satisfies $\langle$Domain, $A_j, \mathbb{S}\rangle$ if $\forall t \in D \;\; t.A_j \in \mathbb{S}$. The profile $\langle$Domain, $A_j, \mathbb{S}\rangle$ is minimal w.r.t. D if $\nexists \mathbb{S}' \subset \mathbb{S}$ s.t. D satisfies the profile $\langle$Domain, $A_j, \mathbb{S}'\rangle$. The technique for discovering a domain $\mathbb{S}$ varies depending on the data type of the attribute. Row 1 shows one, others are in [30].*

$People_{fail}$ (Figure 2) satisfies $\langle$Domain, `gender`*, $\{F, M\}\rangle$, as all tuples draw values from $\{F, M\}$ for the attribute* `gender`*. Our case studies of* `Sentiment Prediction` *and* `Cardiovascular Disease Prediction` *show the application of the profile Domain (Section 5).*

Example 14. ***Indep*** *requires three parameters: two attributes $A_j, A_k \in \mathcal{R}(D)$, and a real value α. A dataset D satisfies the profile $\langle$Indep, $A_j, A_k, \alpha\rangle$ if the dependency between $D.A_j$ and $D.A_k$ does not exceed α. Different techniques exist to quantify the dependency. Row 7 models dependency using correlation (others are in [30]).*

$\langle$Indep, `race`*,* `high_expenditure`*, 0.67$\rangle$ is satisfied by $People_{fail}$ using the PVT triplet of row 7, as χ^2-statistic between* `race` *and* `high_expenditure` *over $People_{fail}$ is 0.67. We show its application in our case study involving the task of* `Income Prediction` *in Section 5.*

Figure 1 describes Domain (row 1) and Indep (row 7) for categorical attributes. Other variations for other data-types are discussed in [30]. While the profiles in Figure 1 are defined over the entire data, analogous to conditional functional dependency [25], an extension to consider is *conditional* profiles, where only a subset of the data satisfy the profiles. E.g., all PVTs on a subset of the data having gender=F are considered to be conditioned on gender=F.

4 INTERVENTION ALGORITHM

We now describe our intervention algorithm to explain the mismatch between a dataset and a system malfunctioning on that dataset. Our algorithm considers a failing and a passing dataset as input and reports a *collection* of PVT triplets (or simply PVTs) as the explanation (cause and fix) of the observed mismatch. To this end, we first identify a set of *discriminative* PVTs—whose profiles take different values in the failing and passing datasets—as potential explanation units, and then intervene on the failing dataset to alter the profiles and observe change in system malfunction. Our greedy strategy, DataPrism iteratively intervenes one PVT at a time based on their estimated likelihood to reduce system malfunction. We start with an example scenario to demonstrate how DataPrism works and then proceed to describe the algorithm.

4.1 Example Scenario

Consider the task of predicting the attribute `high_expenditure` to determine if a customer should get a discount (Example 1). The system calculates bias of the trained classifier against the unprivileged groups (measured using disparate impact [40]) as its malfunction score. We seek the causes of mismatch between this prediction pipeline and $People_{fail}$ (Figure 2), for which the pipeline fails with a malfunction score of 0.75. We assume the knowledge of $People_{pass}$ (Figure 3), for which the malfunction score is 0.15. The goal is to identify a minimal set of PVTs whose transformation functions bring down the malfunction score of $People_{fail}$ below 0.20.

id	name	gender	age	race	zip code	phone	high expenditure
t_1	Shanice Johnson	F	45	A	01004	2088556597	no
t_2	DeShawn Bad	M	40	A	01004	2085374523	no
t_3	Malik Ayer	M	60	A	01005	2766465009	no
t_4	Dustin Jenner	M	22	W	01009	7874891021	yes
t_5	Julietta Brown	F	41	W	01009		yes
t_6	Molly Beasley	F	32	W		7872899033	no
t_7	Jake Bloom	M	25	W	01101	4047747803	yes
t_8	Luke Stonewald	M	35	W	01101	4042127741	yes
t_9	Scott Nossenson	M	25	W	01101		yes
t_{10}	Gabe Erwin	M	20	W		4048421581	yes

Figure 2: A sample dataset $People_{fail}$ with 10 entities. A logistic regression classifier trained over this dataset discriminates against African Americans (race = 'A') and women (gender = 'F') (Example 1).

(Step 1) To identify the profiles whose parameters differ between $People_{fail}$ and $People_{pass}$, DataPrism identifies the exhaustive set of PVTs over $People_{pass}$ and $People_{fail}$ and discards the identical ones (in terms of profile-parameter values). We call the PVTs of the passing dataset whose profile-parameter values differ from the failing one *discriminative* PVTs. Figure 4 lists a few profiles of the discriminative PVTs wrt $People_{pass}$ and $People_{fail}$.

(Step 2) Next, DataPrism ranks discriminative PVTs based on their likelihood to explain the malfunction. Our intuition here is that if an attribute A is related to the malfunction, then many PVTs containing A in their profiles would differ between $People_{fail}$ and $People_{pass}$. Additionally, altering A with respect to one PVT is likely to automatically "fix" other PVTs associated with A.[2] Based on this intuition, DataPrism constructs a *bipartite graph*, called PVT-attribute graph, with discriminative PVTs on one side and data attributes on the other side (Figure 5). In this graph, a PVT X is connected to an attribute A if X_P is defined over A. In this graph, the degree of an attribute A captures the number of discriminative PVTs associated with A. During intervention, DataPrism prioritizes PVTs associated with a high-degree attribute. For instance, since `high_expenditure` has the highest degree in Figure 5, PVTs associated with it are considered for intervention before others.

(Step 3) DataPrism further ranks the subset of the discriminative PVTs that are connected to the highest-degree attributes in the PVT-attribute graph based on their *benefit score*. Benefit score of a PVT X encodes the likelihood of reducing system malfunction when the failing dataset is altered using X_T. The benefit score of X is estimated from (1) the violation score that the failing dataset incurs w.r.t. X_V, and (2) the number of tuples in the failing dataset that are altered by X_T. For example, to break the dependence between `high_expenduture` and `race`, the transformation corresponding to Indep modifies five tuples in $People_{fail}$ by perturbing (adding noise to) `high_expenditure`. In contrast, the transformation for Missing needs to change only one value (t_6 or t_{10}). Since more tuples are affected by the former, it has higher likelihood of reducing the malfunction score. The intuition behind this is that if a transformation alters more tuples in the failing dataset, the more likely it is to reduce the malfunction score. This holds particularly in applications where the system optimizes aggregated statistics such as accuracy, recall, F-score, etc.

(Step 4) DataPrism starts intervening on $People_{fail}$ using the transformation of the PVT corresponding to the profile

[2] DataPrism extends to the case when altering values of A wrt a PVT increases violation wrt some other PVTs, however, may require sub-optimal number of interventions.

id	name	gender	age	race	zip code	phone	high expenditure
t_1	Darin Brust	M	25	W	01004	2088556597	no
t_2	Rosalie Bad	F	22	W	01005		no
t_3	Kristine Hilyard	F	50	W	01004	2766465009	yes
t_4	Chloe Ayer	F	22	A		7874891021	yes
t_5	Julietta Mchugh	F	51	W	01009	9042899033	yes
t_6	Doria Ely	F	32	A	01101		yes
t_7	Kristan Whidden	F	25	W	01101	4047747803	no
t_8	Rene Strelow	M	35	W	01101	6162127741	yes
t_9	Arial Brent	M	45	W	01102	4089065769	yes

Figure 3: A sample dataset $People_{pass}$ with 9 entities. A logistic regression classifier trained over this dataset does not discriminate against any specific race or gender, and, thus, is fair (Example 1).

$\langle$Indep, `race`, `high_expenditure`, 0.04$\rangle$ as its transformation offers the most likely fix. Then, it evaluates the malfunction of the system over the altered version of $People_{fail}$. Breaking the dependence between `high_expenditure` and `race` helps reduce bias in the trained classifier, and, thus, we observe a malfunction score of 0.35 w.r.t. the altered dataset. This exposes the first explanation of malfunction.

(Step 5) DataPrism then removes the processed PVT (Indep) from the PVT-attribute graph, updates the graph according to the altered dataset, and re-iterates steps 2–4. Now the PVT corresponding to the profile Selectivity is considered for intervention as it has the highest benefit score. To do so, DataPrism oversamples tuples corresponding to female customers with `high_expenditure` = *yes*. This time, DataPrism intervenes on the transformed dataset obtained from the previous step. After this transformation, bias of the learned classifier further reduces and the malfunction score falls below the required threshold. Therefore, with these two interventions, DataPrism is able to expose two issues that caused undesirable behavior of the prediction model trained on $People_{fail}$.

(Step 6) DataPrism identifies an initial explanation over two PVTs: Indep and Selectivity. However, to verify whether it is minimal, DataPrism tries to drop from it one PVT at a time to obtain a proper subset of the initial explanation that is also an explanation. This procedure guarantees that the explanation only consists of PVTs that are *necessary*, and, thus, is minimal. In this case, both Indep and Selectivity are necessary, and, thus, are part of the minimal explanation. DataPrism finally reports the following as a minimal explanation of the malfunction, where failure to satisfy the profiles is the cause and the transformations indicate fix (violation and transformation functions are omitted).

$$\{\langle \text{Indep}, \texttt{race}, \texttt{high\_expenditure}, 0.04\rangle,$$
$$\langle \text{Selectivity}, \texttt{gender} = \text{F} \wedge \texttt{high\_expenditure} = \text{yes}, 0.44\rangle\}$$

4.2 Assumptions and Observations

We now proceed to describe our intervention algorithms more formally. We first state our assumptions and then proceed to present our observations that lead to the development of our algorithms.

Assumptions. DataPrism makes the following assumptions:

(A1) The ground-truth explanation of malfunction is captured by at least one of the discriminative PVTs. This assumption is prevalent in software-debugging literature where program predicates are assumed to be expressive enough to capture the root causes [26, 50].

(A2) If the fix corresponds to a composition of transformations, then the malfunction score achieved after applying the composition of transformations is less than the malfunction score achieved after applying any of the constituents, and all these scores are less than the malfunction score of the original dataset. E.g., consider two

$People_{pass}$	$People_{fail}$
⟨Domain, age, [22, 51]⟩	⟨Domain, age, [20, 60]⟩
⟨Missing, `zip_code`, 0.11⟩	⟨Missing, `zip_code`, 0.2⟩
⟨Indep, `race`, `high_expenditure`, 0.04⟩	⟨Indep, `race`, `high_expenditure`, 0.67⟩
⟨Selectivity, `gender` = F ∧`high_expenditure` = yes, 0.44⟩	⟨Selectivity, `gender` = F ∧`high_expenditure` = yes, 0.1⟩

Figure 4: A list of PVTs that discriminate $People_{pass}$ (Figure 3) and $People_{fail}$ (Figure 2) based on the scenario of Example 1 . We omit the violation and transformation functions for ease of exposition.

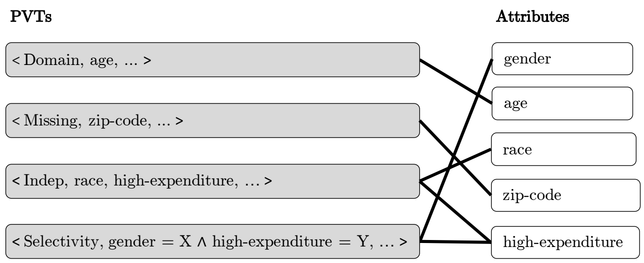

Figure 5: PVT-attribute graph. The attribute `high_expenditure` is associated with two discriminative PVTs. For ease of exposition, we only show profile within a PVT to denote the entire PVT.

discriminative PVTs X and Y and a failing dataset D_{fail}. Our assumption is that if $\{X, Y\}$ corresponds to a minimal explanation, then $m_S((Y_T \circ X_T)(D_{fail})) < m_S(X_T(D_{fail})) < m_S(D_{fail})$ and $m_S((Y_T \circ X_T)(D_{fail})) < m_S(Y_T(D_{fail})) < m_S(D_{fail})$. Intuitively, this assumption states that X and Y have consistent (independent) effect on reducing the malfunction score, regardless of whether they are intervened together or individually in any order. This assumption generally holds when different causes of malfunction involve different attributes. Consider a failing dataset that contains formatting error in address and correlation between race and income. Here, the transformation to fix the formatting issues in address does not affect race or income. Therefore, applying transformations in any order would have similar effect on system malfunction. We empirically observed that this assumption generally holds for most real-world scenarios.

Observations. We make the following observations:

(O1) If the ground-truth explanation of malfunction corresponds to an attribute, then multiple PVTs that involve the same attribute are likely to differ across the passing and failing datasets. This observation motivates us to prioritize interventions based on PVTs that are associated with high-degree attributes in the PVT-attribute graph. Additionally, intervening on the data based on one such PVT is likely to result in an automatic "fix" of other PVTs connecting via the high-degree attribute. For example, adding noise to `high_expenditure` in Example 1 breaks its dependence with not only `race` but also with other attributes.

(O2) The PVT for which the failing dataset incurs higher violation score is more likely to be a potential explanation of malfunction.

(O3) A transformation function that affects a large number of data tuples is likely to result in a higher change in the malfunction score, after the transformation is applied.

PVT-attribute graph. DataPrism leverages observation O1 by constructing a bipartite PVT-attribute graph (G_{PA}), with all attributes $A \in \mathcal{R}(D)$ as nodes on one side and all discriminative PVTs $X \in \mathcal{X}$ on the other side. An attribute A is connected to a PVT X if and only if X_P has A as one of its parameters.

Algorithm 1: DataPrism

Input: Failing dataset D_{fail}, passing dataset D_{pass}, malfunction score threshold τ

Output: A minimal explanation set of PVTs $\mathcal{X}^*$

```
1  X_f ← Discover-PVT(D_fail)
2  X_p ← Discover-PVT(D_pass)
3  X_∩ ← X_f ∩ X_p                                  /* Common PVTs */
4  X ← X_p \ X_∩                            /* Discriminative PVTs */
5  G_PA(V_G, E_G) ← Construct-PVT-Attr-Graph(X, D_fail)
6  B ← Calculate-Benefit-Score(X, G_PA, D_fail)
7  X* ← ∅   /* Initialize minimal explanation set to be empty */
8  D ← D_fail        /* Initialize dataset to the failing dataset */
9  while m_S(D) > τ do
10     X_hda = {X∈X | (X, A)∈E_G ∧ A = arg max_{A∈R(D)} deg_G(A)}
               /* PVTs adjacent to high-degree attributes in G_PA */
11     X = arg max_{X∈X_hda} B(X)          /* Highest-benefit PVT */
12     Δ ← m_S(D) − m_S(X_T(D))           /* Malfunction reduction */
13     G_PA ← G_PA.Remove(X)                        /* Update G_PA */
14     if Δ > 0 then                        /* Reduces malfunction */
15         D ← X_T(D)                       /* Apply transformation */
16         G_PA.Update(D)  /* Update the PVT-attribute graph */
17         B.Update(D)                      /* Update benefit scores */
18         X* ← X* ∪ {X}               /* Add P to explanation set */
19         X ← X \ {X}              /* Remove P from the candidates */
20 X* = Make-Minimal(X*)                   /* Obtain minimality of X* */
21 return X*                         /* X* is a minimal explanation */
```

E.g., Figure 5 shows the PVT-attribute graph w.r.t. $People_{fail}$ and $People_{pass}$ (Example 1). In this graph, the PVT corresponding to ⟨Indep, `race`, `high_expenditure`⟩ is connected to two attributes, `race` and `high_expenditure`. Intuitively, this graph captures the dependence relationship between PVTs and attributes, where an intervention with respect to a PVT X modifies an attribute A connected to it. If this intervention reduces the malfunction score then it could possibly fix other PVTs that are connected to A.

Benefit score calculation. DataPrism uses the aforementioned observations to compute a benefit score for each PVT to model their likelihood of reducing system malfunction if the corresponding transformation is used to modify the failing dataset D_{fail}. Intuitively, it assigns a high score to a PVT with a high violation score (O2) and if the corresponding transformation function modifies a large number of tuples in the dataset (O3). Formally, the benefit score of a PVT X is defined as the product of violation score of D_{fail} w.r.t. X_V and the "coverage" of X_T. The coverage of X_T is defined as the fraction of tuples that it modifies. Note that the benefit calculation procedure acts as a proxy of the likelihood of a PVT to offer an explanation, without actually applying any intervention.

4.3 Greedy Approach

Algorithm 1 presents the pseudocode of our greedy technique DataPrism, which takes a passing dataset D_{pass} and a failing dataset D_{fail} as input and returns the set of PVTs that corresponds to a minimal explanation of system malfunction.

Lines 1-2 Identify two sets of PVTs $\mathcal{X}_f$ and $\mathcal{X}_p$ satisfied by D_{fail} and D_{pass}, respectively.

Lines 3-4 Discard the PVTs $\mathcal{X}_f \cap \mathcal{X}_p$ from $\mathcal{X}_p$ and consider the remaining discriminative ones $\mathcal{X} \equiv \mathcal{X}_p \setminus \mathcal{X}_f$ as candidates for potential explanation of system malfunction.
Line 5 Compute the PVT-attribute graph G_{PA}, where the candidate PVTs $\mathcal{X}$ correspond to nodes on one side and the data attributes correspond to nodes on the other side.
Line 6 Calculate the benefit score of each discriminative PVT $X \in \mathcal{X}$ w.r.t. D_{fail}. This procedure relies on the violation score using the violation function of the PVT and the coverage of the corresponding transformation function over D_{fail}.
Line 7-8 Initialize the solution set $\mathcal{X}^*$ to $\emptyset$ and the dataset to perform intervention on D to the failing dataset $\mathcal{D}_{fail}$. In subsequent steps, $\mathcal{X}^*$ will converge to a minimal explanation set and D will be transformed to a dataset for which the system passes.
Line 9 Iterate over the candidate PVTs $\mathcal{X}$ until the dataset D (which is being transformed iteratively) incurs an acceptable violation score (less than the allowable threshold τ).
Line 10 Identify the subset of PVTs $\mathcal{X}_{\mathsf{hda}} \subseteq \mathcal{X}$ such that all $X \in \mathcal{X}_{\mathsf{hda}}$ are adjacent to at least one of the highest degree attributes in the current PVT-attribute graph (Observation O1).
Line 11 Choose the PVT $X \in \mathcal{X}_{\mathsf{hda}}$ that has the maximum benefit.
Line 12 Calculate the reduction in malfunction score if the dataset D is transformed according to the transformation X_T.
Line 13 Remove X from G_{PA} as it has been explored.
Lines 14-19 If the malfunction score reduces over $X_T(D)$, then X is added to the solution set $\mathcal{X}^*$, and D is updated to $X_T(D)$, which is then used to update the PVT-attribute graph and benefit of each PVT. The update procedure recalculates the benefit scores of all PVTs that are connected to the attributes adjacent to X in G_{PA}.
Line 20 Post-process the set $\mathcal{X}^*$ to identify a minimal subset that ensure that malfunction score remains less than the threshold τ. This procedure iteratively removes one PVT at a time (say X) from $\mathcal{X}^*$ and recalculates the malfunction score over the failing dataset D_{fail} transformed according to the transformation functions of the PVTs in the set $\mathcal{X}' = \mathcal{X}^* \setminus \{X\}$. If the transformed dataset incurs a violation score less than τ then $\mathcal{X}^*$ is replaced with $\mathcal{X}'$.

Generalization. Algorithm 1 assumes that the system is evaluated on a single malfunction metric. DataPrism extends to multiple notions of malfunction by calculating malfunction reduction wrt each notion in line 12 (say Δ_i for malfunction m_S^i) and changing line 14 to ensure that malfunction score is reduced wrt some notion ($\exists j \;\; \Delta_j > 0$) without hurting others ($\Delta_i \geq 0, \forall i$). Details are in [30].

5 EXPERIMENTAL EVALUATION

Our experiments aim to answer the following research questions:

- (RQ1) In practice, can DataPrism *correctly* identify the cause and corresponding fix of mismatch between a system and a dataset for which the system fails? (Section 5.1)
- (RQ2) How *sensitive* is DataPrism to the choices of system parameters and facets of the framework? (Sections 5.2 and 5.3)
- (RQ3) Is DataPrism *scalable* and *robust* to varying problem complexity? (Section 5.4)
- (RQ4) How *efficient* is DataPrism compared to other alternative techniques? (Sections 5.1 and 5.4)

Baselines. Since no prior work supports dataset-level interventions guided by PVTs, we adapted state-of-the-art interventional debugging and explanation techniques that aim to explain the cause of system failure. To adapt these approaches to our problem setting, we replaced their intervention mechanism with the transformation functions we use in DataPrism. We consider three baselines:

– `BugDoc` [52] is a recent debugging technique that explores different parameter configurations of the system. We adapt `BugDoc` to consider each PVT as a parameter of the system configuration and interventions as modified configurations.

– `Anchors` [66] is a local explanation technique that explains individual predictions of a classifier based on a surrogate model. We consider each intervention as a data point and PVTs as features for the surrogate model to predict whether the system malfunctions over a dataset or not.

– `GroupTest (GT)` [22] is an adaptive group-testing approach that performs group interventions to expose the mismatch between the input dataset and the system. It recursively partitions the PVTs until a PVT that reduces system malfunction is identified. However, it requires additional assumptions, which we discuss in our case study involving cardiovascular-disease prediction (Section 5.1.3).

Settings. We consider open-source implementations of `Anchors` and `BugDoc` with the default parameter settings. Since `BugDoc` requires a budget on the number of interventions, we specify the smallest value that guarantees `BugDoc` to expose the ground-truth cause as budget. We implemented DataPrism in Python 3.6.

DataPrism uses attribute types (text, categorical, or numerical) to construct PVTs of each attribute. If an attribute contains values of heterogenous types, then DataPrism constructs different PVTs for each homogenous partition. E.g., for an attribute with 90% numerical and 10% text values, DataPrism generates two Domain PVTs, one for text and one for numerical values. DataPrism generates all PVTs discussed in Section 3. Additionally, DataPrism generates conditional PVTs by considering the subset of tuples that share the same value for other categorical attribute(s). E.g., consider a dataset with two attributes: $X \in \{x_1, \dots, x_4\}$ and $Y \in \{y_1, \dots, y_{10}\}$. DataPrism generates 14 conditional PVTs: 4 pivoting on X and 10 pivoting on Y. One such conditional PVT is Domain of Y over the tuples where $X = x_2$.

For synthetic pipelines (Section 5.4) we generate numerical attributes in a dataset within the range $[1, 10]$. We inject error in the data by randomly choosing noisy attributes and then modifying their data distribution. Unless otherwise specified, all datasets for the synthetic pipelines contain 10 attributes, the ground-truth cause of malfunction is characterized by the Domain PVT, and the malfunction-score threshold is 0.

5.1 Real-world Case Studies

We design seven case studies focusing on real applications—consisting of a diverse set of ML models [3, 4, 59] and data-analysis tasks—as opaque systems over real-world datasets. Figure 6 presents a summary of our evaluation results.

5.1.1 *Sentiment Prediction.*

The system in this study predicts sentiment of input text (reviews/tweets) and we consider misclassification rate as the malfunction score. Internally, it uses flair [4], a neural-network model. The system assumes that for the target attribute, '1' indicates positive sentiment and '-1' indicates negative sentiment. We test the system over two datasets: `IMDb` [42] ($\sim 50K$

	Number of Interventions				Execution Time (seconds)			
Case study	DP	BugDoc	Anchors	GT	DP	BugDoc	Anchors	GT
Sentiment	4	4	303	4	25.1	64.6	4594.9	21.2
Income	2	20	103	10	11.8	20.0	195.5	98.4
Cardiovascular	5	100	3503	–	7.6	62.1	8602.9	–
Flights	5	78	601	–	27.5	1037.7	10509.2	–
Amazon	2	8	303	4	7.3	14.80	207.58	8.1
Open Data	8	14	102	–	1.34	7.64	73.97	–
Physicians	30	46	104	47	2.25	78.29	34.5	18.2

Figure 6: Comparison wrt number of interventions and execution time of DataPrism (denoted by DP) with other baselines. '–' denotes that the technique failed to identify the cause of malfunction because assumption A3 did not hold.

tuples) and twitter [71] ($\sim$ 1.6M tuples). The malfunction score of the system over IMDb is only 0.09 while it is 1.0 over twitter.

We consider IMDb as the passing dataset and twitter as the failing dataset and use DataPrism to find the cause of mismatch between twitter and the system. DataPrism identifies a total of 4 discriminative PVTs, including Domain of the target attribute that differs between the two datasets: $\{-1, 1\}$ for IMDb and $\{0, 4\}$ for twitter. DataPrism performs four interventions and finds that a 64% reduction in malfunction score is achieved when the following transformation is applied on the target attribute in twitter: map $0 \rightarrow -1$ and $4 \rightarrow 1$. Consequently, DataPrism reports this as a cause of the malfunction. With a close investigation, we found that twitter uses '4' to denote positive and '0' to denote negative sentiment [71], which matches the reported explanation.

When compared with other baselines, GroupTest and BugDoc require 4 interventions to explain the cause while and Anchors requires 303 interventions. Note that Anchors calculates system malfunction on many datasets that are transformed according to various local perturbations guided by the PVTs.

5.1.2 Income Prediction. The issue here is unfairness in ML predictions. The system trains a random forest classifier [59] to predict the income of individuals. We use normalized disparate impact [40] (a metric to measure discrimination) of the trained classifier wrt the sensitive attribute (sex) as the malfunction score.

We create two datasets—from subsets of census records [23] that contain demographic attributes of individuals—and manually add noise to one of them to break the correlation between income (target attribute) and sex (sensitive attribute). The system incurs a malfunction score of 0.195 for the noisy dataset and 0.58 for the other. The malfunction (unfairness) here is caused by the existence of correlation between income and sex.

DataPrism identifies 132 discriminative PVTs and constructs a PVT-attribute graph, where income has a degree of 19 and all other attributes have smaller degrees. As a result, DataPrism prioritizes exploring PVTs that involve income. The transformations corresponding to the PVT Indep between income and other attributes break the dependence between income and all other attributes, thereby, reducing the malfunction score to 0.32. DataPrism requires only 2 interventions to discover the cause of malfunction. This study demonstrates the effectiveness of intervening on PVTs in non-increasing order of benefit (observations O2 and O3).

GroupTest requires 10 interventions, which is in the order of $\log n$ where n is the number of discriminative PVTs (132 in this case). BugDoc and Anchors do not identify discriminative PVTs explicitly and consider all PVTs (136 in this case) for intervention. Anchors performs 103 local interventions to find the correct cause, while BugDoc finds a valid cause with an intervention budget of 20.

5.1.3 Cardiovascular Disease Prediction. This system trains an AdaBoost classifier [3] on patients' medical records [15]—containing age, height (in centimeters), weight etc.—to predict patients with disease. The pipeline returns the additive inverse of recall as the malfunction score. We tested the pipeline with two datasets generated through a random selection of tuples: (1) the passing dataset satisfies the format assumptions of the pipeline; (2) for the failing dataset we inject noise by converting height to inches.

DataPrism identifies 87 discriminative PVTs and considers the Domain of height as the fifth intervention, where it alters the failing dataset by applying a linear transformation, which reduces the malfunction from 0.71 to 0.30. This explanation matches the ground-truth cause of malfunction. In contrast, BugDoc and Anchors perform 100 and 3500 interventions, respectively. GroupTest does not identify the ground-truth cause of system malfunction because performing groups of transformations together increases system malfunction. Specifically, we observe that the malfunction score with a composition of transformation functions is higher than that of the original dataset if the composition involves the PVT Indep. This behavior is observed because adding noise to intervene with respect to Indep worsens the classifier performance. GroupTest is applicable only in cases where different groups of interventions do not worsen the malfunction score, i.e., if X and Y correspond to any two discriminative PVTs, then $m_S((Y_T \circ X_T)(D_{fail})) < m_S(D_{fail})$ if $m_S(Y_T(D_{fail})) < m_S(D_{fail})$ or $m_S(X_T(D_{fail})) < m_S(D_{fail})$.

Conditional PVTs. We consider a modified failing dataset where the heights of admitted ($inactive = 1$) individuals were recorded in inches but others were in centimeters. The malfunction score in the failing dataset is 0.55 and the goal is to reduce it to below 0.30. In this case, we augmented the PVTs discussed in Figure 1 with conditional PVTs. The number of conditional PVTs is about 12 times the number of the original PVTs. DataPrism requires 12 interventions to identify that transforming the height of individuals with $inactive = 1$ reduces the system malfunction to 0.30. This explanation matches the ground-truth cause of the malfunction. During the search process, DataPrism prioritizes PVTs that contain height before others, as height has the highest degree in the PVT-attribute graph. However, transformations involving all tuples have a higher benefit and are considered before the conditional PVTs. Therefore, it explores multiple different conditional PVTs that involve height to identify that only the subset having $inactive = 1$ should be transformed.

5.1.4 Flight Delay Prediction. This system considers a logistic regression based pipeline trained by Fariha et al. [27] to predict flight delays. The pipeline is trained on a dataset containing more than 5.4M tuples and is tested on two different datasets corresponding to daytime flights and overnight flights, respectively [27]. The system returns mean absolute error (MAE) of the predicted delay for the test dataset. We observe that the unnormalized malfunction score is less than 20 for daytime flights (passing dataset) and 81 for overnight flights (failing dataset). As reported in [27], the ground-truth explanation of system malfunction is the violation of conformance constraints by certain overnight flights, where the arrival time (next day) is *before* the departure time (previous day). DataPrism identifies 81 discriminative PVTs on these datasets and the ground-truth cause of malfunction is ranked 8 according to the benefit of discriminative PVTs, which is identified as the cause of system malfunction in fewer than 10 interventions. In contrast, violation of conformance constraints was identified as a root cause by `BugDoc` and `Anchors` in 78 and 601 interventions, respectively. `GroupTest` does not apply to this dataset because intervening on multiple PVTs worsens system malfunction even if one of the involved PVTs is the ground-truth cause of malfunction.

5.1.5 Amazon Entity Linking. The system comprises of an NLP pipeline that identifies entities in the input text and maps them to a knowledge graph. The pipeline returns two malfunction scores, where the first score is 1 if the average time taken per row is more than 200ms, and the second malfunction score is 1 if the pipeline fails due to timeout (runs for more than 30 min). We test the system with a dataset containing short text like Amazon product titles (~ 10K tuples) as a passing dataset.

A dataset containing Amazon product reviews (~ 100K tuples) is considered failing as the pipeline's runtime exceeded 30 minutes, incurring a malfunction score of 1 wrt both criteria. The larger text length of reviews and size of the dataset are the ground-truth causes of malfunction.

DataPrism identifies numerous discriminative PVTs, among which the PVT DataSize is connected to all attributes in the PVT-attribute graph, thereby, ranking it as the most beneficial PVT. DataPrism identifies it as a cause of malfunction in the first intervention. After performing this intervention, it ranks the length of review text as the most beneficial PVT and considers it for intervention. Therefore, DataPrism identifies both ground-truth causes of timeout in 2 interventions. `GroupTest` identifies the ground-truth cause in 4 interventions, while `BugDoc` and `Anchors` require 8 and 303 interventions, respectively.

5.1.6 Open data analysis and visualization. The system in this study clusters different locations in the input dataset according to their attributes and generates a visualization. The system extracts first three characters of telephone number as area code, which is used for clustering. The dataset from NYC Open Data repository contains telephone numbers in different formats. For example, many tuples contain the area code in parentheses, e.g., (213) 352-0235. For such tuples, area code is incorrectly identified as '(21', which is not numerical. This causes the distance estimation function of the clustering algorithm to throw an exception, causing a crash. The system returns a malfunction score of 1 if it crashes or the clusters are inaccurate, and 0 otherwise. We tested the pipeline by considering the aforementioned dataset as failing, and a small cleaned dataset as passing with a malfunction threshold of 0.

DataPrism identifies different domains for telephone number (captured by Domain) as one of the discriminative PVTs. The subsequent components intervene on this PVT in fewer than 8 interventions. The corresponding transformation of removing parentheses from telephone numbers is identified as the ground-truth transformation. `GroupTest` fails to identify the ground-truth explanation because the simultaneous transformations lead to inaccurate clustering output. `BugDoc` finds the ground truth in 14 interventions, where `Anchors` makes more than 100 interventions only to report an incorrect cause, as it does not operate on non-numeric PVTs.

Application	DataPrism	DataPrism$_{No-G}$	DataPrism$_{No-B}$
Sentiment	4	4	4
Income	2	15	5
Cardiovascular	5	8	60
Flights	5	14	20
Amazon	2	2	3
Open Data	8	5	9
Physicians	30	75	63

Figure 7: Comparison of number of interventions

Conditional PVTs. To test the effect of conditional PVTs, we consider a modified pipeline that identifies area code from the postal code of a tuple. If the postal code is missing, it uses telephone number to identify the area code. The noisy dataset for this study is generated by randomly removing postal code of some tuples. In this case, the ground-truth explanation is a conditional PVT that transforms telephone number of tuples where postal code is missing. Transforming all tuples is a valid intervention that reduces system malfunction, but is not minimal. DataPrism identifies that transforming all tuples is sufficient to reduce system malfunction in 8 interventions but requires 3 additional interventions to identify the minimal cause. Therefore, DataPrism successfully identifies the ground-truth cause of malfunction in 11 interventions.

5.1.7 Physicians: Data Integration. The system here ensures data quality by testing functional dependencies involving zip code, state, and county such that two tuples with the same zip code must have the same state and county (Zip Code→ State, County), and two tuples with the same county must have the same state. It outputs the fraction of tuples that do not satisfy these functional dependencies as the malfunction score. Physicians dataset has a number of data-quality issues like '0' is mistakenly written as 'o', 'xl' instead of 'al' (Alaska), and so on. We construct two datasets through a random selection of tuples, considering one of them as failing, and repair the other using HoloClean [61] to be used as a passing dataset.

The failing (passing) dataset returns a malfunction score of 0.12 (0). We test DataPrism with a malfunction threshold of 0. The minimal ground-truth cause of malfunction comprises of violation of two functional dependencies (1) ZipCode → CountyName, and (2) CountyName → State. DataPrism returns the ground truth cause of malfunction in fewer than 30 interventions. Intervening with respect to the functional dependency "ZipCode → CountyName" alone reduces malfunction to 0.02 and it is identified as one of the causes of malfunction in 11 interventions. DataPrism requires the rest of the interventions to identify the second PVT to reduce malfunction to 0. `Anchors` finds the ground-truth in 104 interventions while `BugDoc` needs 46.

5.2 Ablation Study

In this section, we test the quality of two different variants of DataPrism. (i) DataPrism$_{No-G}$ does not construct the PVT-attribute graph and directly considers all discriminative PVTs for subsequent steps. (ii) DataPrism$_{No-B}$ does not perform benefit calculation and ranks PVTs only based on their degree in the PVT-attribute graph. Figure 7 presents the number of interventions required by these variants. Overall, we observed that DataPrism requires the least number of interventions across all pipelines. Whenever the highest-degree attribute in the PVT-attribute graph correctly captures the ground truth cause of malfunction, ignoring benefit calculation does not worsen the number of required interventions drastically (income case study). However, in the income case study, ignoring the PVT-attribute graph prioritizes the PVTs that have high benefit but are focussed on low-degree attributes. Therefore, DataPrism$_{No-G}$ requires 15 interventions as compared to 2 interventions by DataPrism. The varied advantages of PVT-attribute graph and benefit calculation across different scenarios justify the two-step procedure of DataPrism. The only case where ignoring the PVT-attribute graph reduces the number of interventions is open data pipeline, where observation O1 does not hold (the Domain of telephone numbers has a smaller degree than other attributes).

Efficiency. Figure 6 presents the execution time of the techniques for the real-world applications. DataPrism is highly efficient and require less than 30 seconds to find the ground-truth cause of malfunction. In contrast, `Anchors` is extremely inefficient, needing more than 143 minutes for `cardiovascular`, while `BugDoc` and `GrpTest` explain the malfunction within 100 seconds.

Key takeaways. Among all real-world case studies, DataPrism requires the fewest interventions to explain the cause of malfunction. `Anchors` requires the highest number of interventions, as it performs many local transformations (small changes to profiles of the failing dataset) to identify the cause of failure. `BugDoc` optimizes interventions by leveraging combinatorial design: it requires more interventions than DataPrism but fewer than `Anchors`. `GroupTest` requires fewer interventions than `BugDoc` and `Anchors` whenever it is applicable. `GroupTest` assumes that intervening groups of PVTs improve system malfunction if any of the constituent PVT reduces system malfunction when applied individually.

5.3 Effect of Malfunction Threshold

In this experiment, we test the effect of varying the malfunction threshold on the number of required interventions. First, we test the cardiovascular pipeline with malfunction threshold varied from 0 to the malfunction of the failing dataset (in intervals of 0.10). In this case, failing dataset has a malfunction of 0.71 and no combination of transformations achieves less than 0.30 malfunction. Whenever the malfunction threshold is varied in the range $(0.30, 0.70]$, DataPrism identifies the ground truth cause of malfunction correctly in fewer than 5 interventions. However, when the malfunction threshold is in the range $[0, 0.30]$, our algorithm returns a PVT that transforms the height from inches to centimeters. This PVT reduces the system malfunction to 0.31 and it is the minimum achievable malfunction for this dataset. Therefore, DataPrism's explanation guarantees a minimal set of PVTs having the minimum achievable malfunction if the input requirement is not feasible. We observe a similar trend in open data case study, where higher malfunction threshold requires fewer interventions as compared to lower values of the threshold.

To further investigate the effect of malfunction threshold, we considered synthetic pipelines where a system fails due to a) wrong domain of input data and b) missing data. In this experiment, all values of malfunction threshold are feasible. Figure 9 shows the effect of increasing malfunction threshold on the number of required interventions. Overall, the number of interventions is stable across a small change in the threshold and it shows a downward trend on average. This evaluation justifies that the effort spent by DataPrism reduces as the malfunction threshold increases.

5.4 Scalability and Robustness

In this experiment, we test the effect of different parameters on the quality of the identified explanation, number of required interventions, and running time of DataPrism. We investigate several configurations by varying the number of data attributes, number of discriminative PVTs, and type of ground truth cause of malfunction.

5.4.1 Effect of the Number of Attributes and PVTs. This experiment tests the effect of the number of dataset attributes and the number of discriminative PVTs on the efficacy of DataPrism, and contrasts those with other state-of-the-art baselines for synthetically generated pipelines. We also investigate the influence of the number of PVTs involved in the root causes and their interactions on the number of interventions each method requires.

Figure 8(a) presents the effect of changing the number of attributes in the datasets on the number of required interventions. DataPrism requires fewer than 5 interventions on average. In contrast, `BugDoc` and `Anchor` require orders of magnitude more interventions. The number of interventions required by `BugDoc` grows linearly with the number of attributes. At the same time, `Anchor` perturbs all PVTs to solve a multi-armed bandit problem: the more PVTs affect the pipeline errors, the more interventions are needed. `GroupTest` requires more interventions than DataPrism, and grows logarithmically with the number of data attributes.

Figure 8(b) depicts the effect of the number of discriminative PVTs on the number of required interventions. DataPrism shows superior performance, requiring fewer than 10 interventions even with more than 100 discriminative PVTs. Here, we observe trends similar to the one in Figure 8(a) for other baselines as the number of PVTs are positively correlated with the number of attributes.

5.4.2 Effect of the size of ground-truth cause. The pipelines presented in Figures 8(a) and 8(b) have a single PVT as the ground-truth cause of the malfunction. In Figure 8(c), we fix the number of attributes to 15 and the number of discriminative PVTs between the passing and the failing datasets to 136. We modify the ground-truth cause to be a conjunction over a set of PVTs of varying cardinalities. We find that the cardinality of the root-cause set (length of the conjunctive cause) does not impact the number of interventions as much as the number of attributes and the number of discriminative PVTs do. However, having more than one cause for malfunction (i.e., a disjunctive cause) requires many more interventions for `Anchor` and `GroupTest`, as shown in Figure 8(d). DataPrism still needs orders of magnitude fewer interventions than these other approaches, although the probability of failing to find any feasible transformation, which decreases malfunctions scores, increases with the number of possible root causes within the disjunction.

We performed this analysis on another set of synthetic pipelines, where a constant fraction of values are deleted from the noisy dataset and the system returns the fraction of missing values as the malfunction score. We observed similar patterns for all the experiments. The average number of interventions for all methods increases as the number of discriminative PVTs increases..

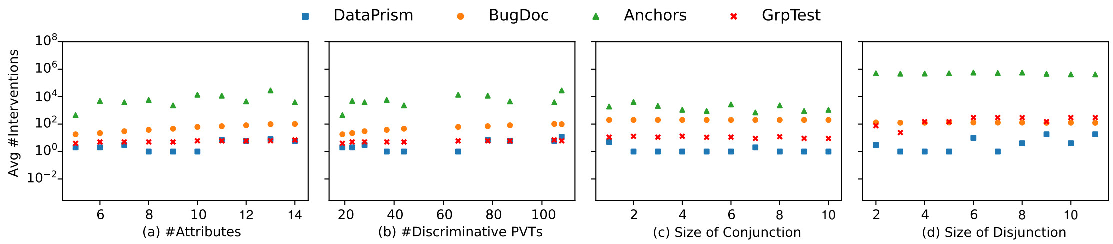

Figure 8: Average number of interventions required by DataPrism and three other techniques for varying number of attributes, discriminative PVTs, size of single conjunctive root causes, and size of disjunctive root causes. Ground-truth cause of malfunction: Domain PVT

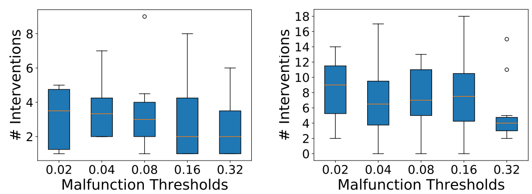

Figure 9: #Interventions for varying malfunction threshold for two data issues: (Left) out-of-domain values, (right) missing values

Scalability. We compare running time of different approaches with increasing number of attributes and discriminative PVTs. The time required by DataPrism to explain the malfunction grows sublinearly in the number of attributes and discriminative PVTs. We observe a similar trend of the number of required interventions on varying these parameters. This experiment demonstrates that DataPrism requires fewer than $O(|\mathcal{X}|)$ interventions in practice (where $\mathcal{X}$ denotes the set of discriminative profiles).

5.4.3 Effect of the set of input PVTs. In this experiment, we consider the synthetic pipelines with 5 attributes (same as Figure 8) and vary the input set of PVTs. We generated a random sample of the set of PVTs as input and added the ground truth cause if it is not present in the sample. We observe that DataPrism is stable (fewer than 10 interventions across different settings) and requires fewer interventions than `BugDoc` and `Anchors`.

6 RELATED WORK

Interventional debugging. AID [26] uses an interventional approach to blame runtime conditions of a program for causing failure; but it is limited to software bugs and does not intervene on datasets. BugDoc [52] finds parameter settings in an opaque-box pipeline as root causes of pipeline failure; but it only reports whether a dataset is a root cause and does not explain why a dataset causes the failure. Prior techniques on data-cleaning [61, 64], fair ML [32, 54, 69, 73], and data profiling [1, 2] evaluate the quality of a dataset for a specific application and propose data transformations for data cleaning or bias removal. These techniques can be modeled using PVTs in DataPrism framework. We demonstrated the flexibility of DataPrism by considering two representative data-cleaning techniques: (1) HoloClean [61] to transform a dataset with respect to functional dependencies, and (2) techniques described in [27] to transform a dataset with respect to conformance constraints. Identifying useful PVTs for a given application is orthogonal to our contribution.

Data explanation. Explanations for query results have been abundantly studied [7, 8, 19, 24, 75]. Some techniques find causes of errors in data generation processes [75], while others discover relationships among attributes [7, 24], and across datasets [19]. Unlike interventional efforts, which DataPrism focuses on, these approaches operate on observational data.

Model explanation. Machine learning interpreters [65, 66] perturb test data to learn a surrogate for models, but their goal is not to find mismatch between data and models. Debugging methods for ML pipelines are similar to data explanation [13, 14], where training data may cause model's underperformance. [74] and [47] discuss principled ways to find reasons for malfunctions. Wu et al. [76] allows users to *complain* about outputs of SQL queries, and presents data points whose removal resolves the complaints. [70] validates when models fail on certain datasets and assumes knowledge of the mechanism that corrupts the data. In contrast, we aim to find discriminative profiles among datasets without such knowledge.

Causal debugging. Data-driven approaches have been taken for causal-inference-based fault localization [5, 6, 18, 35, 68], software testing [28, 29, 33, 39, 43, 45, 78], and statistical debugging [50, 79]. However, they assume transparency of the software or are application-specific. Our work shares similarity with BugEx [67], which generates test cases to isolate root causes. However, it assumes complete knowledge of the program and data-flow paths.

Data debugging. Porting concepts of debugging from software to data has gained attention in data-management community [12, 55]. Dagger [62, 63] provides data-debugging primitives for transparent interactions with data-driven pipelines. CheckCell [9] ranks data cells that unusually affect output of a given target. However, it is not meant for datasets where single cells are unlikely to cause malfunction. DataPrism is general-purpose and application-agnostic, identifying causally verified mismatch between the data and the system.

7 SUMMARY AND FUTURE DIRECTIONS

We introduced the problem of identifying causes and fixes of mismatch between data and systems that operate on data. To this end, we presented DataPrism, a framework that reports violation of data profiles as causally verified root causes of system malfunction and reports fixes in the form of transformation functions. We demonstrated the effectiveness and efficacy of DataPrism in explaining the reason of mismatch in several real-world and synthetic data-driven pipelines, significantly outperforming the state of the art. In future, we plan to extend DataPrism to support multi-objective requirements to generate minimal and interpretable explanations.

Acknowledgements. This work was supported by NSF grants #2030859, IIS-2106888, CCF-1763423, IIS-1943971, IIS-1916505, and OAC-1934464, and the DARPA D3M program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF and DARPA.

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *The VLDB Journal* 24, 4 (2015), 557–581.

[2] Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. 2018. Data profiling. *Synthesis Lectures on Data Management* 10, 4 (2018), 1–154.

[3] AdaBoost Classifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[4] Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual String Embeddings for Sequence Labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*. 1638–1649.

[5] Mona Attariyan, Michael Chow, and Jason Flinn. 2012. X-Ray: Automating Root-Cause Diagnosis of Performance Anomalies in Production Software. In *Proceedings of USENIX OSDI* (Hollywood, CA, USA) *(OSDI'12)*. USENIX Association, USA, 307–320.

[6] Mona Attariyan and Jason Flinn. 2011. Automating Configuration Troubleshooting with ConfAid. *;login:* 36, 1 (2011), 1–14.

[7] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. MacroBase: Prioritizing Attention in Fast Data. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. ACM, New York, NY, USA, 541–556.

[8] Daniel W Barowy, Emery D Berger, and Benjamin Zorn. 2018. ExceLint: Automatically finding spreadsheet formula errors. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 1–26.

[9] Daniel W. Barowy, Dimitar Gochev, and Emery D. Berger. 2014. CheckCell: data debugging for spreadsheets. In *OOPSLA*. 507–523.

[10] Rachel KE Bellamy, Kuntal Dey, Michael Hind, Samuel C Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, et al. 2018. AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. *arXiv preprint arXiv:1810.01943* (2018).

[11] Bias in Amazon Hiring. https://becominghuman.ai/amazons-sexist-ai-recruiting-tool-how-did-it-go-so-wrong-e3d14816d98e

[12] Mike Brachmann, Carlos Bautista, Sonia Castelo, Su Feng, Juliana Freire, Boris Glavic, Oliver Kennedy, Heiko Müeller, Rémi Rampin, William Spoth, et al. 2019. Data debugging and exploration with vizier. In *Proceedings of the 2019 International Conference on Management of Data*. 1877–1880.

[13] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2019. Data validation for machine learning. In *Conference on Systems and Machine Learning (SysML). https://www. sysml. cc/doc/2019/167. pdf*.

[14] Gabriel Cadamuro, Ran Gilad-Bachrach, and Xiaojin Zhu. 2016. Debugging machine learning models. In *ICML Workshop on Reliable Machine Learning in the Wild*.

[15] Cardiovascular Disease dataset. https://www.kaggle.com/sulianova/cardiovascular-disease-dataset

[16] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2016. On the discovery of relaxed functional dependencies. In *Proceedings of the 20th International Database Engineering & Applications Symposium*. 53–61.

[17] Giuseppe Casalicchio, Christoph Molnar, and Bernd Bischl. 2018. Visualizing the feature importance for black box models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 655–670.

[18] Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. 2002. Pinpoint: Problem Determination in Large, Dynamic Internet Services. In *Proceedings of IEEE DSN (DSN'02)*. IEEE, USA, 595–604.

[19] Fernando Chirigati, Harish Doraiswamy, Theodoros Damoulas, and Juliana Freire. 2016. Data Polygamy: The Many-Many Relationships Among Urban Spatio-Temporal Data Sets. In *Proceedings of ACM SIGMOD (SIGMOD '16)*. ACM, New York, NY, USA, 1011–1025.

[20] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *PVLDB* 6, 13 (2013), 1498–1509.

[21] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. 2014. From Data Fusion to Knowledge Fusion. *PVLDB* 7, 10 (2014), 881–892.

[22] Dingzhu Du, Frank K Hwang, and Frank Hwang. 2000. *Combinatorial group testing and its applications*. Vol. 12. World Scientific.

[23] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[24] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and Informative Explanations of Outcomes. *PVLDB* 8, 1 (Sept. 2014), 61–72.

[25] Wenfei Fan, Floris Geerts, Laks V. S. Lakshmanan, and Ming Xiong. 2009. Discovering Conditional Functional Dependencies. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*. 1231–1234.

[26] Anna Fariha, Suman Nath, and Alexandra Meliou. 2020. Causality-Guided Adaptive Interventional Debugging. In *SIGMOD*. 431–446.

[27] Anna Fariha, Ashish Tiwari, Arjun Radhakrishna, Sumit Gulwani, and Alexandra Meliou. 2021. Conformance Constraint Discovery: Measuring Trust in Data-Driven Systems. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 499–512.

[28] Gordon Fraser and Andrea Arcuri. 2013. Whole test suite generation. *IEEE Transactions on Software Engineering* 39, 2 (2013), 276–291.

[29] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*. 498–510.

[30] Sainyam Galhotra, Anna Fariha, Raoni Lourenço, Juliana Freire, Alexandra Meliou, and Divesh Srivastava. 2021. *DataPrism: Exposing Disconnect between Data and Systems*. Technical Report. https://arxiv.org/abs/2105.06058.

[31] Sainyam Galhotra, Udayan Khurana, Oktie Hassanzadeh, Kavitha Srinivas, Horst Samulowitz, and Miao Qi. 2019. Automated Feature Enhancement for Predictive Modeling using External Knowledge. In *2019 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 1094–1097.

[32] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé III, and Kate Crawford. 2018. Datasheets for Datasets. *CoRR* abs/1803.09010 (2018). arXiv:1803.09010

[33] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. 2008. Automated whitebox fuzz testing. In *Proceedings of NDSS*. 151–166.

[34] Google Vision Racism. https://algorithmwatch.org/en/story/google-vision-racism/

[35] Muhammad Ali Gulzar, Siman Wang, and Miryung Kim. 2018. BigSift: Automated Debugging of Big Data Analytics in Data-Intensive Scalable Computing. In *Proceedings of ESEC/FSE* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. ACM, New York, NY, USA, 863–866.

[36] Brent Hailpern and Padmanabhan Santhanam. 2002. Software debugging, testing, and verification. *IBM Systems Journal* 41, 1 (2002), 4–12.

[37] Joseph M Hellerstein. 2008. Quantitative Data Cleaning for Large Databases. (2008).

[38] Thomas A Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. 2003. Software verification with BLAST. In *International SPIN Workshop on Model Checking of Software*. Springer, 235–239.

[39] Christian Holler, Kim Herzig, and Andreas Zeller. 2012. Fuzzing with code fragments. In *Proceedings of USENIX Security Symposium*. 445–458.

[40] IBM AIF 360. https://aif360.mybluemix.net/

[41] Ihab F Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. 647–658.

[42] IMDb Dataset. https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

[43] Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. [n.d.]. CADET: A Systematic Method For Debugging Misconfigurations using Counterfactual Reasoning. ([n. d.]).

[44] Is Amazon same-day delivery service racist? 2016. The Christian Science Monitor. https://www.csmonitor.com/Business/2016/0423/Is-Amazon-same-day-delivery-service-racist

[45] Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2020. Causal Testing: Finding Defects' Root Causes. In *ICSE*.

[46] Nick Koudas, Avishek Saha, Divesh Srivastava, and Suresh Venkatasubramanian. 2009. Metric functional dependencies. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 1275–1278.

[47] Todd Kulesza, Margaret Burnett, Weng-Keen Wong, and Simone Stumpf. 2015. Principles of explanatory debugging to personalize interactive machine learning. In *Proceedings of the 20th international conference on intelligent user interfaces*. 126–137.

[48] Amresh Kumar, M Kiran, and BR Prathap. 2013. Verification and validation of mapreduce program model for parallel k-means algorithm on hadoop cluster. In *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. IEEE, 1–8.

[49] Philipp Langer and Felix Naumann. 2016. Efficient order dependency detection. *VLDB J.* 25, 2 (2016), 223–241.

[50] Ben Liblit, Mayur Naik, Alice X Zheng, Alex Aiken, and Michael I Jordan. 2005. Scalable statistical bug isolation. *Acm Sigplan Notices* 40, 6 (2005), 15–26.

[51] Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. 2019. What bugs cause production cloud incidents?. In *Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS 2019, Bertinoro, Italy, May 13-15, 2019*. 155–162.

[52] Raoni Lourenço, Juliana Freire, and Dennis E. Shasha. 2020. BugDoc: Algorithms to Debug Computational Processes. In *SIGMOD*. 463–478.

[53] Ali Mesbah, Arie Van Deursen, and Danny Roest. 2011. Invariant-based automatic testing of modern web applications. *IEEE Transactions on Software Engineering* 38, 1 (2011), 35–53.

[54] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model Cards for Model Reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency* (Jan 2019).

[55] Kıvanç Muşlu, Yuriy Brun, and Alexandra Meliou. 2013. Data debugging with continuous testing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 631–634.

[56] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional dependency discovery: An experimental evaluation of seven algorithms. *PLDB* 8, 10 (2015), 1082–1093.

[57] Thorsten Papenbrock, Sebastian Kruse, Jorge-Arnulfo Quiané-Ruiz, and Felix Naumann. 2015. Divide & conquer-based inclusion dependency discovery. *PLDB* 8, 7 (2015), 774–785.

[58] Python Rexpy package. https://tdda.readthedocs.io/en/v1.0.30/rexpy.html

[59] Random Forest Classifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[60] Kaivalya Rawal, Ece Kamar, and Himabindu Lakkaraju. 2020. Can I Still Trust You?: Understanding the Impact of Distribution Shifts on Algorithmic Recourses. *arXiv preprint arXiv:2012.11788* (2020).

[61] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. HoloClean: holistic data repairs with probabilistic inference. *PVLDB* 10, 11 (2017), 1190–1201.

[62] El Kindi Rezig, Ashrita Brahmaroutu, Nesime Tatbul, Mourad Ouzzani, Nan Tang, Timothy G. Mattson, Samuel Madden, and Michael Stonebraker. 2020. Debugging Large-Scale Data Science Pipelines using Dagger. *PVLDB* 13, 12 (2020), 2993–2996.

[63] El Kindi Rezig, Lei Cao, Giovanni Simonini, Maxime Schoemans, Samuel Madden, Nan Tang, Mourad Ouzzani, and Michael Stonebraker. 2020. Dagger: A Data (not code) Debugger. In *CIDR 2020, 10th Conference on Innovative Data Systems Research, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings.*

[64] El Kindi Rezig, Mourad Ouzzani, Walid G Aref, Ahmed K Elmagarmid, Ahmed R Mahmood, and Michael Stonebraker. 2021. Horizon: scalable dependency-driven data cleaning. *PVLDB* 14, 11 (2021), 2546–2554.

[65] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining.* 1135–1144.

[66] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-Precision Model-Agnostic Explanations.. In *AAAI*, Vol. 18. 1527–1535.

[67] Jeremias Rößler, Gordon Fraser, Andreas Zeller, and Alessandro Orso. 2012. Isolating failure causes through test case generation. In *International Symposium on Software Testing and Analysis, ISSTA 2012, Minneapolis, MN, USA, July 15-20, 2012*, Mats Per Erik Heimdahl and Zhendong Su (Eds.). ACM, 309–319.

[68] Babak Salimi, Harsh Parikh, Moe Kayali, Lise Getoor, Sudeepa Roy, and Dan Suciu. 2020. Causal Relational Learning. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020.* 241–256.

[69] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. 2019. Interventional Fairness: Causal Database Repair for Algorithmic Fairness. In *SIGMOD.* 793–810.

[70] Sebastian Schelter, Tammo Rukat, and Felix Bießmann. 2020. Learning to Validate the Predictions of Black Box Classifiers on Unseen Data. In *SIGMOD.* 1289–1299.

[71] Sentiment 140 dataset. https://www.kaggle.com/kazanova/sentiment140

[72] Shaoxu Song and Lei Chen. 2011. Differential dependencies: Reasoning and discovery. *ACM Transactions on Database Systems (TODS)* 36, 3 (2011), 1–41.

[73] Julia Stoyanovich and Bill Howe. 2019. Nutritional Labels for Data and Models. *IEEE Data Eng. Bull.* 42, 3 (2019), 13–23.

[74] Paroma Varma, Dan Iter, Christopher De Sa, and Christopher Ré. 2017. Flipper: A systematic approach to debugging training sets. In *Proceedings of the 2nd Workshop on Human-in-the-Loop Data Analytics.* 1–5.

[75] Xiaolan Wang, Xin Luna Dong, and Alexandra Meliou. 2015. Data X-Ray: A Diagnostic Tool for Data Errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* 1231–1245.

[76] Weiyuan Wu, Lampros Flokas, Eugene Wu, and Jiannan Wang. 2020. Complaint-driven Training Data Debugging for Query 2.0. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data.* 1317–1334.

[77] Jing Nathan Yan, Oliver Schulte, Mohan Zhang, Jiannan Wang, and Reynold Cheng. 2020. SCODED: Statistical Constraint Oriented Data Error Detection. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020.* 845–860.

[78] Andreas Zeller. 1999. Yesterday, My Program Worked. Today, It Does Not. Why?. In *Software Engineering - ESEC/FSE'99, 7th European Software Engineering Conference, Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Toulouse, France, September 1999, Proceedings.* 253–267.

[79] Alice X. Zheng, Michael I. Jordan, Ben Liblit, Mayur Naik, and Alex Aiken. 2006. Statistical Debugging: Simultaneous Identification of Multiple Bugs. In *Proceedings of ICML* (Pittsburgh, Pennsylvania, USA) *(ICML'06).* ACM, New York, NY, USA, 1105–1112.