Data Invariants: On Trust in Data-Driven Systems

Anna Fariha[1*] Ashish Tiwari[2*] Arjun Radhakrishna[2] Sumit Gulwani[2] Alexandra Meliou[1]
[1]University of Massachusetts
Amherst, MA, USA
{afariha, ameli}@cs.umass.edu
[2]Microsoft
Bellevue, WA, USA
{astiwar, arradha, sumitg}@microsoft.com

ABSTRACT

The reliability and proper function of data-driven applications hinge on the data's continued conformance to the applications' initial design. When data deviates from this initial profile, system behavior becomes unpredictable. Data profiling techniques such as functional dependencies and denial constraints encode patterns in the data that can be used to detect deviations. But traditional methods typically focus on exact constraints and categorical attributes, and are ill-suited for tasks such as determining whether the prediction of a machine learning system can be trusted or for quantifying data drift. In this paper, we introduce *data invariants*, a new data-profiling primitive that models arithmetic relationships involving multiple numerical attributes within a (noisy) dataset and which complements the existing data-profiling techniques. We propose a *quantitative semantics* to measure the degree of violation of a data invariant, and establish that strong data invariants can be constructed from observations with low variance on the given dataset. A concrete instance of this principle gives the surprising result that *low-variance components* of a principal component analysis (PCA), which are usually discarded, generate better invariants than the high-variance components. We demonstrate the value of data invariants on two applications: trusted machine learning and data drift. We empirically show that data invariants can (1) reliably detect tuples on which the prediction of a machine-learned model should not be trusted, and (2) quantify data drift more accurately than the state-of-the-art methods. Additionally, we show four case studies where an intervention-centric explanation tool uses data invariants to explain causes for tuple non-conformance.

1. INTRODUCTION

Data is central to modern applications in a wide range of domains, including healthcare, transportation, finance, and many others. Such data-driven applications typically rely on learning components trained over large datasets, or components specifically designed to target particular data and workloads. While these data-driven approaches have seen wide adoption and success, their reliability and proper function hinges on the data's continued conformance to the applications' initial design. When data deviates from these initial parameters, application performance degrades and system behavior becomes unpredictable. In particular, as data-driven prediction models are employed in applications with great human and societal impact, we need to be able to assess the trustworthiness of their predictions.

The ability to understand and quantify the deviations of data from the ranges and distributions that an application can safely support is critical in determining whether the prediction of a machine-learning system can be trusted [83, 86, 72, 69, 46], when a system needs to be retrained because of data drift [68, 53, 31, 8], and when a database needs to be retuned [50]. In this paper, we characterize and quantify such deviations with a new data-profiling primitive, a *data invariant*, that captures an implicit constraint over multiple numerical attributes that tuples in a reference dataset satisfy. We proceed to describe a real-world example of data invariants, drawn from our case-study evaluation on the problem of *trusted machine learning* (TML), which aims to quantify trust in the prediction of a machine-learned model over a new input tuple.

...	dep_date	dep_time	arr_time	duration (minute)
...	May 2	14:30	18:20	230
...	July 22	09:05	12:15	195
...	June 6	10:20	20:00	582
...	May 19	11:10	13:05	117
...	April 7	22:30	06:10	458

Figure 1: Sample of the airlines dataset (details are in Section 6.1), showing departure, arrival, and duration information. All times are in the same time zone. The dataset does not report arrival date, but an arrival time earlier than departure time (as in the last row), indicate an overnight flight. The dataset has some noise in the reported times and duration.

Example 1. We used a dataset with flight information that includes data on departure and arrival times, flight duration, etc. (Figure 1) to train a linear regression model to predict flight delays. The model was trained on a subset of the data that happened to include only daytime flights (such as the first 4 tuples in Figure 1). In an empirical evaluation of the regression accuracy, we found that the mean absolute error of the regression output more than quadruples for overnight flights (such as the last tuple in Figure 1), compared to daytime flights. The reason is that data for overnight flights deviates from the *profile* of the training data. Specifically, daytime flights satisfy the *invariant* that "arrival time is later than departure time and their difference is very close to the flight duration", which does not hold for overnight flights. Critically, even though this invariant is unaware of the target attribute (delay), it was still a good proxy of the regressor's performance.

In this paper, we propose *data invariants*, a new data-profiling primitive that complements the existing literature on modeling data constraints. Specifically, data invariants capture *arithmetic relationships* over multiple numerical attributes in a possibly noisy dataset. For example, the data invariant of Example 1 corresponds to the constraint: $\epsilon_1 \leq (arr\_time - dep\_time) - duration \leq \epsilon_2$, where ϵ_1 and ϵ_2 are small values. Data invariants can capture complex linear dependencies across attributes. For example, if the flight departure and arrival data reported the hours and the minutes across separate attributes, the invariant would be $\epsilon_1 \leq (60 \cdot arr\_hour + arr\_min) - (60 \cdot dep\_hour + dep\_min) - duration \leq \epsilon_2$. Existing

*Both authors contributed equally to this paper.

constraint models, such as functional dependencies and denial constraints, do not capture such arithmetic dependencies, and are typically sensitive to noise and not well-suited for numerical attributes.

A key insight in our work is that learning systems implicitly rely on data invariants (e.g., by reducing the weight of an attribute that can be deduced by others); thus, we can use a tuple's deviation from these invariants as a proxy for the trust on the system's prediction for that tuple. We focus on *quantitative semantics* of data invariants, so that we not only capture the (Boolean) violation of data invariants by a new tuple, but we can also measure the degree of violation. Through this mechanism, data invariants can quantify trust in prediction outcomes, detect data drift, and specify when a database should be retuned.

We first proceed to discuss where data invariants fit with respect to the existing literature on data profiling: specifically, functional dependencies and denial constraints. Then, we provide the core intuition and insights for modeling and deriving data invariants.

Prior art on modeling arithmetic constraints

Data invariants, just like other constraint models, fall under the umbrella of data profiling, which refers to the task of extracting technical metadata about a given dataset [4]. A key task in data profiling is to learn relationships among attributes. Denial constraints (DC) [16, 11, 65] encapsulate a number of different data-profiling primitives such as functional dependencies (FD). However, most DC discovery techniques are restricted to hard constraints—(1) *all* tuples must satisfy the constraints, and (2) the constraints should be *exactly* satisfied—and are not suitable when the data is noisy.

DCs can adjust to noisy data by adding predicates until the constraint becomes exact over the entire dataset, but this can make the constraint extremely large, complex, and uninterpretable. Moreover, such a constraint might not even provide the desired generalization. For example, a finite DC—whose language is limited to universally-quantified first-order logic—cannot model the data invariant of Example 1, which involves an arithmetic expression (addition and multiplication with a constant). Expressing data invariants require a richer language that includes linear arithmetic expressions. Pattern functional dependencies [67] move towards addressing this limitation of DCs using regular expressions, but they only focus on non-numerical attributes. While techniques for approximate DC discovery [65, 41, 52] exist, they rely on the users to provide an acceptable error threshold. In contrast, data invariants do not require any error threshold.

Using a FD like $\{arr\_hour, arr\_min, dep\_hour, dep\_min\} \rightarrow duration$ to model the invariant of Example 1 suffers from several limitations. First, since the data is noisy, in the traditional setting, no FD would be learned. Metric FDs [51] allow small variations in the data (similar attribute values are considered identical); however, to model this invariant using a metric FD, one needs to specify non-trivial similarity metrics involving multiple attributes. In our example, such a metric would need to encode that (1) the attribute combination $\langle$hour = 4, min = 59$\rangle$ is very similar to $\langle$hour = 5, min = 1$\rangle$ and (2) the attribute combination $\langle$hour = 4, min = 1$\rangle$ is not similar to $\langle$hour = 5, min = 59$\rangle$. Moreover, existing work on metric FDs only focuses on their verification [51]; to the best of our knowledge, no prior work exists on the discovery of metric FDs.

Challenges

Our work targets the following challenges.

Numerical attributes. Numerical attributes are inherently noisy and their domain can be infinite. Most existing approaches are restricted to categorical or discretized attributes and typically do not handle or, even when they handle, perform poorly with numerical attributes. Data invariants focus on numerical attributes specifically, and can model complex linear dependencies among them.

Impact of violation. In applications of constraint violations, some violations may be less critical than others. Our work considers a notion of invariant importance, and weighs violations against invariants accordingly. Intuitively, violating a stricter invariant (an invariant specifying low variability in the underlying data) is likely more critical than violating a looser one.

Measuring violation. Existing approaches typically require that constraints are *exactly* satisfied by *all* tuples. As a result, they discard inexact constraints that may still be useful and treat violation as Boolean (a tuple satisfies the constraint or not). Data invariants offer flexibility in the derived constraints, and specifically allow for measuring the degree of violation. This measure weighs the violation of each invariant based on a measure of its impact.

Scalability. Unlike data invariants, most existing approaches focus on precise and exact constraints. When the size of the data grows, especially in the presence of noise, these approaches extract less useful information as exact constraints become more rare. Thus, scaling to larger data sizes can render these methods less effective.

Efficiency. The complexity of existing techniques grows exponentially with the number of attributes, making dependency discovery impractical for even a moderately large number of attributes (~ 50). In contrast, the complexity of data-invariants computation is only cubic in the number of attributes.

In summary, we envision data invariants—which encode constraints that express approximate arithmetic relationships among numerical attributes—as an essential primitive that will enrich the data-profiling literature and is complementary to existing techniques in denial constraints.

Key insights of data invariants

Data invariants fall under the umbrella of the general task of data profiling. They specify constraints over numerical attributes, complementing existing work on data constraints. In this paper in particular, we focus on data invariants specifying linear arithmetic constraints. Data invariants focus on finding a closed-form (and thus explainable) function over the numerical attributes, such that the function, when evaluated on the tuples, results in *low variance*.

We present a method for learning linear invariants inspired by principal component analysis (PCA). Our key observation is that the principal components with low variance (on the dataset) yield strong data invariants. Note that this is different from—and in fact completely opposite to—the traditional approaches that perform multidimensional analysis after reducing dimensionality using PCA [68]. Beyond simple linear invariants—such as the one in Example 1—we also derive *disjunctive linear invariants*, which are disjunctions of linear invariants. We achieve this derivation by dividing the dataset into disjoint partitions, and learning simple linear invariants for each partition.

In this paper, we introduce a simple language for data invariants. Furthermore, given an invariant and a tuple, we derive a numerical score that measures how much the tuple violates the invariant: a score of zero indicates no violation and a positive score indicates that the tuple violates the invariant, with higher score indicating greater violation. We also provide a mechanism to aggregate the violation of a set of data invariants, by weighing violation of stricter (i.e., low-variance) invariants more and looser (i.e., high-variance) invariants less. Our experimental evaluation (Section 6) demonstrates that this violation score is an effective measure of confidence in the prediction of learned models and effectively captures data drift.

Contributions. Our work makes the following contributions:

- We ground our motivation and our work with two case studies on trusted machine learning (TML) and data drift. (Section 2)
- We introduce and formalize *data invariants*, describe a language for expressing them, and discuss their semantics. (Section 3)
- We formally establish that strong data invariants are constructed from derived attributes with small variance and small mutual correlation on the given dataset. We provide an efficient, scalable, and highly parallelizable PCA-based algorithm for computing simple linear invariants and disjunctions over them. We also analyze their time and memory complexity. (Section 4)
- We formalize the notion of *non-conforming tuples* in the context of trusted machine learning and provide a mechanism to detect whether a tuple is non-conforming using data invariants. To that end, we focus on *weak* invariants, whose violation is sufficient, but not necessary, to detect non-conformance. (Section 5)
- We empirically analyze the effectiveness data invariants in our two case-study applications—TML and data-drift quantification. We find that data invariants can reliably predict the trustworthiness of linear models and quantify data drift precisely, outperforming the state of the arts. We also show how an intervention-centric explanation tool—built on top of data invariants—can explain causes for tuple non-conformance (by assigning responsibility to the attributes) on real-world datasets. (Section 6)

2. CASE STUDIES OF DATA INVARIANTS

Like other data-profiling mechanisms, data invariants have general applicability in understanding and describing datasets. Within the scope of our work, we focus in particular on the utility of data invariants in detecting when applications may operate outside their *safety envelope* [83], i.e., when the operation of a system may become untrustworthy or unreliable. We describe two case studies that motivate our work. We later provide an extensive evaluation over these two applications in Section 6.

Trusted Machine Learning. Trusted machine learning (TML) refers to the problem of quantifying trust in the prediction made by a machine-learned model on a new input tuple. This is particularly useful in case of *extreme verification latency* [77], where ground-truth outputs for new test tuples are not immediately available to evaluate the performance of a learned model. If a model is trained using dataset D, then data invariants for D specify a safety envelope [83] that characterizes the tuples for which the learned model is expected to make trustworthy predictions. If the new tuple falls outside the safety envelope, i.e., it violates the invariants, then the learned model is likely to produce an untrustworthy prediction. Intuitively, the higher the violation, the lower the trust. While some machine-learning approaches return a confidence measure along with the prediction, these confidence measures are not well-calibrated, and it is well-known that there are several issues with interpreting these confidence measures as a measure of trust in the prediction [46, 33].

In data-driven systems, *feature-drift* [8] is one of the reasons for observing non-conforming tuples. In the context of trusted machine learning, we formalize the notion of *non-conforming tuples*. A key result that we present is that data invariants provide a sound and complete procedure for detecting whether a given tuple is non-conforming. Since our proof of this result is non-constructive, we present a second result that establishes sufficient, but not necessary, conditions for solving it (Section 5). This result indicates that the search for invariants should be guided by the class of models considered by the corresponding machine learning technique.

Data drift. Our second use-case focuses on detecting and quantifying *data drift* [68, 53, 31, 8]. Data drift specifies a significant change in the underlying data distribution that typically requires that systems be updated and models retrained. Our data invariant-based approach here is simple: given two datasets D_1 and D_2, we first compute data invariants for D_1, and then evaluate the invariants on D_2. If D_2 satisfies the invariants, then we have no evidence of drift. In contrast, if D_2 violates the invariants, then that serves as an indication of drift; and the degree of violation measures how much D_2 has drifted from D_1.

While we focus on these two applications in this paper, we point out to few other applications of data invariants in Section 7.

3. DATA INVARIANTS

In this section, we define data invariants, a new data-profiling primitive that allows us to capture complex dependencies across numerical attributes. Intuitively, a data invariant is an implicit constraint that the data satisfies. We first provide the general definition of data invariants, and then propose a language for representing them. We then define quantitative semantics over data invariants, which allows us to quantify their violation.

Basic notations. We use $\mathcal{R}(A_1, A_2, \ldots, A_m)$ to denote a relation schema where A_i denotes the i^{th} attribute of $\mathcal{R}$. 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. We use $t \in \mathbf{Dom}^m$ to denote a tuple in the schema $\mathcal{R}$. A dataset $D \subseteq \mathbf{Dom}^m$ is a specific instance of the schema $\mathcal{R}$. For ease of notation, we assume some order of tuples in D and we use $t_i \in D$ to refer to the i^{th} tuple and $t_i.A_j \in \texttt{Dom}_j$ to denote the value of the j^{th} attribute of t_i.

We start with a *strict* definition of data invariants, and then explain how it generalizes to account for noise in the data.

Definition 1 (Data invariant (strict))**.** A data invariant for a dataset $D \subseteq \mathbf{Dom}^m$ is another dataset $\texttt{Inv} \subseteq \mathbf{Dom}^m$ s.t. $D \subseteq \texttt{Inv}$.

Intuitively, a data invariant specifies a set of allowable tuples ($\texttt{Inv}$). The tightest invariant for D is D itself, whereas the loosest invariant is $\mathbf{Dom}^m$. We represent an invariant $\texttt{Inv}$ using its characteristic function (or formula) $\phi : \mathbf{Dom}^m \mapsto \{\texttt{True}, \texttt{False}\}$. By definition, $t \in \texttt{Inv}$ if and only if $\phi(t) = \texttt{True}$. A characteristic function ϕ is an invariant for D if $\forall t \in D\ \phi(t) = \texttt{True}$. We write $\phi(t)$ and $\neg\phi(t)$ to denote that t satisfies and does not satisfy the invariant ϕ, respectively. In this paper, we do not distinguish between the invariant $\texttt{Inv}$ and its representation ϕ, and use them interchangeably.

In practice, because of noise, some tuples in D may not satisfy an invariant, i.e., $\exists t \in D$ s.t. $t \notin \texttt{Inv}$. To account for noise, we relax the definition of invariants as follows.

Definition 2 (Data invariant (relaxed))**.** A relaxed data invariant for a dataset $D \subseteq \mathbf{Dom}^m$ is another dataset $\texttt{Inv} \subseteq \mathbf{Dom}^m$ s.t. $|D - \texttt{Inv}| \ll |D|$.

The set $(D - \texttt{Inv})$ denotes atypical points in D that do not satisfy the invariant (and thus are not in $\texttt{Inv}$). In our work, we do not need to know the set $(D - \texttt{Inv})$, nor do we need to purge the atypical points from the dataset. Our techniques derive invariants in ways that ensure a small $(D - \texttt{Inv})$ (Section 4). In this paper, when we talk about data invariants, we refer to relaxed data invariants, unless otherwise specified.

3.1 A Language for Data Invariants

Projection. A central concept in our language for data invariants is that of a *projection*. A projection is a function $F : \mathbf{Dom}^m \mapsto \mathbb{R}$

that maps a tuple $t \in \mathbf{Dom}^m$ to a real number $F(t)$. We extend a projection F to a dataset D by defining $F(D)$ to be the sequence of reals obtained by applying F on each tuple in D.

Our language for data invariants consists of formulas Φ generated by the grammar:

$$\begin{aligned} \phi &:= \texttt{lb} \leq F(\vec{A}) \leq \texttt{ub} \mid \wedge(\phi, \ldots, \phi) \\ \psi_A &:= \vee(\ (A = c_1) \triangleright \phi,\ (A = c_2) \triangleright \phi,\ \ldots) \\ \Psi &:= \psi_A \mid \wedge(\psi_{A_1}, \psi_{A_2}, \ldots) \\ \Phi &:= \phi \mid \Psi \end{aligned}$$

The language consists of (1) bounded-projection constraints $\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}$ where F is a projection on $\mathbf{Dom}^m$, $\vec{A}$ is the tuple of formal parameters $(A_1, A_2, \ldots, A_m)$, and $\texttt{lb}, \texttt{ub} \in \mathbb{R}$ are reals; (2) equality constraints $A = c$ where A is an attribute and c is a constant in the A's domain; and (3) operators ($\triangleright$, $\wedge$, and $\vee$,) that connect the constraints. Intuitively, $\triangleright$ is a switch operator that specifies which invariant ϕ applies based on the value of the attribute A, $\wedge$ denotes conjunction, and $\vee$ denotes disjunction.

Invariant formulas generated by ϕ are called *simple invariants* and those generated by Ψ are called *compound invariants*. Note that a formula generated by ψ_A only allows equality constraints on a single attribute, namely A, among all the disjuncts.

3.2 Quantitative Semantics of Data Invariants

Data invariants have a natural Boolean semantics: a tuple either satisfies an invariant or it does not. However, Boolean semantics is of limited use in practical applications, because it does not quantify the degree of invariant violation. We interpret data invariants using a quantitative semantics, which quantifies violations. Quantitative semantics has the additional benefit that it reacts to noise more gracefully than Boolean semantics.

Given a formula Φ, the quantitative semantics $[\![\Phi]\!](t)$ is a measure of the violation of Φ on a tuple t—with a value of 0 indicating no violation and a value greater than 0 indicating violation. If $\Phi(t)$ is `True` (in Boolean semantics), then $[\![\Phi]\!](t)$ will be 0. Formally, $[\![\Phi]\!]$ is a mapping from $\mathbf{Dom}^m$ to $[0, 1]$

Quantitative semantics of simple invariants. The quantitative semantics of simple invariants is defined as:

$$[\![\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}]\!](t) := \eta(\alpha \cdot \max(0, F(t) - \texttt{ub}, \texttt{lb} - F(t)))$$
$$[\![\wedge(\phi_1, \ldots, \phi_K)]\!](t) := \textstyle\sum_k^K \gamma_k \cdot [\![\phi_k]\!](t)$$

The quantitative semantics uses the following parameters:

Scaling factor $\alpha \in \mathbb{R}^+$.
Projections are unconstrained functions and different projections can map the same tuple to vastly different values. We use a scaling factor α to standardize the values computed by a projection F, and to bring the values of different projections to the same comparable scale. The scaling factor is automatically computed as the inverse of the standard deviation: $\frac{1}{\sigma(F(D))}$. (We set α to a large positive number when $\sigma(F(D))$ = 0.)

Normalization function $\eta(.) : \mathbb{R} \mapsto [0, 1]$.
The normalization function maps values in the range $[0, \infty)$ to the range $[0, 1)$. While any monotone mapping from $\mathbb{R}^{\geq 0}$ to $[0, 1)$ can be used, we pick $\eta(z) = 1 - e^{-z}$.

Importance factors $\gamma_k \in \mathbb{R}^+$, $\sum_k^K \gamma_k$=1.
The weights γ_k control the contribution of each bounded-projection invariant in a conjunctive formula. This allows for prioritizing invariants that may be more significant than others within the context of a particular application. In our work, we derive the importance factor of an invariant automatically, based on its projection's standard deviation over D.

Quantitative semantics of compound invariants. Compound invariants are first simplified into simple invariants, and they get their meaning from the simplified form. We define a function $\texttt{simp}(\psi, t)$ that takes a compound invariant ψ and a tuple t and returns a simple invariant. It is defined recursively as follows:

$$\texttt{simp}(\vee(\ (A = c_1) \triangleright \phi_1, (A = c_2) \triangleright \phi_2, \ldots), t) := \phi_k \text{ if } t.A = c_k$$
$$\texttt{simp}(\wedge(\psi_{A_1}, \psi_{A_2}, \ldots), t) := \wedge(\texttt{simp}(\psi_{A_1}, t), \texttt{simp}(\psi_{A_2}, t), \ldots)$$

If the condition in the definition above does not hold for any c_k, then $\texttt{simp}(\psi, t)$ is undefined and $\texttt{simp}(\wedge(\ldots, \psi, \ldots), t)$ is also undefined. If $\texttt{simp}(\psi, t)$ is undefined, then $[\![\psi]\!](t) := 1$. When $\texttt{simp}(\psi, t)$ is defined, the quantitative semantics of ψ is given by:

$$[\![\psi]\!](t) := [\![\texttt{simp}(\psi, t)]\!](t)$$

Since compound invariants simplify to simple invariants, we mostly focus on simple invariants. Even there, we pay special attention to bounded-projection invariants (ϕ) of the form $\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}$, which lie at the core of simple invariants.

4. SYNTHESIZING DATA INVARIANTS

In this section, we describe our techniques for deriving data invariants. We first focus on the synthesis of simple invariants (the ϕ invariants in our language specification), followed by compound invariants (the Ψ invariants in our language specification). Finally, we analyze the time and memory complexity of our algorithms.

4.1 Simple Invariants

We now describe how we discover simple invariants for a given dataset. We start by discussing how we synthesize bounds for a given projection. We then describe a principle for identifying effective projections. We establish that: *a strong data invariant for a dataset is made from projections that (1) do not have large correlations among each other and (2) have small standard deviations on that dataset.* Finally, we provide a constructive procedure—based on principal component analysis—to pick the appropriate projections to use in a simple invariant, along with their importance factors. By putting all these pieces together, we get a procedure for synthesizing simple invariants.

4.1.1 Synthesizing Bounds for Projections

Fix a projection F and consider the bounded-projection invariant ϕ of the form $\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}$. Given a dataset D, a trivial way to compute bounds is: $\texttt{lb} = \min(F(D))$ and $\texttt{ub} = \max(F(D))$. However, this choice is very sensitive to noise: adding a single "atypical" tuple to D can produce very different invariants. Hence, we instead use the following more robust choices:

$$\texttt{lb} = \mu(F(D)) - C \cdot \sigma(F(D))$$
$$\texttt{ub} = \mu(F(D)) + C \cdot \sigma(F(D))$$

Here, $\mu(F(D))$ and $\sigma(F(D))$ denote the mean and standard deviation of the values in $F(D)$, respectively, and C is some positive constant. With these bounds, $[\![\phi]\!](t) = 0$ implies that $F(t)$ is within $C \times \sigma(F(D))$ from the mean $\mu(F(D))$. In our experiments, we set $C = 4$, which ensures that $|D - \texttt{Inv}| \ll |D|$ for many distributions of the values in $F(D)$. Specifically, if $F(D)$ follows a normal distribution, 99.99% of the population is expected to lie within 4 standard deviations from mean.

Setting the bounds `lb` and `ub` as $C \cdot \sigma(F(D))$-away from the mean, and the scaling factor $\alpha = \frac{1}{\sigma(F(D))}$, guarantees the following property for our quantitative semantics:

Lemma 1. Let D be a dataset and let ϕ_k be $\texttt{lb}_k \leq F_k(\vec{A}) \leq \texttt{ub}_k$ for $k = 1, 2$. Then, for any tuple t, if $\frac{|F_1(t)-\mu(F_1(D))|}{\sigma(F_1(D))} \geq \frac{|F_2(t)-\mu(F_2(D))|}{\sigma(F_2(D))}$, then $[\![\phi_1]\!](t) \geq [\![\phi_2]\!](t)$.

Plainly, this means that larger deviation from an invariant (proportionally to the standard deviation) results in higher degree of violation under our semantics.

The proof follows from the fact that the normalization function $\eta(.)$ is monotonically increasing, and hence, $[\![\phi_k]\!](t)$ is a monotonically non-decreasing function of $\frac{|F_k(t)-\mu(F_k(D))|}{\sigma(F_k(D))}$.

4.1.2 Principle for Synthesizing Projections

To understand how to derive the right projections, we need to first understand what makes an invariant more effective than others in a particular task. Primarily, an effective invariant: (1) should not overfit the data, but rather generalize by capturing the properties of the data, and (2) should not underfit the data, because it would be too permissive and fail to identify deviations effectively. Our flexible bounds (Section 4.1.1) serve to avoid overfitting. In this section, we focus on identifying the principles that help us avoid underfitting.

An effective invariant should help identify deviating tuples. To analyze what makes an invariant more effective than another, we formalize two terms: (1) the strength of invariants as it corresponds to the degree of violation, and (2) *incongruous* tuples, which are those tuples that deviate from the relative trend of two invariants.

Stronger. An invariant ϕ_1 is *stronger* than another invariant ϕ_2 on a subset $H \subseteq \mathbf{Dom}^m$ if $\forall t \in H,\ [\![\phi_1]\!](t) \geq [\![\phi_2]\!](t)$.

Incongruous. For a dataset $D \subseteq \mathbf{Dom}^m$ and a projection F, let $\Delta F(t) = F(t) - \mu(F(D))$. For projections F_1 and F_2, the correlation coefficient ρ_{F_1,F_2} is defined as $\frac{\frac{1}{|D|}\sum_{t\in D} \Delta F_1(t)\Delta F_2(t)}{\sigma(F_1(D))\sigma(F_2(D))}$. Informally, an incongruous tuple for F_1 and F_2 is one that does not follow the general trend of correlation between F_1 and F_2; e.g., if F_1 and F_2 are positively correlated (i.e., $\rho_{F_1,F_2} > 0$), an incongruous tuple deviates in opposite ways from the mean of each projection, i.e., $\Delta F_1(t) \cdot \Delta F_2(t) < 0$. More formally, a tuple t is *incongruous* w.r.t. a projection pair $\langle F_1, F_2\rangle$ on D if:

$$\rho_{F_1,F_2} \cdot \Delta F_1(t) \cdot \Delta F_2(t) < 0$$

Example 2. Let $D = \{(1,1),(2,2),(3,3)\}$ be a dataset with two attributes A_1 and A_2. The projections $F_1(A_1, A_2) = A_1$ and $F_2(A_1, A_2) = A_2$ are positively correlated ($\rho_{F_1,F_2} > 0$); hence, the tuples $(1, 3)$ and $(3, 1)$ are both incongruous, whereas $(3, 4)$ and $(1, 0)$ are not incongruous w.r.t. $\langle F_1, F_2\rangle$.

We proceed to show that when two projections are highly correlated, their linear combination leads to a projection with lower standard deviation and a stronger invariant. We will then generalize this result to multiple projections in Theorem 3. This provides the key insight of this analysis, which is that projections with low standard deviation define stronger invariants (and are thus preferable), and that an invariant with multiple highly-correlated projections is suboptimal (as highly-correlated projections can be linearly combined into one with lower standard deviation). We write ϕ_F to denote $\texttt{lb} \leq F(\vec{A}) \leq \texttt{ub}$, the invariant synthesized from F. (Proofs are in the Appendix.)

Lemma 2. Let D be a dataset and F_1, F_2 be two projections on D s.t. $|\rho_{F_1,F_2}| \geq \frac{1}{2}$. Then, $\exists \beta_1, \beta_2 \in \mathbb{R}$ s.t. $\beta_1^2 + \beta_2^2 = 1$ and for the new projection $F = \beta_1 F_1 + \beta_2 F_2$:
(1) $\sigma(F(D)) < \sigma(F_1(D))$ and $\sigma(F(D)) < \sigma(F_2(D))$, and
(2) ϕ_F is stronger than both ϕ_{F_1} and ϕ_{F_2} on the set of tuples that are incongruous w.r.t. $\langle F_1, F_2\rangle$.

We can now use this lemma in an inductive argument to generalize the result to multiple projections.

Theorem 3 (Low Standard Deviation Invariants)**.** Given a dataset D, let $\mathcal{F} = \{F_1, \ldots, F_K\}$ denote a set of projections on D s.t. $\exists F_i, F_j \in \mathcal{F}$ with $|\rho_{F_i,F_j}| \geq \frac{1}{2}$. Then, there exist a nonempty subset $I \subseteq \{1, \ldots, K\}$ and a projection $F = \sum_{k\in I} \beta_k F_k$, where $\beta_k \in \mathbb{R}$ s.t.
(1) $\forall k \in I, \sigma(F(D)) < \sigma(F_k(D))$,
(2) $\forall k \in I$, ϕ_F is stronger than ϕ_{F_k} on the subset H, where $H = \{t \mid \forall k \in I(\beta_k \Delta F_k(t) \geq 0) \vee \forall k \in I(\beta_k \Delta F_k(t) \leq 0)\}$, and
(3) $\forall k \notin I, |\rho_{F,F_k}| < \frac{1}{2}$.

The theorem establishes that to detect violations for certain tuples (those in H) (1) projections with low standard deviation are better and (2) an invariant with multiple highly-correlated projections may be suboptimal. Note that H is a conservative estimate for the set of tuples where ϕ_F is stronger than each ϕ_{F_k}; there exist tuples outside of H for which ϕ_F is stronger.

Example 3. Consider $D = \{(1,1),(2,2),(3,3)\}$ and projections $F_1(A_1, A_2) = A_1$ and $F_2(A_1, A_2) = A_2$. On D, both projections have the same mean $\mu(F_1(D)) = \mu(F_2(D)) = 2$ and standard deviation $\sigma(F_1(D)) = \sigma(F_2(D)) = \sqrt{2/3}$. The correlation coefficient is $\rho_{F_1,F_2} = 1$ since $F_1 = F_2$ on D. We derive a new projection $F = F_1 - F_2$, and note that $F(D) = \{0, 0, 0\}$ and hence, $\sigma(F(D)) = 0 < \sqrt{2/3}$. Furthermore, ϕ_F is stronger than ϕ_{F_1} and ϕ_{F_2} on all tuples t s.t. $\Delta F_1(t)\Delta F_2(t) < 0$, i.e., $H = \{t \mid (t.A_1 > 2 \wedge t.A_2 < 2) \vee (t.A_1 < 2 \wedge t.A_2 > 2)\}$.

Note that there can be tuples outside of H for which ϕ_F is not stronger than ϕ_{F_1} and ϕ_{F_2}. For example, for the tuple $t = (10, 10)$, $F(t) = 0$ and $\Delta F(t) = 0$. Hence, the invariant $-4 \cdot \sigma(F(D)) \leq F(t) \leq 4 \cdot \sigma(F(D))$ is satisfied (violation score is 0). However, the invariants $-4 \cdot \sigma(F_1(D)) \leq F_1(t) \leq 4 \cdot \sigma(F_1(D))$ and $-4 \cdot \sigma(F_2(D)) \leq F_2(t) \leq 4 \cdot \sigma(F_2(D))$ are not satisfied (violation score > 0). The intuition is that t falls outside the observed trends for F_1 and F_2 (A_1 and A_2), but it is still within the combined trend ($A_1 = A_2$), which better generalized the observed data in D.

4.1.3 PCA-inspired Projection Derivation

Theorem 3 sets the requirements for good projections: prefer projections with small standard deviation because they are more sensitive to change [85, 59, 53], and avoid highly correlated projections. We now present Algorithm 1, inspired by principal component analysis (PCA), for generating linear projections over a dataset D that meet these requirements:

Line 1 Drop all non-numerical attributes from D to get D_N.
Line 2 Add a new column to D_N that consists of the constant 1, that is, $D'_N := [\vec{1}; D_N]$, where $\vec{1}$ denotes the column vector with 1 everywhere.
Line 3 Compute K eigenvectors of the square matrix ${D'_N}^T D'_N$, where K denotes the number of columns in D'_N.
Lines 5–6 Remove the first element of each eigenvector and normalize them to generate projections. (The 2-norm $||\vec{v}||$ is $\sqrt{\vec{v}^T\vec{v}}$.)
Line 7 Compute importance factor for each projection.
Line 8 Return the linear projections with corresponding normalized importance factors.

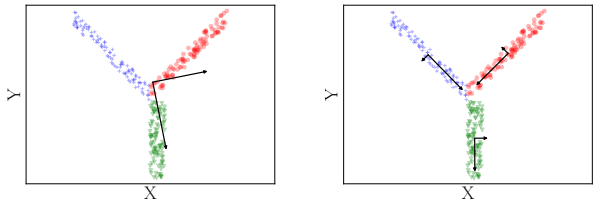
We now claim that the projections returned by Algorithm 1 include the projection with minimum standard deviation.

Theorem 4 (Correctness of Algorithm 1)**.** Given a numerical dataset D, let $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$ be the set of linear projections returned by Algorithm 1. Let $\sigma^* = \min_k^K \sigma(F_k(D))$. Then,
(1) $\sigma^* \leq \sigma(F(D))$ for every possible linear projection F, and
(2) $\forall F_j, F_k \in \mathcal{F}$ s.t. $F_j \neq F_k$, $\lim_{|D|\to\infty} \rho_{F_j,F_k} = 0$.

Algorithm 1: Procedure to generate linear projections.

Inputs : A dataset $D \subset \mathbf{Dom}^m$
Output: A set $\{(F_1, \gamma_1), \ldots, (F_K, \gamma_K)\}$ of projections and importance factors

$D_N \leftarrow D$ after dropping non-numerical attributes
$D'_N \leftarrow [\vec{1}; D_N]$
$\{\vec{w}_1, \ldots, \vec{w}_K\} \leftarrow$ eigenvectors of ${D'_N}^T D'_N$
foreach $1 \leq k \leq K$ **do**
  $\vec{w}'_k \leftarrow \vec{w}_k$ with first element removed
  $F_k \leftarrow \lambda \vec{A} : \frac{\vec{A}^T \vec{w}'_k}{||\vec{w}'_k||}$
  $\gamma_k \leftarrow \frac{1}{\log(2+\sigma(F_k(D_N)))}$
return $\{(F_1, \frac{\gamma_1}{Z}), \ldots, (F_K, \frac{\gamma_K}{Z})\}$, where $Z = \sum_k \gamma_k$



(a) PCA **(b)** Disjoint PCA

Figure 2: Learning PCA-based invariants globally results in low quality invariants when data satisfies strong local invariants.

If Algorithm 1 returns projections $F_1, \ldots, F_K$, and importance factors $\gamma_1, \ldots, \gamma_K$, then we generate the simple (conjunctive) invariant with K conjuncts: $\bigwedge_k \texttt{lb}_k \leq F_k(\vec{A}) \leq \texttt{ub}_k$, where the bounds $\texttt{lb}_k$ and $\texttt{ub}_k$ are computed as described in Section 4.1.1; we use the importance factor γ_k for the k^{th} conjunct in the quantitative semantics.

Note that the lowest variance principal component of $[\vec{1}; D_N]$ is close to the ordinary least squares estimate for predicting $\vec{1}$ from D. However, PCA offers multiple projections at once that range from low to high variance, and have low mutual correlation. For robustness, rather than discarding high variance projections, we assign them significantly small importance factors.

4.2 Compound Invariants

The quality of our PCA-based linear invariants (simple invariants) relies on how many low variance linear projections we are able to find on the given dataset. For many datasets, it is possible we find very few, or even none, such linear projections. In these cases, it is fruitful to search for compound invariants; we first focus on *disjunctive invariants* (defined by ψ_A in our language grammar).

Example 4. Our PCA-based approach fails in cases where there exist different piecewise linear trends within the data. If we apply PCA to learn invariants on the entire dataset of Figure 2(a), it will learn a low-quality invariant, with very high variance. In contrast, partitioning the dataset into three partitions (Figure 2(b)), and then learning invariants separately on each partition, will result in significant improvement of the learned invariants.

A disjunctive invariant is a compound invariant of the form $\bigvee_k ((A = c_k) \triangleright \phi_k)$, where each ϕ_k is not necessarily an invariant for all of D, but for a specific partition of D. Finding disjunctive invariants involves partitioning the dataset D into smaller (disjoint) datasets $D_1, D_2, \ldots$, where each D_k has the same attributes as D but only a subset of the rows of D.

Our strategy for partitioning D is to use categorical attributes with a small domain in D; in our implementation, we use those attributes A_j for which $|\{t.A_j | t \in D\}| \leq 50$. If A_j is such an attribute with values $v_1, v_2, \ldots, v_L$, we partition D into L disjoint datasets $D_1, D_2, \ldots, D_L$, where $D_l = \{t \in D | t.A_j = v_l\}$. Let $\phi_1, \phi_2, \ldots, \phi_L$ be the L simple invariants we learn for $D_1, D_2, \ldots, D_L$ using Algorithm 1, respectively. We compute the following disjunctive invariant for D:

$$((A_j = v_1) \triangleright \phi_1) \vee ((A_j = v_2) \triangleright \phi_2) \vee \cdots \vee ((A_j = v_L) \triangleright \phi_L)$$

Under closed-world semantics (i.e., A_j always takes one of the values $v_1, v_2, \ldots, v_L$) and Boolean violations, we can express this disjunctive invariant using notation from traditional denial constraints [16]:

$$\neg((A_j{=}v_1){\wedge}\neg\phi_1){\wedge}\neg((A_j{=}v_2){\wedge}\neg\phi_2){\wedge}\cdots{\wedge}\neg((A_j{=}v_L){\wedge}\neg\phi_L)$$

Note however that linear arithmetic inequalities are disallowed in denial constraints, which only allow atomic constraints that involve only one or two attributes in ϕ (with no arithmetic allowed). Our key contribution is discovering simple linear invariants ϕ, involving multiple numerical attributes. Also note that under an open-world assumption, compound invariants are more conservative than denial constraints. For example, a new tuple t with $t.A_j = v_{L+1}$, where $v_{L+1} \notin \{v_1, v_2, \ldots, v_L\}$, will satisfy the denial constraint but not the compound invariant.

We repeat this process and partition D over multiple attributes and generate a compound disjunctive invariant for each attribute. Finally, we generate a compound conjunctive invariant (Ψ), which is the conjunction of all these compound disjunctive invariants as the final data invariant for D.

4.3 Complexity Analysis

When computing simple invariants, there are two main computational steps: (1) computing $X^T X$, where X is an $n \times m$ matrix with n tuples (rows) and m attributes (columns), which takes $\mathcal{O}(nm^2)$ time, and (2) computing the eigenvalues and eigenvectors of an $m \times m$ positive definite matrix, which has complexity $\mathcal{O}(m^3)$ [62]. Once we obtain the linear projections using the above two steps, we need to compute the mean and variance of these projections on the original dataset, which takes $\mathcal{O}(nm^2)$ time. In summary, the overall procedure is cubic in the number of attributes and linear in the number of tuples.

When computing disjunctive invariants, we greedily pick attributes that take at most K (typically small) distinct values, and then run the above procedure for simple invariants at most K times. This adds just a constant factor overhead per attribute.

The procedure can be implemented in $\mathcal{O}(m^2)$ space. The key observation is that $X^T X$ can be computed as $\sum_{i=1}^{n} t_i t_i^T$, where t_i's are the n tuples in the dataset. Thus, $X^T X$ can be computed incrementally by loading only one tuple at a time into memory, computing $t_i t_i^T$, and then adding that to a running sum, which can be stored in $\mathcal{O}(m^2)$ space. Note that instead of such an incremental computation, this can also be done in an embarrassingly parallel way where we partition the data (row-wise) and each partition is then computed in parallel. Due to the low time and memory complexity, our approach scales gracefully to large datasets.

5. TRUSTED MACHINE LEARNING

In this section, we investigate the use of data invariants in Trusted Machine Learning (TML). In particular, we undertake a theoretical analysis of trusted machine learning by formalizing, in an ideal (noise-free) setting, the notion of "non-conformance" of a new tuple t w.r.t. an existing dataset D. We show that data invariants provide a sound and complete check for non-conformance. This provides justification for using data invariants to achieve

trusted machine learning. Since we perform our theoretical study in a noise-free setting, we use the *strict* notion of data invariants all through this section.

In trusted machine learning, we are interested in determining whether we can confidently use a prediction made by some machine-learned model on a new tuple. Since data can only provide an incomplete specification for most tasks, there is no certainty in predictions made using models learned from data, but some predictions are nevertheless more trustworthy than others. We now formalize the notion of *non-conforming* tuples, on which a machine-learned model produces an untrustworthy prediction. We focus on the setting of supervised machine learning, but our problem definition and solution approach naturally generalize to the setting of unsupervised learning as well.

5.1 Non-conforming Tuples

Consider the task of predicting an output $\hat{y} \in \texttt{coDom}$ for a tuple t, where coDom is the output domain. Let g be a function in $\mathcal{C}$ that represents the ground truth, i.e., the correct output $y = g(t)$, where $\mathcal{C}$ denotes some class of functions from $\mathbf{Dom}^m$ to coDom. Suppose that, a machine learning procedure learns a function $f \in \mathcal{C}$, using D as the training dataset and $Y \in \texttt{coDom}^n$ as the training output, where $Y = g(D)$.[1] So for tuple t, it predicts $f(t)$ as output.

Assuming the dataset D is an $n \times m$ matrix—containing n tuples, each with m attributes—we use $[D; Y]$ to denote the *annotated dataset* given by an $n \times (m+1)$ matrix, that is obtained by appending Y as a new column to D. We denote the i^{th} row of an annotated dataset $[D; Y]$ by (t_i, y_i).

Definition 3 (Non-conforming tuple)**.** Let $\mathcal{C}$ be a collection of functions with signature $\mathbf{Dom}^m \mapsto \texttt{coDom}$, and $[D; Y] \subset (\mathbf{Dom}^m \times \texttt{coDom})$ be an annotated dataset. $t \in \mathbf{Dom}^m$ is a non-conforming tuple w.r.t. $\mathcal{C}$ and $[D; Y]$, if there exist $f, g \in \mathcal{C}$ s.t. $f(D) = g(D) = Y$ but $f(t) \neq g(t)$.

Intuitively, a tuple t is non-conforming if it is possible to learn two different functions from the training data such that they both agree on all tuples in the training data, but disagree on t. This means that we cannot be sure whether f or g is the ground truth, because both are consistent with the observations Y, generated by the (unknown) ground truth, for D. This would not have been a problem if both f and g predicted the same output for t. Therefore, when that is not the case, we classify t as a non-conforming tuple, and argue that we should be cautious about the prediction on t, made by any model that learned from $[D; Y]$.

Proposition 5. If $t \in \mathbf{Dom}^m$ is a non-conforming tuple w.r.t. $\mathcal{C}$ and $[D; Y]$, then for any $f \in \mathcal{C}$ s.t. $f(D) = Y$, there exists a $g \in \mathcal{C}$ s.t. $g(D) = Y$ but $f(t) \neq g(t)$.

Note that even when we mark t as non-conforming, it is possible that the learned model makes the correct prediction. However, it is still useful to be aware of non-conforming tuples, because a-priori we do not know if the learned model actually matches the ground truth on tuples that fall outside of D.

The key point, when deciding whether a tuple t is non-conforming or not, is that we have access to the class $\mathcal{C}$ of functions (over which the learning procedure searches for a model) and the annotated dataset $[D; Y]$, but not to the actual learned model or the ground-truth function that generated Y from D. Hence, the computational procedure for detecting whether a tuple is non-conforming can only use knowledge of the class $\mathcal{C}$, $[D; Y]$, and t.

[1]With slight abuse of notation, we use $Y = g(D)$ to denote that $y_i = g(t_i)$, where y_i is the label for t_i, for all $1 \leq i \leq n$.

Example 5. Let $D = \{(0,1), (0,2), (0,3)\}$ be a dataset with two attributes A_1, A_2, and let the output Y be 1, 2, and 3, respectively. Let $\mathcal{C} \subseteq ((\mathbb{R} \times \mathbb{R}) \mapsto \mathbb{R})$ be the class of linear functions over two variables A_1 and A_2. Consider a new tuple $(1, 4)$. This is non-conforming since there exist two different functions, namely $f(A_1, A_2) = A_2$ and $g(A_1, A_2) = A_1 + A_2$, that agree with each other on D, but disagree on $(1, 4)$. In contrast, $(0, 4)$ is not non-conforming because there is no function in $\mathcal{C}$ that maps D to Y, but produces an output different from 4.

We start by providing some intuitions behind the use of data invariants in characterizing non-conforming tuples and then proceed to discuss the data invariant-based approach.

5.2 Data Invariants as Preconditions for TML

Let $\mathcal{C}$ be a fixed class of functions. Given a dataset D, suppose that a tuple t is non-conforming. This means that there exist $f, g \in \mathcal{C}$ s.t. $f(t) \neq g(t)$, but $f(D) = g(D)$. Now, consider the logical claim that $f(D) = g(D)$. Clearly, f is not identical to g since $f(t) \neq g(t)$. Therefore, there is a nontrivial "proof" (in some logic) of the fact that "for all tuples $t \in D : f(t) = g(t)$". This "proof" will use some facts (properties) about D, and let ϕ be the formula denoting these facts. If ϕ_D is the characteristic function for D, then the above argument can be written as,

$$\phi_D(\vec{A}) \Rightarrow \phi(\vec{A}), \quad \text{and} \quad \phi(\vec{A}) \Rightarrow f(\vec{A}) = g(\vec{A})$$

where $\Rightarrow$ denotes logical implication. In words, ϕ is a data invariant for D and it serves as a *precondition* in the "correctness proof" that shows (a potentially machine-learned model) f is equal to (potentially a ground truth) g. If a tuple t fails to satisfy the precondition ϕ, then it is possible that the prediction of f on t will not match the ground truth $g(t)$.

5.3 Non-conforming Tuple Detection: A Data Invariant-based Approach

Given a class $\mathcal{C}$ of functions, an annotated dataset $[D; Y]$, and a tuple t, our high-level procedure for determining whether t is non-conforming involves the following two steps:

1. Learn a data invariant ϕ for the dataset D.
2. Declare t as non-conformingif t does not satisfy ϕ.

This approach is sound and complete, thanks to the following proposition that establishes existence of a data invariant that characterizes whether a tuple is non-conforming or not.

Proposition 6. There exists an invariant Inv for D s.t. the following statement is true: "$t \notin \texttt{Inv}$ iff t is non-conforming w.r.t. $\mathcal{C}$ and $[D; Y]$ for all $t \in \mathbf{Dom}^m$".

Proposition 6 establishes existence of an ideal invariant, but does not yield a constructive procedure. In practice, we also have the common issue that the ideal invariant formula may not have a simple representation. Nevertheless, Proposition 6 provides motivation for finding invariants that can approximate this ideal invariant.

Example 6. We apply Proposition 6 on the sets D, Y, and $\mathcal{C}$ given in Example 5. Here, $\mathcal{C}$ is the set of all linear functions given by $\{\alpha A_1 + A_2 \mid \alpha \in \mathbb{R}\}$. The set Inv, whose complement characterizes the set of non-conforming tuples w.r.t. $\mathcal{C}$ and $[D; Y]$, is $\{(A_1, A_2) \mid \forall \alpha_1, \alpha_2 : \alpha_1 A_1 + A_2 = \alpha_2 A_1 + A_2\}$, which is equivalent to $\{(A_1, A_2) \mid A_1 = 0\}$.

5.4 Sufficient Check for Non-conformance

In practice, finding invariants that are necessary and sufficient for non-conformance is difficult. Hence, we focus on weaker invariants whose violation is sufficient, but not necessary, to classify

a tuple as non-conforming. We can use such invariants in the high-level approach mentioned in 5.3 to get a procedure that has false negatives (fails to detect some non-conforming tuples), but no false positives (never classifies a tuple as non-conforming when it is not).

Model Transformation using Equality Invariants

For certain invariants, we can prove that an invariant violation by t implies non-conformance of t by showing that those invariants can transform a model f that works on D to a different model g that also works on D, but $f(t) \neq g(t)$. We claim that equality invariants (of the form $F(\vec{A}) = 0$) are useful in this regard. First, we make the point using the scenario from Example 5.

Example 7. Consider the set $\mathcal{C}$ of functions, and the annotated dataset $[D; Y]$ from Example 5. The two functions f and g, where $f(A_1, A_2) = A_2$ and $g(A_1, A_2) = A_1 + A_2$, are equal when restricted to D; that is, $f(D) = g(D)$. What property of D suffices to prove $f(A_1, A_2) = g(A_1, A_2)$, i.e., $A_2 = A_1 + A_2$? It is $A_1 = 0$. Going the other way, if we have $A_1 = 0$, then $f(A_1, A_2) = A_2 = A_1 + A_2 = g(A_1, A_2)$. Therefore, we can use the equality invariant $A_1 = 0$ to transform the model f into the model g in such a way that the g continues to match the behavior of f on D. Thus, an equality invariant can be exploited to produce multiple different models starting from one given model. Moreover, if t violates the equality invariant, then it means that the models, f and g, would not agree on their prediction on t; for example, this happens for $t = (1, 4)$.

Let $F(\vec{A}) = 0$ be an equality invariant for the dataset D. If a learned model f returns a real number, then it can be transformed into another model $f + F$, which will agree with f only on tuples t where $F(t) = 0$. Thus, in the presence of equality invariants, a learner can return f or its transformed version $f+F$ (if both models are in the class $\mathcal{C}$). This condition is a "relevancy" condition that says that F is "relevant" for class $\mathcal{C}$. If the model does not return a real, then we can still use equality invariants to modify the model under some assumptions that include "relevancy" of the invariant.

A Theorem for Sufficient Check for Non-conformance

We first formalize the notions of *nontrivial* datasets—which are annotated datasets such that at least two output labels differ—and *relevant* invariants—which are invariants that can be used to transform models in a class to other models in the same class.

Nontrivial. An annotated dataset $[D; Y]$ is *nontrivial* if there exist i, j s.t. $y_i \neq y_j$.

Relevant. An invariant $F(\vec{A}) = 0$ is *relevant* to a class $\mathcal{C}$ of models if whenever $f \in \mathcal{C}$, then $\lambda t : f(\texttt{ite}(\alpha F(t), t^c, t)) \in \mathcal{C}$ for a constant tuple t^c and real number α. The if-then-else function $\texttt{ite}(r, t^c, t)$ returns t^c when $r = 1$, returns t when $r = 0$, and is free to return anything otherwise. If tuples admit addition, subtraction, and scaling, then one such if-then-else function is $t + r * (t^c - t)$.

We now state a sufficient condition for identifying a tuple as non-conforming. (Proofs are in the the Appendix.)

Theorem 7 (Sufficient Check for Non-conformance)**.** Let $[D; Y] \subset \mathbf{Dom}^m \times \texttt{coDom}$ be an annotated dataset, $\mathcal{C}$ be a class of functions, and F be a projection on $\mathbf{Dom}^m$ s.t.

A1. $F(\vec{A}) = 0$ is a strict invariant for D,
A2. $F(\vec{A}) = 0$ is relevant to $\mathcal{C}$,
A3. $[D; Y]$ is nontrivial, and
A4. there exists $f \in \mathcal{C}$ s.t. $f(D) = Y$.

For $t \in \mathbf{Dom}^m$, if $F(t) \neq 0$, then t is non-conforming.

We caution that our definition of non-conforming is liberal: existence of even one pair of functions f, g—that differ on t, but agree on the training set D—is sufficient to classify t as non-conforming. It ignores issues related to the probabilities of finding these models by a learning procedure. Our intended use of Theorem 7 is to guide the choice for the class of invariants, given the class $\mathcal{C}$ of models, so that we can use violation of an invariant in that class as an indication for caution. For most classes of models, linear arithmetic invariants are relevant.

Example 8. Consider the annotated dataset $[D; Y]$ and the class $\mathcal{C}$, from Example 7. Consider the equality invariant $F(A_1, A_2) = 0$, where the projection F is defined as $F(A_1, A_2) = A_1$. Clearly, $F(D) = \{0\ 0\ 0\}$, and hence, $F(A_1, A_2) = 0$ is an invariant for D. The invariant is also relevant to the class of linear models $\mathcal{C}$. Clearly, $[D; Y]$ is nontrivial, since $y_1 = 1 \neq 2 = y_2$. Also, there exists $f \in \mathcal{C}$ (e.g., $f(A_1, A_2) = A_2$) s.t. $f(D) = Y$. Now, consider the tuple $t = (1, 4)$. Since $F(t) = 1 \neq 0$, Theorem 7 implies that t is a non-conforming tuple.

6. EXPERIMENTAL EVALUATION

We evaluate data invariants over our two case-study applications (Section 2): TML and data drift. Our experiments target the following research questions:

[RQ1] How effective are data invariants for trusted machine learning? Is there a relationship between invariant violation score and the ML model's prediction accuracy? (Section 6.1)

[RQ2] Can data invariants be used to quantify data drift? How do they compare to other state-of-the-art drift-detection techniques? (Section 6.2)

[RQ3] Can data invariants be used to explain the causes for tuple non-conformance? (Section 6.3)

Efficiency. In all our experiments, our algorithms for deriving data invariants were extremely fast, and took only a few seconds even for datasets with 6 million rows. The number of attributes were reasonably small (∼40), which is true for most practical applications. As our theoretical analysis showed (Section 4.3), our approach is linear in number of data rows and cubic in number of attributes. Since the runtime performance of our techniques is straightforward, we opted to not include further discussion of efficiency here and instead focus this empirical analysis on the techniques' effectiveness.

Implementation: DISYNTH. We create an open-source implementation of data invariants and our method for synthesizing them, DISYNTH, in Python 3. Experiments were run on a Windows 10 machine (3.60 GHz processor and 16GB RAM).

Datasets

Airlines [7] contains data about flights and has 14 attributes, such as departure and arrival time, carrier, delay, etc. We used a subset of the data containing all flight information for year 2008. The training and test set contain 5.4M and 0.4M rows, respectively.

Human Activity Recognition (HAR) [81] is a real-world dataset about activities for 15 individuals, 8 males and 7 females, with varying fitness levels and BMIs. We use data from two sensors—accelerometer and gyroscope—attached to 6 body locations—head, shin, thigh, upper arm, waist, and chest. We consider 5 activities—lying, running, sitting, standing, and walking. The dataset contains 36 numerical attributes (2 sensors × 6 body-locations × 3 co-ordinates) and 2 categorical attributes—activity-type and person-ID. We pre-processed the dataset to aggregate the measurements over a small time window, resulting in 10,000 tuples per person and activity, for a total of 750,000 tuples.

Extreme Verification Latency (EVL) [77] is a widely used benchmark to evaluate drift-detection algorithms in non-stationary environments under extreme verification latency. It contains 16 synthetic datasets with incremental and gradual concept drifts. The number of attributes of these datasets vary from 2 to 6 and each of them has one categorical attribute.

Datasets for non-conformance explanation case studies. We evaluate the effectiveness of data invariants in explaining tuple non-conformance through an intervention-centric explanation tool built on top of DISYNTH, called ExTuNe [28]. We use four datasets for this evaluation: (1) *Cardiovascular Disease* [1] is a real-world dataset that contains information about cardiovascular patients with attributes such as height, weight, cholesterol level, glucose level, systolic and diastolic blood pressures, etc. (2) *Mobile Prices* [3] is a real-world dataset that contains information about mobile phones with attributes such as ram, battery power, talk time, etc. (3) *House Prices* [2] is a real-world dataset that contains information about houses for sale with attributes such as basement area, number of bathrooms, year built, etc. (4) *LED* (Light Emitting Diode) [10] is a synthetic benchmark. The dataset has a digit attribute, ranging from 0 to 9, 7 binary attributes—each representing one of the 7 LEDs relevant to the digit attribute—and 17 irrelevant binary attributes. This dataset includes gradual concept drift every 25,000 rows.

6.1 Trusted Machine Learning

We now demonstrate the applicability of DISYNTH in the trusted machine learning problem. We show that, test tuples that violate the data invariants derived from the training data are non-conforming, and therefore, a machine-learned model is more likely to perform poorly on those tuples.

Airlines. For the airlines dataset, we design a regression task of predicting the arrival delay and train a linear regression model for the task. Our goal is to observe how the mean absolute error (MAE) of the predicted value correlates to the invariant violation for the test tuples. In other words, we want to observe whether DISYNTH can correctly detect the non-conforming tuples.

In a process analogous to the one described in Example 1, our training dataset (`train`) comprises of daytime flights, i.e., flights that have arrival time later than the departure time. We design three test sets: (1) `Daytime`: flights that have arrival time later than the departure time (similar to `train`), (2) `Overnight`: flights that have arrival time earlier than the departure time (the dataset does not explicitly report the date of arrival), and (3) `Mixed`: a mixture of `Daytime` and `Overnight`.

Figure 3 shows the average violations of invariants derived by DISYNTH and the mean absolute errors (MAE) computed from the values predicted by a linear regression model. We note that invariant violation is a good proxy for prediction error. The reason is that DISYNTH derives invariants, such as "arrival time is later than departure time and their difference is very close to the flight duration," for `train`; the regression model makes the implicit assumption that these invariants always hold. Thus, when this assumption fails, the data invariant is violated and the regression performance also degrades.

To further investigate on the tuple granularity, we sample 1000 tuples from `Mixed`. We compute their invariant violations and show them in descending order of violations (Figure 4). Tuples on the left incur high violations and predictions for them also incur high absolute errors. Note that although DISYNTH was unaware of the target attribute (delay), it still correctly predicts when a tuple is non-conforming and the prediction is potentially untrustworthy.

HAR. On the HAR dataset, we design the supervised classification task to identify persons from their activity data. We construct

	Train	Test		
		Daytime	Overnight	Mixed
Average violation	0.0002	0.0002	0.2768	0.0887
MAE	18.95	18.89	80.54	38.60

Figure 3: Average invariant violation and MAE (linear regression) for four data splits on the airlines dataset. The invariants were learned on `train`, excluding the target attribute, delay.

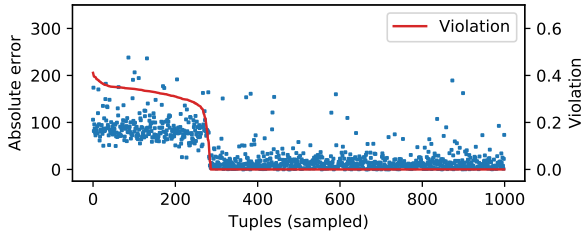


Figure 4: Invariant violation strongly correlates with the absolute error of delay prediction of a linear regression model.

`train_x` with data for sedentary activities (lying, standing, and sitting), and `train_y` with the corresponding person-IDs. We learn data invariants on `train_x`, and train a Logistic Regression classifier using the annotated dataset [`train_x`; `train_y`]. During test, we mix mobile activities (walking and running) with held-out data for sedentary activities and observe how the classification's mean accuracy-drop relates to average invariant violation. Figure 5(a) depicts our findings: classification degradation has a clear positive correlation with violation (pcc = 0.99 with p-value = 0).

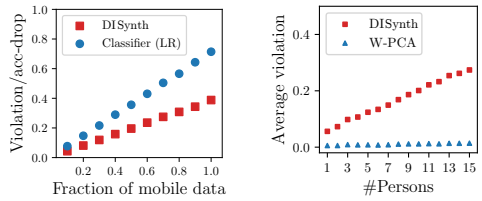
6.2 Data Drift

We now present results of using DISYNTH as a drift-detection tool; specifically, for *quantifying* drift in data. Given a baseline dataset D, and a new dataset D', the drift is measured as average violation of tuples in D' on invariants learned for D.

HAR. We perform three drift-quantification experiments on the HAR dataset which we discuss next.

Gradual drift. For observing how DISYNTH detects gradual drift, we introduce drift in an organic way. The initial training dataset contains data of exactly one activity for each person. This is a realistic scenario as one can think of it as taking a snapshot of what a group of people are doing at a particular, reasonably small, time window. We introduce gradual drift to the initial dataset by altering the activity of one person at a time. To control the amount of drift, we use a parameter K. When $K = 1$, the first person switches her activity, when $K = 2$, the second person also switches her activity, and so on. As we increase K from 1 to 15, we expect a gradual increase in the drift magnitude compared to the initial training data. When $K = 15$, all persons switch their activities, and we expect to observe maximum drift. We repeat this experiment 10 times, and display the average invariant violation in Figure 5(b). We note that the drift magnitude (violation) indeed increases as more people alter their activities.

In contrast, the baseline weighted-PCA (W-PCA) method fails to detect this drift. This is because W-PCA does not model local invariants (who is doing what), and learns some global invariants from the overall data. Thus, it fails to detect the gradual local drift, as the global situation "a group of people are performing some activities" is not changing. In contrast, DISYNTH learns disjunctive invariants that encode which person is performing which activity, and hence, is capable to detect drift when some individuals switch their activities.



(a) (b)

Figure 5: (a) As more fractions of mobile activity data is mixed with sedentary activity data, invariants are violated more, and the classifier's mean accuracy-drops keep increasing. (b) Detection of gradual local drift on HAR dataset. The drift magnitude increases as more people start changing their activities. In contrast, weighted-PCA (W-PCA) fails to detect drift in absence of any strong global drift.

	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	Fitness	BMI	Gender
p1	0	0.3	0.4	0.3	0.3	0.3	0.3	0.4	0.3	0.3	0.3	0.3	0.3	0.3	0.3	Moderate	Underweight	Female
p2	0.4	0	0.4	0.3	0.3	0.3	0.3	0.5	0.3	0.3	0.4	0.3	0.4	0.4	0.3	Moderate	Normal	Male
p3	0.5	0.5	0	0.4	0.5	0.5	0.5	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	Moderate	Overweight	Male
p4	0.2	0.2	0.3	0	0.3	0.2	0.3	0.5	0.2	0.3	0.2	0.3	0.3	0.3	0.3	Moderate	Normal	Male
p5	0.2	0.3	0.4	0.2	0	0.2	0.2	0.4	0.2	0.3	0.3	0.3	0.3	0.3	0.3	Moderate	Normal	Male
p6	0.3	0.3	0.4	0.3	0.3	0	0.3	0.4	0.2	0.2	0.2	0.2	0.3	0.2	0.4	High	Normal	Female
p7	0.3	0.3	0.4	0.3	0.3	0.3	0	0.4	0.3	0.2	0.3	0.3	0.3	0.3	0.3	Moderate	Overweight	Male
p8	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0	0.4	0.5	0.4	0.5	0.5	0.5	0.5	Low	Obese	Female
p9	0.4	0.4	0.5	0.4	0.3	0.3	0.5	0.5	0	0.3	0.4	0.4	0.4	0.4	0.5	High	Overweight	Male
p10	0.4	0.3	0.4	0.3	0.3	0.3	0.4	0.5	0.3	0	0.3	0.3	0.3	0.4	0.3	Moderate	Obese	Male
p11	0.4	0.4	0.5	0.3	0.4	0.3	0.4	0.5	0.3	0.3	0	0.3	0.3	0.3	0.4	Moderate	Normal	Female
p12	0.3	0.3	0.4	0.3	0.3	0.3	0.3	0.5	0.3	0.3	0.3	0	0.3	0.4	0.4	Moderate	Normal	Female
p13	0.3	0.3	0.4	0.3	0.3	0.2	0.3	0.4	0.2	0.3	0.2	0.3	0	0.3	0.4	Moderate	Normal	Female
p14	0.3	0.3	0.4	0.3	0.3	0.2	0.3	0.4	0.2	0.3	0.3	0.3	0.3	0	0.3	High	Normal	Male
p15	0.4	0.4	0.5	0.4	0.4	0.5	0.4	0.5	0.5	0.4	0.4	0.4	0.5	0.5	0	Low	Normal	Female

Figure 6: Inter-person invariant violation heat map. Each person has a very low self-violation.

	lying	standing	sitting	walking	running
lying	0.05	0.41	0.57	0.68	0.78
standing	0.62	0.02	0.51	0.56	0.71
sitting	0.57	0.23	0.04	0.59	0.72
walking	0.21	0.01	0.06	0	0.25
running	0.12	0	0.03	0.02	0.01

Figure 7: Inter-activity invariant violation heat map. Mobile activities violate the invariants of the sedentary activities more.

Inter-person drift. The goal of this experiment is to observe drift among persons. We use DISYNTH to learn disjunctive invariants for each person over all activities, and use the violation w.r.t. the learned invariants to measure how much the other persons drift. Figure 6 illustrates our findings. The violation score at row `p1` and column `p2` denotes how much `p2` drifts from `p1`. We use half of each person's data to learn the invariants, and compute violation on the held out data. While computing drift between two persons, we compute activity-wise invariant violation scores and then average them out. As one would expect, we observe a very low self-drift across the diagonal. Interestingly, our result also shows that some people are more different from others, which appears to have some correlation with (the hidden ground truth) fitness and BMI values. This asserts that the invariants we learn for each person are an accurate abstraction of that person's activities, as people do not deviate too much from their usual activity patterns.

Inter-activity drift. Similar to inter-person invariant violation, we also compute inter-activity invariant violation. Figure 7 shows our findings. Note the asymmetry of violation scores between activities, e.g., `running` is violating the invariants of `standing` much more than the other way around. A close observation reveals that, all mobile activities violate all sedentary activities more than the other way around. This is due to the fact that, the mobile activities behave as a "safety envelope" for the sedentary activities. For example, while a person walks, she also stands (for a brief moment). But the opposite does not happen.

EVL. We now compare DISYNTH against other state-of-the-art drift detection approaches on the EVL benchmark.

Baseline Approaches. In our experiments, we use two drift-detection approaches as baselines which we describe below:

(1) PCA-SPLL [53] similar to us, also argues that principal components with lower variance are more sensitive to a general drift, and uses those for dimensionality reduction. It then models multivariate distribution over the reduced dimensions and applies semi-parametric log-likelihood (SPLL) to detect drift between two multivariate distributions. However, PCA-SPLL discards all high-variance principal components and does not model disjunctive invariants.

(2) CD (Change Detection) [68] is another PCA-based approach for drift detection in data streams. But unlike PCA-SPLL, it ignores low-variance principal components. CD projects the data onto top k high-variance principal components, which results into multiple univariate distributions. We compare against two variants of CD: CD-Area, which uses the intersection area under the curves of two density functions as a divergence metric, and CD-MKL, which uses Maximum KL-divergence as a symmetric divergence metric, to compute divergence between the univariate distributions.

Figure 8 depicts how DISYNTH compares against CD-MKL, CD-Area, and PCA-SPLL, on 16 datasets in the EVL benchmark. For PCA-SPLL, we retain principal components that contribute to a cumulative explained variance below 25%. Beyond drift detection, which just detects if drift is above some threshold, we focus on drift quantification. A tuple (x, y) in the plots denotes that drift magnitude for dataset at x^{th} time window, w.r.t. the dataset at the first time window, is y. Since different approaches report drift magnitudes in different scales, we normalize the drift values within $[0, 1]$. Additionally, since different datasets have different number of time windows, for the ease of exposition, we normalize the time window indices. Below we state our key findings from this experiment:

DISYNTH*'s drift quantification matches the ground truth.* In all of the datasets in the EVL benchmark, DISYNTH is able to correctly quantify the drift, which matches the ground truth exceptionally well.[2] In contrast, as CD focuses on detecting the drift point, it is ill-equipped to precisely quantify the drift, which is demonstrated in several cases (e.g., 2CHT), where CD fails to distinguish the deviation in drift magnitudes. In contrast, both PCA-SPLL and DISYNTH correctly quantify the drift. Since CD only retains high-variance principal components, it is more susceptible to noise and considers noise in the dataset as significant drift, which leads to incorrect drift quantification. In contrast, PCA-SPLL and DISYNTH ignore the noise and only capture the general notion of drift. In all of the EVL datasets, we found CD-Area to work better than CD-MKL, which also agrees with the authors' experiments.

DISYNTH *models local drift.* When the dataset contains instances from multiple classes, the drift may be just local, and not global. Figure 9 demonstrates such a scenario for the 4CR dataset. If we ignore the color/shape of the tuples, we will not observe any significant drift across different time steps. In such cases, PCA-SPLL fails to detect drift (4CR, 4CRE-V2, and FG-2C-2D). In contrast, DISYNTH learns disjunctive invariants and can quantify local drifts accurately.

[2]EVL video: `sites.google.com/site/nonstationaryarchive/home`

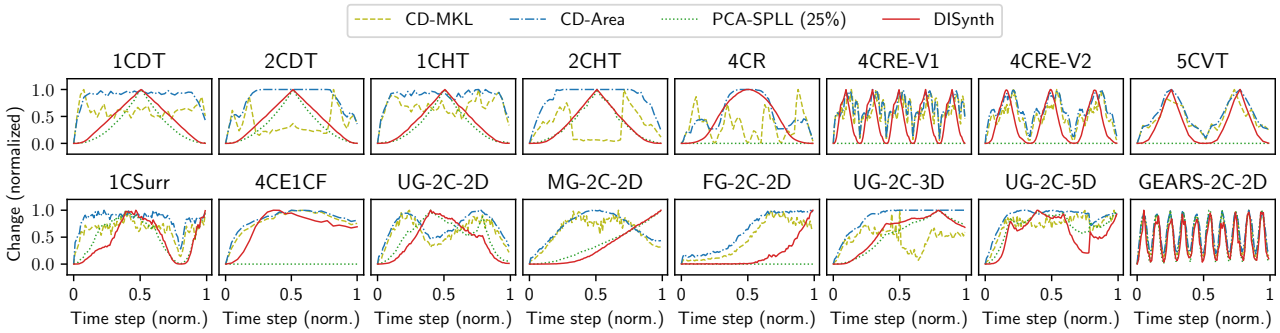


Figure 8: In the EVL benchmark, DISYNTH quantifies drift correctly for all cases, outperforming other approaches. PCA-SPLL fails to detect drift in a few cases by discarding all principal components; CD-MKL and CD-Area are too sensitive to small drift and detect spurious drifts.

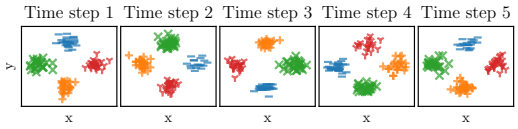


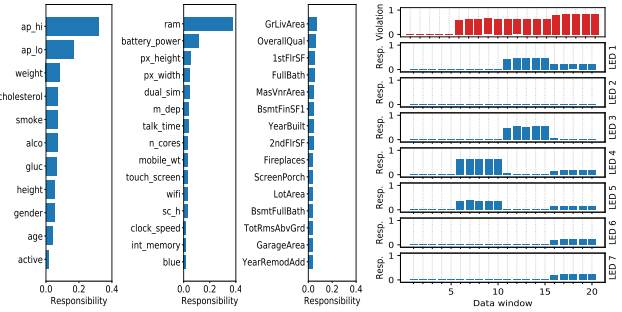
Figure 9: Snapshots over time for 4CR dataset with local drift. It reaches maximum drift from the initial distribution at time step 3 and goes back to the initial distribution at time step 5.

6.3 Explaining Non-conformance

When a test dataset is determined to be sufficiently different or drifted from the training set, the next step often is to characterize the difference. A common way of characterizing these differences is to perform a causality or responsibility analysis to determine which attributes are most responsible for the observed drift (non-conformance). We use the violation values produced by data invariants, along with well-established principles of causality, to quantify responsibility for non-conformance.

ExTuNe. We built a tool ExTuNe [28], on top of DISYNTH, to compute the responsibility values as described next. Given a training dataset D and a non-conforming tuple $t \in \mathbf{Dom}^m$, we measure the *responsibility* of the i^{th} attribute A_i towards the non-conformance as follows: (1) We intervene on $t.A_i$ by altering its value to the mean of A_i over D to obtain the tuple $t^{(i)}$. (2) In $t^{(i)}$, we compute how many additional attributes need to be altered to obtain a tuple with no violation. If K additional attributes need to be altered, A_i has responsibility $\frac{1}{K+1}$. (3) This responsibility value for each tuple t can be averaged over the entire test dataset to obtain an aggregate responsibility value for A_i. Intuitively, for each tuple, we are "fixing" the value of A_i with a "more typical" value, and checking how close (in terms of additional fixes required) this takes us to a conforming tuple. The larger the number of additional fixes required, the lower the responsibility of A_i.

Case studies. ExTuNe produces bar-charts of responsibility values as depicted in Figure 10. Figures 10(a), 10(b), and 10(c) show the explanation results for Cardiovascular Disease, Mobile Price, and House Price datasets, respectively. For the cardiovascular disease dataset, the training and test sets consist of data for patients without and with cardiovascular disease, respectively. For the House Price and Mobile Price datasets, the training and test sets consist of houses and mobiles with prices below and above a certain threshold, respectively. As one can guess, we get many useful insights from the non-conformance responsibility bar-charts such as: "abnormal (high or low) blood pressure is a key cause for non-conformance of patients with cardiovascular disease w.r.t. normal people", "RAM is a distinguishing factor between expensive and cheap mobiles", "the reason for houses being expensive depends holistically on several attributes".



(a) **(b)** **(c)** **(d)**

Figure 10: Responsibility assignment on attributes for drift on (a) Cardiovascular disease: trained on patients with no disease and tested on patients with disease, (b) Mobile Prices: trained on cheap mobiles and tested on expensive mobiles and (c) House Prices: trained on house with price <= 100K and tested on house with price >= 300K. (d) Detection of drift on LED dataset. The dataset drifts every 5 windows (25,000 tuples). At each drift, a certain set of LEDs malfunction and take responsibility of the drift.

Figure 10(d) shows a similar result on the LED dataset. Instead of one test set, we had 20 test sets (the first set is also used as a training set to learn data invariants). We call each test set a window where each window contains 5,000 tuples. This dataset introduces gradual concept drift every 25,000 rows (5 windows) by making a subset of LEDs malfunctioning. As one can clearly see, during the initial 5 windows, no drift is observed. In the next 5 windows, LED 4 and LED 5 starts malfunctioning; in the next 5 windows, LED 1 and LED 3 starts malfunctioning, and so on.

7. RELATED WORK

Applications of data invariants. In database systems, data invariants can be used to detect change in data and query workloads, which can help in database tuning and indexing [50], required for cardinality estimation and query optimization. Data invariants have application in data cleaning (error detection and missing value imputation): the degree of invariant violation serves as a measure of error, and missing values can be imputed by exploiting relationships among attributes that data invariants capture. Data invariants can also detect outliers by exposing the tuples that significantly violate the invariants. Another interesting data management application of data invariants is *data-diff* [79] for exploring differences between two datasets: our disjunctive invariants can explain how different partitions of the two datasets vary.

In machine learning, data invariants can be used to suggest when to retrain a machine-learned model. Further, given a pool of machine-learned models and the corresponding training datasets, we can use data invariants to *synthesize* a new model for a new

dataset. A simple way to achieve this is to pick the model such that invariants learned from its training data are minimally violated by the new dataset. Finally, identifying non-conforming tuples is analogous to *input validation*: perform sanity checks on an input before it is processed by an application.

Data profiling. Data invariants fall under the umbrella of data profiling using metadata [4]. Metadata provides an insight of the dataset and has various use cases, such as database design [91, 40], data exploration [23], data summarization [15], data cleaning [25, 12, 17], data integration [26, 88], query optimization [90, 32, 56], reverse data management [60, 66, 27], data validation and sanitization [93], big data analytics [94], and so on. There is extensive literature on data-profiling primitives that model relationships among data attributes, such as unique column combinations [34], functional dependencies (FD) [63] and their variants (metric [51], conditional [26], soft [43], approximate [41, 52], relaxed [14], etc.), differential dependencies [75], order dependencies [55, 80], inclusion dependencies [64, 58], and denial constraints [16, 11, 65]. However, none of the existing approaches focus on learning approximate arithmetic relationships that involve multiple numerical attributes in a noisy setting, which is the focus of our work.

Relaxed functional dependencies [14] relax the traditional definition of FDs. Some replace data equality with data similarity, some allow a certain (small) fraction of tuples in the dataset to violate the learned FDs, and some allow both. Some variants of FDs consider noisy data, but they require the allowable noise parameter to be explicitly specified by the user. For example, approximate FDs [41, 52] allow violations by a certain fraction of tuples (specified as a system parameter), but determining the right setting for this parameter is non-trivial. Metric FDs [51] allow small variations in the data (attribute values within a distance of a user-specified threshold are considered identical), but the existing work focuses on verification of a given metric FD, rather than the discovery of metric FDs. All of these approaches treat violation as Boolean, and do not measure the degree of violation. In contrast, we do not require any explicit noise parameter and provide a way to quantify the degree of violation of data invariants.

Conditional FDs [26] require the FDs to be satisfied conditionally (e.g., a FD may hold for US residents and a different FD for Europeans). Denial constraints (DC) are a universally-quantified first-order-logic formalism [16] and can adjust to noisy data, by adding predicates until the constraint becomes exact over the entire dataset. However, this can make DCs large, complex, and uninterpretable. While approximate denial constraints [65] exist, similar to approximate FD techniques, they also rely on the users to provide the error threshold. Soft FDs [44] model correlation and generalize traditional FDs by allowing uncertainty, but are limited in modeling relationships between only a pair of attributes.

Trusted AI. The issue of trust, resilience, and interpretability of artificial intelligence (AI) systems has been a theme of increasing interest recently [45, 86, 72], particularly for high-stakes and safety-critical data-driven AI systems [87, 83]. A standard way to decide whether to trust a classifier or not, is to use the classifier-produced prediction confidence score. However, as a prior work [46] argues, this is not always effective since the classifier's confidence scores are not well-calibrated. The work on explaining black-box machine-learned models [54, 69] does not directly address the detection of non-conforming tuples.

Data drift. Prior work on data drift, change detection, and covariate shift [6, 13, 19, 20, 22, 24, 37, 38, 48, 49, 73, 76, 42] relies on modeling the data distribution, where change is detected when the data distribution changes. However, data distribution does not necessarily capture data invariants, which is the primary focus of our work. Moreover, these are based on multivariate distribution modeling, without emphasizing low-variance projections, and provide poor interpretability. Instead of detecting drift globally, only a handful of works model local concept-drift [84] or drift for imbalanced data [89]. Few data-drift detection mechanisms rely on active learning [18, 95], availability of classification accuracy [30, 9, 29, 71], or availability of classification "blindspots" [74]. Some of these works focus on adapting change in data, i.e., learning in an environment where change in data is expected [9, 31, 61, 78, 92]. Such adaptive techniques are useful to obtain better performance for specific tasks; however their goal is orthogonal to ours.

Representation learning and one-class classification. Few works [21, 35], related to our data invariant-based approach, observe the quality of an autoencoder's [36, 70] input reconstruction and use it to determine if a new data point is out-of-distribution. Another data sanity check mechanism [57] learns data *assertions* via autoencoders towards effective detection of invalid inputs during system runtime. However, such an approach is task-specific and needs a specific system (e.g., a deep neural network) to begin with.

Learning data invariants and using that to detect non-conformance fall in the area of one-class-classification (OCC) [82], where the training data contains tuples from only the positive class. Data invariants also achieve OCC, but do so under the *additional requirement* that they generalize the data in a way that is aligned with the generalization obtained by a given class of ML models.

In general, there exists a rich literature in the machine learning community on representation learning using generative models [5], black-box models such as auto-encoders [36, 70], Bayesian representation learning [47], and so on. These techniques are sophisticated and require significant training effort including data cleaning, encoding the entire data to numerical domain, hyper-parameter tuning, and so on. In contrast, data invariants are more abstract, yet informative, descriptions that can be constructed using efficient and highly-scalable techniques. There is a clear gap between representation learning (that models data likelihood) and the (constraint-oriented) data-profiling techniques. Our aim is to bridge this gap by introducing data invariants—which can model approximate arithmetic relationship over numerical attributes—as a new kind of constraint to the existing data-profiling techniques.

8. SUMMARY AND FUTURE DIRECTIONS

We introduced data invariants, and the notion of non-conforming tuples for trusted machine learning. We presented an efficient and highly-scalable approach for synthesizing data invariants, and demonstrated their effectiveness to tag non-conforming tuples, explain causes of non-conformance, and quantify data drift. The experiments validate our theory and our principle of using low variance projections to generate useful data invariants.

There is tremendous potential for future work. We have only studied two use-cases from a large pool of potential applications of data invariants. We have also restricted ourselves to linear invariants, but powerful nonlinear invariants can be extracted from autoencoders, which will significantly expand the class of models where these new invariants will be relevant. Moreover, invariants can be learned in a decision-tree-like structure where categorical attributes will guide the splitting conditions and leaves of the decision tree will contain simpler invariants. Further, we envision a mechanism to explore differences between two datasets—built on top of data invariants—where the user can explore differences between sub-populations of the datasets in an interactive (in a drill-down fashion) way.

9. REFERENCES

[1] Cardiovascular disease: https://www.kaggle.com/sulianova/cardiovascular-disease-dataset, Feb 2020.

[2] House prices: Advanced regression techniques: Advanced regression techniques. https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data, Feb 2020.

[3] Mobile prices: https://www.kaggle.com/iabhishekofficial/mobile-price-classification, Feb 2020.

[4] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDB J.*, 24(4):557–581, 2015.

[5] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017.

[6] C. C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *SIGMOD*, pages 575–586, 2003.

[7] Airline dataset., 2019. http://kt.ijs.si/elena_ikonomovska/data.html.

[8] J. P. Barddal, H. M. Gomes, F. Enembreck, and B. Pfahringer. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, 127:278–294, 2017.

[9] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *SDM*, pages 443–448, 2007.

[10] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, 2010.

[11] T. Bleifuß, S. Kruse, and F. Naumann. Efficient denial constraint discovery with hydra. *PVLDB*, 11(3):311–323, 2017.

[12] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 746–755, 2007.

[13] L. Bu, C. Alippi, and D. Zhao. A pdf-free change detection test based on density difference estimation. *IEEE Trans. Neural Netw. Learning Syst.*, 29(2):324–334, 2018.

[14] L. Caruccio, V. Deufemia, and G. Polese. On the discovery of relaxed functional dependencies. In *Proceedings of the 20th International Database Engineering & Applications Symposium*, pages 53–61, 2016.

[15] V. Chandola and V. Kumar. Summarization - compressing data into an informative representation. *Knowl. Inf. Syst.*, 12(3):355–378, 2007.

[16] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.

[17] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.

[18] A. F. J. Costa, R. A. S. Albuquerque, and E. M. dos Santos. A drift detection method based on active learning. In *IJCNN*, pages 1–8, 2018.

[19] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Symp. on the Interface of Statistics, Computing Science, and Applications*, 2006.

[20] R. F. de Mello, Y. Vaz, C. H. G. Ferreira, and A. Bifet. On learning guarantees to unsupervised concept drift detection on data streams. *Expert Syst. Appl.*, 117:90–102, 2019.

[21] T. Denouden, R. Salay, K. Czarnecki, V. Abdelzad, B. Phan, and S. Vernekar. Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance. *CoRR*, abs/1812.02765, 2018.

[22] D. M. dos Reis, P. A. Flach, S. Matwin, and G. E. A. P. A. Batista. Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. In *SIGKDD*, pages 1545–1554, 2016.

[23] M. Drosou and E. Pitoura. Ymaldb: exploring relational databases via result-driven recommendations. *VLDB J.*, 22(6):849–874, 2013.

[24] W. J. Faithfull, J. J. R. Diez, and L. I. Kuncheva. Combining univariate approaches for ensemble change detection in multivariate data. *Information Fusion*, 45:202–214, 2019.

[25] W. Fan, F. Geerts, and X. Jia. A revival of integrity constraints for data cleaning. *PVLDB*, 1(2):1522–1523, 2008.

[26] W. Fan, F. Geerts, L. V. S. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 1231–1234, 2009.

[27] A. Fariha and A. Meliou. Example-driven query intent discovery: Abductive reasoning using semantic similarity. *PVLDB*, 12(11):1262–1275, 2019.

[28] A. Fariha, A. Tiwari, A. Radhakrishna, and S. Gulwani. ExTuNe: Explaining tuple non-conformance. (to appear as a demo). In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Portland, OR, USA, June 14-19, 2020*, 2020.

[29] M. M. Gaber and P. S. Yu. Classification of changes in evolving data streams using online clustering result deviation. In *Proc. Of International Workshop on Knowledge Discovery in Data Streams*, 2006.

[30] J. Gama, P. Medas, G. Castillo, and P. P. Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence - SBIA*, pages 286–295, 2004.

[31] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, 2014.

[32] W. Ge, X. Li, C. Yuan, and Y. Huang. Correlation-aware partitioning for skewed range query optimization. *World Wide Web*, 22(1):125–151, 2019.

[33] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017.

[34] A. Heise, J. Quiané-Ruiz, Z. Abedjan, A. Jentzsch, and F. Naumann. Scalable discovery of unique column combinations. *PVLDB*, 7(4):301–312, 2013.

[35] D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *CoRR*, abs/1610.02136, 2016.

[36] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[37] S. Ho. A martingale framework for concept change detection in time-varying data streams. In *ICML*, pages 321–327, 2005.

[38] B. Hooi and C. Faloutsos. Branch and border: Partition-based change detection in multivariate time series. In *SDM*, pages 504–512, 2019.

[39] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, New York, NY, USA, 2nd edition, 2012.

[40] B. C. Housel, V. E. Waddle, and S. B. Yao. The functional dependency model for logical database design. In *Fifth International Conference on Very Large Data Bases, October 3-5, 1979, Rio de Janeiro, Brazil, Proceedings*, pages 194–208, 1979.

[41] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.

[42] D. Ienco, A. Bifet, B. Pfahringer, and P. Poncelet. Change detection in categorical evolving data streams. In *Symposium on Applied Computing, SAC*, pages 792–797, 2014.

[43] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 647–658, 2004.

[44] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 647–658, 2004.

[45] S. Jha. Trust, resilience and interpretability of AI models. In *Numerical Software Verification - 12th International Workshop, NSV@CAV*, pages 3–25, 2019.

[46] H. Jiang, B. Kim, M. Y. Guan, and M. R. Gupta. To trust or not to trust A classifier. In *NeurIPS*, pages 5546–5557, 2018.

[47] T. Karaletsos, S. Belongie, and G. Rätsch. Bayesian representation learning with oracle constraints. *arXiv preprint arXiv:1506.05011*, 2015.

[48] Y. Kawahara and M. Sugiyama. Change-point detection in time-series data by direct density-ratio estimation. In *SDM*, pages 389–400, 2009.

[49] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *PVLDB*, pages 180–191, 2004.

[50] R. Koch. Sql database performance tuning for developers. `https://moa.cms.waikato.ac.nz/datasets/`, Sep 2013.

[51] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1275–1278. IEEE, 2009.

[52] S. Kruse and F. Naumann. Efficient discovery of approximate dependencies. *Proceedings of the VLDB Endowment*, 11(7):759–772, 2018.

[53] L. I. Kuncheva and W. J. Faithfull. PCA feature extraction for change detection in multidimensional unlabeled data. *IEEE Trans. Neural Netw. Learning Syst.*, 25(1):69–80, 2014.

[54] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec. Faithful and customizable explanations of black box models. In *AAAI/ACM Conference on AI, Ethics, and Society, AIES*, pages 131–138, 2019.

[55] P. Langer and F. Naumann. Efficient order dependency detection. *VLDB J.*, 25(2):223–241, 2016.

[56] H. Liu, D. Xiao, P. Didwania, and M. Y. Eltabakh. Exploiting soft and hard correlations in big data query optimization. *PVLDB*, 9(12):1005–1016, 2016.

[57] H. Lu, H. Xu, N. Liu, Y. Zhou, and X. Wang. Data sanity check for deep learning systems via learnt assertions. *CoRR*, abs/1909.03835, 2019.

[58] F. D. Marchi, S. Lopes, and J. Petit. Unary and n-ary inclusion dependency discovery in relational databases. *J. Intell. Inf. Syst.*, 32(1):53–73, 2009.

[59] T. Martin and I. K. Glad. Online detection of sparse changes in high-dimensional data streams using tailored projections. *arXiv preprint arXiv:1908.02029*, 2019.

[60] A. Meliou, W. Gatterbauer, and D. Suciu. Reverse data management. *PVLDB*, 4(12):1490–1493, 2011.

[61] Z. Ouyang, Y. Gao, Z. Zhao, and T. Wang. Study on the classification of data streams with concept drift. In *International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, volume 3, pages 1673–1677. IEEE, 2011.

[62] V. Y. Pan and Z. Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 507–516. ACM, 1999.

[63] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.

[64] T. Papenbrock, S. Kruse, J.-A. Quiané-Ruiz, and F. Naumann. Divide & conquer-based inclusion dependency discovery. *Proceedings of the VLDB Endowment*, 8(7):774–785, 2015.

[65] E. H. Pena, E. C. de Almeida, and F. Naumann. Discovery of approximate (and exact) denial constraints. *Proceedings of the VLDB Endowment*, 13(3):266–278, 2019.

[66] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of renormalized relational databases. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 218–227. IEEE, 1996.

[67] A. Qahtan, N. Tang, M. Ouzzani, Y. Cao, and M. Stonebraker. Pattern functional dependencies for data cleaning. *Proceedings of the VLDB Endowment*, 13(5):684–697, 2020.

[68] A. A. Qahtan, B. Alharbi, S. Wang, and X. Zhang. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *SIGKDD*, pages 935–944, 2015.

[69] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *SIGKDD*, pages 1135–1144, 2016.

[70] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[71] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda. A new method for data stream mining based on the misclassification error. *IEEE Trans. Neural Netw. Learning Syst.*, 26(5):1048–1059, 2015.

[72] S. Saria and A. Subbaswamy. Tutorial: Safe and reliable machine learning. *CoRR*, abs/1904.07204, 2019.

[73] T. S. Sethi and M. M. Kantardzic. On the reliable detection of concept drift from streaming unlabeled data. *Expert Syst. Appl.*, 82:77–99, 2017.

[74] T. S. Sethi, M. M. Kantardzic, and E. Arabmakki. Monitoring classification blindspots to detect drifts from unlabeled data. In *IEEE International Conference on Information Reuse and Integration, IRI*, pages 142–151, 2016.

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

[76] X. Song, M. Wu, C. M. Jermaine, and S. Ranka. Statistical change detection for multi-dimensional data. In *SIGKDD*, pages 667–676, 2007.

[77] V. M. A. Souza, D. F. Silva, J. Gama, and G. E. A. P. A. Batista. Data stream classification guided by clustering on nonstationary environments and extreme verification latency. In *SDM*, pages 873–881, 2015.

[78] A. Subbaswamy, P. Schulam, and S. Saria. Preventing failures due to dataset shift: Learning predictive models that transport. In *International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 3118–3127, 2019.

[79] C. A. Sutton, T. Hobson, J. Geddes, and R. Caruana. Data diff: Interpretable, executable summaries of changes in distributions for data wrangling. In *SIGKDD*, pages 2279–2288, 2018.

[80] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava. Effective and complete discovery of order dependencies via set-based axiomatization. *PVLDB*, 10(7):721–732, 2017.

[81] T. Sztyler and H. Stuckenschmidt. On-body localization of wearable devices: An investigation of position-aware activity recognition. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–9, 2016.

[82] D. M. J. Tax and K. Müller. Feature extraction for one-class classification. In *ICANN/ICONIP*, pages 342–349, 2003.

[83] A. Tiwari, B. Dutertre, D. Jovanovic, T. de Candia, P. Lincoln, J. M. Rushby, D. Sadigh, and S. A. Seshia. Safety envelope for security. In *International Conference on High Confidence Networked Systems (part of CPS Week), HiCoNS*, pages 85–94, 2014.

[84] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In *IEEE International Symposium on Computer-Based Medical Systems (CBMS)*, pages 679–684, 2006.

[85] M. Tveten. Which principal components are most sensitive to distributional changes? *arXiv preprint arXiv:1905.06318*, 2019.

[86] K. R. Varshney. Trustworthy machine learning and artificial intelligence. *ACM Crossroads*, 25(3):26–29, 2019.

[87] K. R. Varshney and H. Alemzadeh. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big Data*, 5(3):246–255, 2017.

[88] D. Z. Wang, X. L. Dong, A. D. Sarma, M. J. Franklin, and A. Y. Halevy. Functional dependency generation and applications in pay-as-you-go data integration systems. In *WebDB*, 2009.

[89] H. Wang and Z. Abraham. Concept drift detection for imbalanced stream data. *CoRR*, abs/1504.01044, 2015.

[90] G. E. Weddell. Reasoning about functional dependencies generalized for semantic data models. *ACM Trans. Database Syst.*, 17(1):32–64, 1992.

[91] Z. Wei and S. Link. Embedded functional dependencies and data-completeness tailored database design. *PVLDB*, 12(11):1458–1470, 2019.

[92] S. Yu, Z. Abraham, H. Wang, M. Shah, Y. Wei, and J. C. Príncipe. Concept drift detection and adaptation with hierarchical hypothesis testing. *J. Franklin Institute*, 356(5):3187–3215, 2019.

[93] Y. Yu and J. Heflin. Extending functional dependency to detect abnormal data in rdf graphs. In *International Semantic Web Conference*, pages 794–809. Springer, 2011.

[94] J. Zhang, X. Yang, and D. Appelbaum. Toward effective big data analysis in continuous auditing. *Accounting Horizons*, 29(2):469–476, 2015.

[95] I. Zliobaite, A. Bifet, G. Holmes, and B. Pfahringer. Moa concept drift active learning strategies for streaming data. In T. Diethe, J. Balcazar, J. Shawe-Taylor, and C. Tirnauca, editors, *Proceedings of the Second Workshop on Applications of Pattern Analysis*, volume 17 of *Proceedings of Machine Learning Research*, pages 48–55. PMLR, 19–21 Oct 2011.

APPENDIX

A. PROOF OF LEMMA 2

Proof. Pick β_1, β_2 s.t. $\beta_1^2 + \beta_2^2 = 1$ and the following equation holds:

$$\texttt{sign}(\rho_{F_1,F_2})\beta_1\sigma(F_1(D)) + \beta_2\sigma(F_2(D)) = 0 \quad (1)$$

Let t be any tuple that is incongruous w.r.t. $\langle F_1, F_2\rangle$. Now, we compute how far t is from the mean of the projection F on D:

$$\begin{aligned}|\Delta F(t)| &= |F(t) - \mu(F(D))| \\ &= |\beta_1 F_1(t) + \beta_2 F_2(t) - \mu(\beta_1 F_1(D) + \beta_2 F_2(D))| \\ &= |\beta_1 \Delta F_1(t) + \beta_2 \Delta F_2(t)| \\ &= |\beta_1 \Delta F_1(t)| + |\beta_2 \Delta F_2(t)|\end{aligned}$$

The last step is correct only when $\beta_1\Delta F_1(t)$ and $\beta_2\Delta F_2(t)$ are of same sign. We prove this by cases:

(Case 1). $\rho_{F_1,F_2} \geq \frac{1}{2}$. In this case, β_1 and β_2 are of different signs due to Equation 1. Moreover, since t is incongruous w.r.t. $\langle F_1, F_2\rangle$, $\Delta F_1(t)$ and $\Delta F_2(t)$ are of different signs. Hence, $\beta_1\Delta F_1(t)$ and $\beta_2\Delta F_2(t)$ are of same sign.

(Case 2). $\rho_{F_1,F_2} \leq -\frac{1}{2}$. In this case, β_1 and β_2 have the same sign due to Equation 1. Moreover, since t is incongruous w.r.t. $\langle F_1, F_2\rangle$, $\Delta F_1(t)$ and $\Delta F_2(t)$ are of same sign. Hence, $\beta_1\Delta F_1(t)$ and $\beta_2\Delta F_2(t)$ are of same sign.

Next, we compute the variance of F on D:

$$\begin{aligned}\sigma(F(D))^2 =& \frac{1}{|D|}\sum_{t\in D}(\beta_1\Delta F_1(t) + \beta_2\Delta F_2(t))^2 \\ =& \beta_1^2\sigma(F_1(D))^2 + \beta_2^2\sigma(F_2(D))^2 \\ &+ 2\beta_1\beta_2\rho_{F_1,F_2}\sigma(F_1(D))\sigma(F_2(D)) \\ =& \beta_1^2\sigma(F_1(D))^2 + \beta_1^2\sigma(F_1(D))^2 - 2\beta_1^2|\rho_{F_1,F_2}|\sigma(F_1(D))^2 \\ =& 2\beta_1^2\sigma(F_1(D))^2(1 - |\rho_{F_1,F_2}|)\end{aligned}$$

Hence, $\sigma(F(D)) = \sqrt{2(1-|\rho_{F_1,F_2}|)}|\beta_1|\sigma(F_1(D))$, which is also equal to $\sqrt{2(1-|\rho_{F_1,F_2}|)}|\beta_2|\sigma(F_2(D))$. Since $\sqrt{2(1-|\rho_{F_1,F_2}|)}| \leq 1$, and since $|\beta_k| < 1$, we conclude that $\sigma(F(D)) < \sigma(F_k(D))$. Now, we compute $\frac{|\Delta F(t)|}{\sigma(F(D))}$ next using the above derived facts about $|\Delta F(t)|$ and $\sigma(F(D))$.

$$\begin{aligned}\frac{|\Delta F(t)|}{\sigma(F(D))} &> \frac{|\beta_1\Delta F_1(t)|}{\sqrt{2(1-|\rho_{F_1,F_2}|)}|\beta_1|\sigma(F_1(D))} \\ &= \frac{|\Delta F_1(t)|}{\sqrt{2(1-|\rho_{F_1,F_2}|)}\sigma(F_1(D))} \quad \geq \frac{|\Delta F_1(t)|}{\sigma(F_1(D))}\end{aligned}$$

The last step uses the fact that $|\rho_{F_1,F_2}| \geq \frac{1}{2}$. Similarly, we also get $\frac{|\Delta F(t)|}{\sigma(F(D))} > \frac{|\Delta F_2(t)|}{\sigma(F_2(D))}$. Hence, ϕ_F is stronger than both ϕ_{F_1} and ϕ_{F_2} on d, using Lemma 1. This completes the proof. □

B. PROOF OF THEOREM 3

Proof. First, we use Lemma 2 on F_i, F_j to construct F. We initialize $I := \{i, j\}$. Next, we repeatedly do the following: We iterate over all F_k, where $k \notin I$, and check if $|\rho_{F,F_k}| \geq \frac{1}{2}$ for some k. If yes, we use Lemma 2 (on F and F_k) to update F to be the new projection F returned by the lemma. We update $I := I \cup \{k\}$, and continue the iterations. If $|\rho_{F,F_k}| < \frac{1}{2}$ for all $k \notin I$, then we stop. The final F and index set I can easily be seen to satisfy the claims of the theorem. □

C. PROOF OF THEOREM 4

Proof. Since D is numerical, $D_N = D$. Let $F = \lambda\vec{A} : \vec{A}^T\vec{w}$ be an arbitrary linear projection. By definition of D'_N, the variance $\sigma(D_N\vec{w})^2$ of F equals $||D'_N \begin{bmatrix}-\mu(D_N\vec{w}) \\ \vec{w}\end{bmatrix}||^2$ divided by the number of rows in D_N. So, to find $\vec{w}$ (with $||\vec{w}|| = 1$) that minimizes $\sigma(D_N\vec{w})$, we can instead find $\vec{v}$ that minimizes $||D'_N\vec{v}||$, where $\vec{v}$ is of the form $\begin{bmatrix}-\mu(D_N\vec{w}) \\ \vec{w}\end{bmatrix}$ and $||\vec{w}|| = 1$. The key point to note is that we do not need the latter restriction since if $\begin{bmatrix}c \\ \vec{w}\end{bmatrix}$ minimizes $||D'_N\vec{v}||$ with $||\vec{w}|| = 1$, then $c = -\mu(D_N\vec{w})$ (To prove this, expand $||D'_N\vec{v}||$ to get an expression quadratic in c and set its derivative to 0 to get $c = -\mu(D_N\vec{w})$). Finally, we know by Courant-Fischer min-max theorem [39] that the $\vec{v}$ that minimizes $||D'_N\vec{v}||$ is the eigenvector of ${D'_N}^T D_N$ corresponding to the lowest eigenvalue. Together, this shows that $\sigma^* \leq \sigma(F(D))$ for every possible linear projection F. Note that if there are l linearly independent projections F with the same standard deviation $\sigma^* = \sigma(F(D))$, then the lowest eigenvalue of ${D'_N}^T D'_N$ will occur with multiplicity l. There will exist l linearly independent (orthogonal) eigenvectors of ${D'_N}^T D'_N$. The return value of our procedure will include l projections (not necessarily independent or orthogonal) on D_N, whose span will contain any F with $\sigma(F(D_N)) = \sigma^*$.

For part (2) of the claim, we note that:

(1) If $\begin{bmatrix}c \\ \vec{w}\end{bmatrix}$ is an eigenvector of ${D'_N}^T D'_N$ corresponding to eigenvalue l, then $c = \frac{-\mu}{1-\frac{l}{|D|}}$, where μ is the mean of $D'_N\vec{w}$. Thus, as $\lim_{|D|\to\infty} c = -\mu$.

(2) If F is a projection defined by $\begin{bmatrix}-\mu \\ \vec{w}\end{bmatrix}$, where μ is the mean of $D\vec{w}$, then σ_F is equal to $||D'_N\vec{w}||/\sqrt{|D|}$.

(3) If F_j is defined by $\vec{v} := \begin{bmatrix}-\mu_j \\ \vec{v}_1\end{bmatrix}$, and F_k is defined by $\vec{w} := \begin{bmatrix}-\mu_k \\ \vec{w}_1\end{bmatrix}$, where μ_j and μ_k are the means of $D\vec{v}_1$ and $D\vec{w}_1$ respectively, then ρ_{F_j,F_k} is exactly equal to $\frac{(D'_N\vec{v})^T D'_N\vec{w}}{|D|\sigma_{F_j}\sigma_{F_k}}$.

(4) If F_j is defined by $\vec{v} := \begin{bmatrix}c_j \\ \vec{v}_1\end{bmatrix}$, and F_k is defined by $\vec{w} := \begin{bmatrix}c_k \\ \vec{w}_1\end{bmatrix}$, and $\vec{v}$ and $\vec{w}$ are eigenvectors of ${D'_N}^T D'_N$, then as $|D|$ increases to ∞, c_j approaches $-\mu_j$, and c_k approaches $-\mu_k$, and hence, ρ_{F_j,F_k} approaches $\frac{(D'_N\vec{v})^T D'_N\vec{w}}{|D|\sigma_{F_j}\sigma_{F_k}}$, which is identically 0 since $(D'_N\vec{v})^T D'_N\vec{w} = \vec{v}^T {D'_N}^T D'_N\vec{w} = \vec{v}^T\varepsilon_w\vec{w} = 0$. where ε_w is the eigenvalue for eigenvector $\vec{w}$. The last step assumes that the eigenvectors are orthogonal, which is always the case when eigenvalues are distinct, and can be assumed without loss of generality if eigenvalues are the same. □

D. PROOF OF PROPOSITION 5

Proof. By the definition of non-conforming tuple, there exist $g, g' \in \mathcal{C}$ s.t. $g(D) = g'(D) = Y$, but $g(t) \neq g'(t)$. Now, given a function $f \in \mathcal{C}$ s.t. $f(D) = Y$, the value $f(t)$ can be either equal to $g(t)$ or $g'(t)$, but not both. WLOG, say $f(t) \neq g(t)$. Then, we have found a function g s.t. $g(D) = Y$ but $f(t) \neq g(t)$, which completes the proof. □

E. PROOF OF PROPOSITION 6

Proof. Consider the set $\texttt{Inv} := \{t \mid f(t) = g(t) \text{ for all } f, g \in \mathcal{C} \text{ s.t. } f(D) = g(D) = Y\}$.

First, we claim that Inv is an invariant for D. For this, we need to prove that $D \subseteq \texttt{Inv}$. Consider any $t' \in D$. We need to prove that $f(t') = g(t')$ for all $f, g \in \mathcal{C}$ s.t. $f(D) = g(D) = Y$. Since $t' \in D$, and since $f(D) = g(D)$, it follows that $f(t') = g(t')$. This shows that Inv is an invariant for D.

Next, we claim that Inv does not contain tuples that are non-conforming w.r.t. $\mathcal{C}$ and $[D; Y]$. Consider any $t' \notin \texttt{Inv}$. By definition of Inv, it follows that there exist $f, g \in \mathcal{C}$ s.t. $f(D) = g(D) = Y$, but $f(t') \neq g(t')$. This is equivalent to saying that t' is non-conforming by definition. Hence, Inv contains exactly those tuples that are *not* non-conforming w.r.t. $\mathcal{C}$ and $[D; Y]$. □

F. PROOF OF THEOREM 7

Proof. WLOG, let t_1, t_2 be the two tuples in D s.t. $y_1 \neq y_2$ (A3). Since $f(D) = Y$ (A4), it follows that $f(t_1) = y_1 \neq y_2 = f(t_2)$. Let t be a new tuple s.t. $F(t) \neq 0$. Clearly, $f(t)$ can not be equal to both y_1 and y_2. WLOG, suppose $f(t) \neq y_1$. Now, consider the function g defined by $\lambda\tau : f(\texttt{ite}(F(\tau), t_1, \tau))$. By (A2), we know that $g \in \mathcal{C}$. Note that $g(D) = Y$ since for any tuple $t_i \in D$, $F(t_i) = 0$ (A1), and hence $g(t_i) = f(\texttt{ite}(0, t_1, t_i)) = f(t_i) = y_i$. Thus, we have two models, f and g, s.t. $f(D) = g(D) = Y$. To prove that t is a non-conforming tuple, we have to show that $f(t) \neq g(t)$. Note that $g(t) = f(\texttt{ite}(F(t), t_1, t)) = f(t_1) = y_1$ (by definition of g). Since we already had $f(t) \neq y_1$, it follows that we have $f(t) \neq g(t)$. This completes the proof. □

G. SYSTEM PARAMETERS

Our technique for deriving (unnormalized) importance factor γ_k, for bounded-projection invariant on projection F_k, uses the mapping $\frac{1}{\log(2+\sigma(F_k(D)))}$. This mapping correctly translates our principles for quantifying violation by putting high weight on low-variance projections, and low weight on high-variance projections. While this mapping works extremely well across a large set of applications (including the ones we show in our experimental results), our quantitative semantics are not limited to any specific mapping. Indeed, the function to compute importance factors for bounded-projections can be user-defined (but we do not require it from the user). Specifically, a user can plug in any custom function to derive the (unnormalized) importance factors. Furthermore, our technique to compute the bounds lb and ub can also be customized (but we do not require it from the user either). Depending on the application requirements, one can apply techniques used in machine learning literature (e.g., cross-validation) to tighten or loosen the data invariants by tuning these parameters. However, we note that our technique for deriving these parameters are very effective in most practical applications.