ExTuNe: Explaining Tuple Non-conformance

Anna Fariha
University of Massachusetts
Amherst, MA, USA
afariha@cs.umass.edu

Ashish Tiwari
Microsoft
Bellevue, WA, USA
astiwar@microsoft.com

Arjun Radhakrishna
Microsoft
Bellevue, WA, USA
arradha@microsoft.com

Sumit Gulwani
Microsoft
Bellevue, WA, USA
sumitg@microsoft.com

ABSTRACT

In data-driven systems, we often encounter tuples on which the predictions of a machine-learned model are untrustworthy. A key cause of such untrustworthiness is *non-conformance* of a new tuple with respect to the training dataset. To check conformance, we introduce a novel concept of *data invariant*, which captures a set of implicit constraints that all tuples of a dataset satisfy: a test tuple is non-conforming if it violates the data invariants. Data invariants model complex relationships among multiple attributes; but do not provide interpretable explanations of non-conformance. We present ExTuNe, a system for **Ex**plaining causes of **Tu**ple **N**on-conformanc**e**. Based on the principles of causality, ExTuNe assigns *responsibility* to the attributes for causing non-conformance. The key idea is to observe change in invariant violation under *intervention* on attribute-values. Through a simple interface, ExTuNe produces a ranked list of the test tuples based on their degree of non-conformance and visualizes tuple-level attribute responsibility for non-conformance through heat maps. ExTuNe further visualizes attribute responsibility, aggregated over the test tuples. We demonstrate how ExTuNe can detect and explain tuple non-conformance and assist the users to make careful decisions towards achieving trusted machine learning.

KEYWORDS

Causality; Trusted machine learning; Data invariants

ACM Reference Format:
Anna Fariha, Ashish Tiwari, Arjun Radhakrishna, and Sumit Gulwani. 2020. ExTuNe: Explaining Tuple Non-conformance. In *2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3318464.3384694

1 INTRODUCTION

In data-driven systems, we often encounter tuples on which the predictions of a machine-learned model are untrustworthy. Early detection of such tuples is necessary to ensure Trusted Machine Learning (TML) [3]. Since data is inherently an incomplete specification for any task, invariably there will exist multiple different models that can be learned from the given training dataset. This in return introduces uncertainty in predictions made using any specific model learned from the dataset. Motivated by the issue of trusting the predictions made by machine learning, we define *non-conforming tuples*—which are tuples on which a machine-learned model makes untrustworthy predictions. One might see non-conforming tuples as outliers. However, traditional definition of outlier overlooks the fact that some outliers are non-conforming, for certain tasks, while others are not.

Example 1.1. Consider a dataset with three training tuples with predictor attributes x_1 and x_2: $\{(1, 10), (2, 20), (4, 40)\}$. We consider a task-agnostic setting and hence omit the target attribute. Now consider two new tuples: $t_1 = (3, 12)$ and $t_2 = (10, 100)$. A traditional distance-based outlier detector will mark t_2 as an outlier and possibly t_1 as an inlier. However, ML models often exploit the consistent relationship observed between the attribute pairs ($x_2 = 10x_1$) within the training tuples, and assume it as an "invariant". Since t_1 violates this invariant, a model that uses $10x_1$ instead of x_2 is likely to make inaccurate prediction on t_1, if x_2 is the true predictor. Here, t_1 is non-conforming and t_2 is conforming with the invariant.

Example 1.1 shows the shortcoming of distance-based outliers in capturing the notion of non-conformance. Identification of non-conforming tuples is a two-step process, which is presented in our previous work [1]: (1) learning *data invariants* from the training dataset, and (2) checking for violation of the learned invariants by the test tuples; violation of data invariants indicates non-conformance. In this paper, we demonstrate ExTuNe, which (1) detects non-conforming tuples and quantifies the degree of non-conformance based on [1], and (2) extends [1] to explain the cause of non-conformance by assigning degree of responsibility to tuple attributes.

SIGMOD '20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3384694

Pattern-based outlier detection mechanisms [8] do not explain outliers. Scorpion [12] explains outliers through common explanation, but does not consider relationships among attributes. In general, no prior work on explaining outliers [6, 7, 12] addresses the question: "which attributes or attribute relationships are responsible for non-conformance?"

Learning data invariants and using that to detect non-conformance is similar to one-class-classification (OCC) [10], where the training data contains tuples from only one 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. Data invariants also share similarity to functional dependencies in databases but the latter encode stricter invariants and do not work well for numeric attributes. Data drift and covariate shift [9] detection has connection to non-conformance detection. However, their techniques are based on multivariate distribution modeling, without emphasizing low-variance dimensions, and provide poor interpretability.

ExTuNe's attribute responsibility has connection to feature importance and feature selection for binary classification. However, ExTuNe works in an OCC setting, where data from other classes (counter-examples) are unavailable during training. Unlike binary classifiers, which start with the knowledge of tuples from both classes, ExTuNe's goal is to (1) predict which tuples fall in the negative class (i.e., non-conforming), and (2) assign responsibility to the attributes for non-conformance. Feature importance for OCC fails here too, as it only considers the training data and overlooks the test tuples. Responsibility computation in ExTuNe is generic and can be applied to any technique (distance- or pattern-based outlier detectors or classifiers) that distinguishes two classes (representing conforming and non-conforming tuples); however, ExTuNe applies it to non-conformance based on data invariants due to our interest in TML.

In our demonstration, participants will observe how ExTuNe detects the non-conforming tuples and provides real-time explanation for non-conformance. We proceed to discuss, at a high level, the solution sketch of ExTuNe and conclude with a detailed outline of our demonstration.

2 SOLUTION SKETCH

The key component of ExTuNe is an invariant learner, which learns data invariants from the training tuples (Section 2.1). When a new test tuple arrives, ExTuNe checks if the test tuple satisfies the invariant, and if it does not, ExTuNe quantifies the degree of non-conformance. To generate explanation for non-conformance, ExTuNe uses an intervention-centric approach that alters values of attributes, and observes change in invariant violation. ExTuNe then assigns degree of responsibility to the attributes for non-conformance (Section 2.2).

2.1 Data Invariants

Our approach for computing data invariants is based on "inverted" principal component analysis (PCA) [1]. Our key observation is that low-variance principal components are the most useful predictors of non-conformance [4, 10, 11]. Essentially, we define invariants on the principal components, while emphasizing the low-variance components.

Traditional ML techniques often discard the low-variance components, indicating their implicit assumption that the low-variance projections will continue to have low variance on test data. ExTuNe's non-conformance test is based on checking this assumed precondition for ML models. Intuitively, for a dataset D, we compute the data invariant I that encodes that "the projection of each tuple in D onto the low-variance component results in values that lie within 4 standard deviations from the mean of the corresponding projection." An invariant I is a constraint that defines a set of *conforming* tuples and $\vec{x} \vdash I$ denotes that the tuple $\vec{x}$ belongs to that conforming tuple set (i.e., satisfies the invariant I).

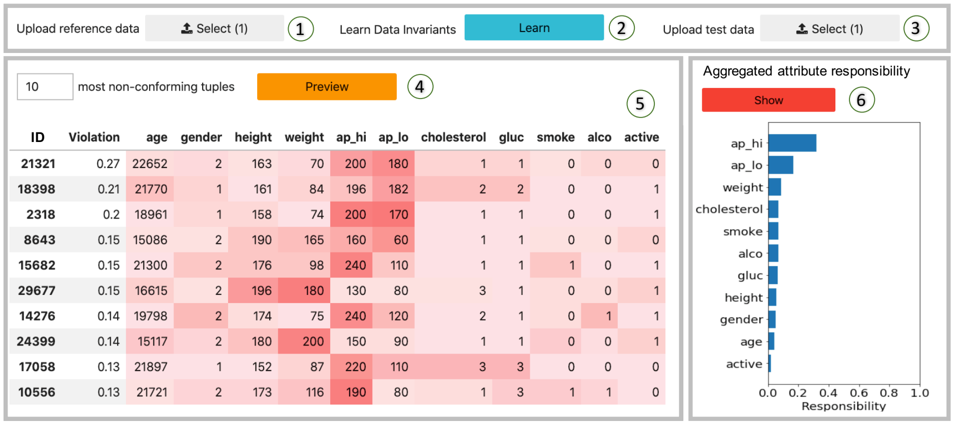
We associate a numeric value characterizing the degree of invariant violation by a tuple. The violation function $violation(\vec{x}, I)$ denotes the "distance" of the tuple $\vec{x}$ from the invariant I with the property that when $\vec{x}$ satisfies I ($\vec{x} \vdash I$), $violation(\vec{x}, I) = 0$; and as $\vec{x}$ moves away from I ($\vec{x} \nvdash I$), $violation(\vec{x}, I)$ approaches 1. A tuple can be non-conforming by violating several invariants. To aggregate the violations w.r.t. a set of invariants, we compute a weighted sum over them while putting more weight on low-variance invariants and less weight on high-variance invariants.

We discover data invariants using principal component analysis, which is efficient and scalable: it is *linear* in number of tuples and *cubic* in number of attributes. Further, we empirically found that data invariants serve as an effective proxy for the trust on the ML model's prediction: when a test tuple violates the learned data invariants, then the model's prediction on that tuple is likely to be inaccurate [1].

2.2 Responsibility for Non-conformance

Responsibility quantifies the contribution of attributes of a tuple for causing non-conformance. In this paper, we adapt Halpern and Pearl's [2] definition of causality. To measure the degree of responsibility of attributes of tuples that violate data invariants, we adapt the notion of *degree of responsibility* from Meliou et al. [5]. We reason about causality by *intervening* on attribute-values: we alter value of an attribute to the attribute-mean over the training dataset, and observe how it affects the tuple's invariant violation.

Counterfactual cause. By definition, C is a counterfactual cause of an event E if E would not occur unless C occurs. In our case, an event is the violation of invariant I by a tuple $\vec{x} = \langle x_1, x_2, \ldots, x_i, \ldots, x_m \rangle$, i.e., $\vec{x} \nvdash I$. The fact



ID	Violation	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
21321	0.27	22652	2	163	70	200	180	1	1	0	0	0
18398	0.21	21770	1	161	84	196	182	2	2	0	0	1
2318	0.2	18961	1	158	74	200	170	1	1	0	0	1
8643	0.15	15086	2	190	165	160	60	1	1	0	0	0
15682	0.15	21300	2	176	98	240	110	1	1	1	0	1
29677	0.15	16615	2	196	180	130	80	3	1	0	0	1
14276	0.14	19798	2	174	75	240	120	2	1	0	1	1
24399	0.14	15117	2	180	200	150	90	1	1	0	0	1
17058	0.13	21897	1	152	87	220	110	3	3	0	0	0
10556	0.13	21721	2	173	116	190	80	1	3	1	1	0

Figure 1: The ExTuNe demo: ① upload reference data, ② learn data invariants, ③ upload test data, ④ select the number of most non-conforming tuples to preview, ⑤ tuple-wise attribute-responsibility heat map, ⑥ aggregated attribute responsibility.

$A_i = x_i$ is a counterfactual cause for the violation if intervening on the value of A_i *prevents* the invariant violation: $\langle x_1, x_2, \ldots, \mu_i, \ldots x_m \rangle \vdash I$, where μ_i denotes the attribute-mean of A_i. In such case, we assign responsibility 1 to attribute A_i.

Actual cause. C is an actual cause of E if E counterfactually depends on C under some *permissive contingency* [5]. To determine causality for an attribute-value that is not a counterfactual cause, we allow alteration of other attributes *that are not counterfactual causes* as permissive contingency. However, we only allow alterations that involve changing the value of an attribute to its attribute-mean. We define *minimum support* to be the minimum number of attribute-value alterations required within a contingency, among all possible contingencies, to achieve counterfactual causality. Contingencies with minimum support are the *minimal contingencies*. We assign responsibility $\frac{1}{M+1}$ to an attribute with minimum support M.

Example: Consider a tuple $\vec{x}=\langle x_1, x_2, \ldots, x_m \rangle$ s.t. $\vec{x} \nvdash I$. Suppose that, individually, none of $A_1=x_1$ and $A_2=x_2$ are counterfactual causes for the violation, but $\langle \mu_1, \mu_2, x_3, \ldots, x_m \rangle \vdash I$. Since $A_1=x_1$ is a counterfactual cause for the violation under the contingency $A_2=\mu_2$, it is an actual cause with minimum support 1. So, we assign responsibility $\frac{1}{1+1}=\frac{1}{2}$ to A_1. Using a symmetric argument, A_2 also gets responsibility $\frac{1}{2}$.

Approximating minimal contingency. Finding the minimal contingency for an actual cause is NP-hard [5]. Hence, we follow a greedy approach to find an approximate minimal contingency. While this approach does not guarantee optimality and might even fail to identify an actual cause, it works well in practice, particularly when responsibility is aggregated over a large set of non-conforming tuples.

The greedy approach iteratively selects attributes to alter based on their contribution to the invariant violation. For example, consider the invariant $-3 \leq F(\vec{A}) \leq 3$ where $F(\vec{A}) = 2A_1 + 4A_2 + 7A_3 + 5A_4 + 4A_5$ and attribute-mean is 0 for all attributes. For a tuple $\vec{x} = \langle 6, -5, 2, 3, 2 \rangle$, $F(\vec{x}) = 12 + (-20) + 14 + 15 + 8 = 29$. None of the attribute-values are counterfactual in this case. Now, suppose that we are looking for minimal contingency (if one exists) of $A_1 = 6$. Clearly, $\vec{x}$ violated the invariant due to 29 being too high than the upper bound 3 of the invariant. We start by *greedily picking* A_4 to alter as it contributes *the maximum* (15) to $F(\vec{x})$. We obtain $\vec{x}' = \langle 6, -5, 2, \underline{0}, 2 \rangle$ and $F(\vec{x}') = 12 + (-20) + 14 + 0 + 8 = 14$. With this contingency, $A_1 = 6$ now counterfactually causes the violation as for $\vec{x}'' = \langle \underline{0}, -5, 2, \underline{0}, 2 \rangle$, $F(\vec{x}'') = 0 + (-20) + 14 + 0 + 8 = 2$, which satisfies the invariant. So, we found $M = 1$ and hence assign responsibility $\frac{1}{2}$ to A_1. If this was not a valid contingency, we would continue to intervene on the next most contributing attribute (A_3 in this case) to find a valid contingency.

Aggregating responsibility. Following the above procedure, we compute $R_{i,j,k}$ which denotes the responsibility of attribute A_i for causing tuple $\vec{x}^{(j)}$ to violate invariant I_k. To ensure that responsibilities are proportionate to the degree of violations, we multiply $R_{i,j,k}$ by $violation(\vec{x}^{(j)}, I_k)$. Finally, we aggregate responsibilities over all invariants and tuples and normalize the responsibilities across all attributes.

3 DEMONSTRATION

We will demonstrate ExTuNe on a real-world cardiovascular disease dataset.[1] The dataset contains information about patients with attributes such as height, weight, cholesterol level, glucose level, systolic and diastolic blood pressures, etc. We expect that most participants will be familiar with this data domain and will be able to correctly interpret the explanations ExTuNe provides. Our goal is to show that ExTuNe can effectively detect non-conforming tuples and meaningfully assign responsibility to attributes for non-conformance.

[1] Cardiovascular disease: kaggle.com/sulianova/cardiovascular-disease-dataset

3.1 Explaining Tuple Non-conformance

Figure 1 shows a screenshot of ExTuNe's graphical user interface, built within a Jupyter Notebook. During the demonstration, we will guide the participants through six steps. We have annotated each step with a circle in Figure 1.

Step ① (Upload reference data): First, the user uploads a reference dataset, whose invariants she is interested to learn. No tuple within the reference dataset should be non-conforming, i.e., the reference data should be clean. For our guided scenario, we upload the tuples with *absence of cardiovascular disease* as the reference data.

Step ② (Learn data invariants): The user issues request for learning data invariants. Following the procedure described in Section 2.1, ExTuNe learns invariants and reports, typically within a few seconds, that the learning is done.

Step ③ (Upload test data): The user uploads the test data containing potentially non-conforming tuples. She wants to identify the non-conforming tuples and understand the cause of non-conformance. For our guided scenario, we use tuples with *presence of cardiovascular disease* as test data.

Step ④ (Top-K non-conforming tuples): The user requests to preview top-K non-conforming tuples from the test dataset. She chooses the value of $K = 10$.

Step ⑤ (Tuple-wise attribute-responsibility heat map): ExTuNe shows top-10 most non-conforming tuples, based on the invariants learned from the reference dataset, in a table where the left-most column denotes identifier of the tuple, followed by the degree of non-conformance (violation). The rest of the table-cells depict a heat map which assigns darker color on more responsible attributes, and lighter color on less responsible ones. The heat map visualizes causes of non-conformance in a tuple-level granularity.

In this dataset, `height` is in cm and `weight` is in kg; `cholesterol` and `glucose` mappings are: 1 = normal, 2 = above normal, 3 = well above normal; `ap_hi` and `ap_lo` correspond to the systolic and diastolic blood pressure measurements, whose normal values are 120 and 80, respectively.

For the first tuple, the non-conformance comes mostly from the abnormally high blood pressures. For the second tuple, besides abnormally high blood pressures, he also has above normal glucose and cholesterol levels. For the sixth tuple, although the blood pressures look normal, she has an abnormally high weight of 180 kg (397 lbs) which is one of the prime causes for her non-conformance.

Step ⑥ (Aggregated attribute responsibility): The user issues a request to visualize attribute responsibility for over-all non-conformance of the test data. ExTuNe visualizes the responsibility of different attributes for non-conformance, aggregated over all tuples in the test data: systolic blood pressure is most responsible for non-conformance, followed by diastolic blood pressure. This is very meaningful since abnormal blood pressure is a primary indicator for cardiovascular disease. This is followed by weight, cholesterol level, and smoking, three other well-known risk factors.

Demonstration engagement. After our guided demonstration, participants will be able to plug their own datasets into ExTuNe. We will also make two additional datasets—MobilePrices[2] (includes attributes such as ram, battery-power, talk-time, etc.) and HousePrices[3] (includes attributes such as area, number-of-rooms, year-built, etc.)—available, as we expect most participants to be familiar with these data domains.

Through the demonstration, we will showcase how ExTuNe can effectively detect non-conforming tuples and explain the causes of the observed non-conformance. Particularly, we expect that the participants will be able to relate the degree of responsibility assigned to different attributes to their real-life experiences (e.g., abnormal blood pressure being a key cause for cardiovascular disease). The key takeaway that our demonstration will highlight is that detecting non-conforming tuples and understanding their causes can significantly help users make decisions about (1) when to trust machine learning models and when not, and (2) how to enrich the training data towards building more robust models.

REFERENCES

[1] A. Fariha, A. Tiwari, A. Radhakrishna, S. Gulwani, and A. Meliou. Data invariants: On trust in data-driven systems. *CoRR*, abs/2003.01289, 2020. http://arxiv.org/abs/2003.01289.

[2] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach: Part 1: Causes. In *UAI*, pages 194–202, 2001.

[3] 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.

[4] 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.

[5] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. WHY so? or WHY no? functional causality for explaining query answers. In *MUD@ VLDB*, pages 3–17, 2010.

[6] Z. Miao, Q. Zeng, C. Li, B. Glavic, O. Kennedy, and S. Roy. CAPE: explaining outliers by counterbalancing. *PVLDB*, 12(12):1806–1809, 2019.

[7] B. Micenková, R. T. Ng, X. Dang, and I. Assent. Explaining outliers by subspace separability. In *ICDM*, pages 518–527, 2013.

[8] A. A. Qahtan, N. Tang, M. Ouzzani, Y. Cao, and M. Stonebraker. ANMAT: automatic knowledge discovery and error detection through pattern functional dependencies. In *SIGMOD*, pages 1977–1980, 2019.

[9] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset shift in machine learning.* The MIT Press, 2009.

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

[11] M. Tveten. Which principal components are most sensitive to distributional changes? *CoRR*, abs/1905.06318, 2019.

[12] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.

[2] Mobile prices: kaggle.com/iabhishekofficial/mobile-price-classification

[3] House prices: kaggle.com/c/house-prices-advanced-regression-techniques