Outliers: The Good, the Bad and the Ugly

Paper ID: 71

ABSTRACT

This paper studies the impact of outliers in relational dataset $\mathcal{D}$ on the accuracy of ML classifiers $\mathcal{M}$, when $\mathcal{D}$ is used to train and evaluate $\mathcal{M}$. Outliers are data points that statistically deviate from the distribution of the majority. We distinguish good outliers, *i.e.,* novel data, from bad ones, *i.e.,* those introduced by errors. Moreover, we separate ugly ones in influential features from the other bad ones. We find that only the ugly ones have negative impact on $\mathcal{M}$, while the good (resp. the other bad) ones have positive (resp. neglectable) impact. To mitigate the negative impact, we propose a class of rules, denoted by OMRs, to identify ugly outliers by embedding ML outlier detectors and statistical functions as predicates. We develop algorithms to (a) learn OMRs from real-life data, and (b) catch and fix ugly outliers using the learned OMRs, instead of removing tuples. Using real-life data, we empirically show that OMRs improve the accuracy of various classifiers by 6.9% on average, up to 21.6%.

1 INTRODUCTION

Outliers are data points that statistically deviate from the distribution of the majority [21]. Outliers are common [6, 15, 103], *e.g.,* in the ADBench benchmark [70], outliers range from 0.03% [127] to 39.91% [27] in the 57 real-world datasets collected. In light of these, there has been a host of work on outlier detection, based on either ML models [17, 23, 29, 35, 39, 67, 70, 73, 74, 74, 98, 99, 114, 117, 129, 135, 151, 170] or logic rules [10, 16, 34, 44, 125, 155].

This paper studies the impact of outliers on machine learning (ML). Consider a dataset $\mathcal{D}$ for training and evaluating an ML classifier $\mathcal{M}$. We aim to understand what outliers in $\mathcal{D}$ have substantial impacts on the accuracy of $\mathcal{M}$. From our experiments, we find the following. (1) There are "good outliers" that arise from novel data, *i.e.,* data of new distributions that are not included in $\mathcal{D}$. We certainly want to keep such outliers in our training data to make $\mathcal{M}$ capable of recognizing new objects. (2) Outliers may be "bad" as they are introduced by errors. However, we should not delete such outliers in order not to change the data distribution of $\mathcal{D}$, lose important associations between variables [30] and thus reduce the accuracy of $\mathcal{M}$. (3) We need to separate "ugly" ones, *i.e.,* those that result in wrong predictions of model $\mathcal{M}$, from the other "bad" ones. Outliers can distort the decision boundary or change the decision rules of $\mathcal{M}$, especially when the data distribution is uneven or the number of outliers is large, leading to incorrect classifications [6, 83, 87, 171, 172]; *e.g.,* an increase of 4% of outliers in ADBench yields an 11.85% reduction in the accuracy of supervised classifiers [70]. It is the ugly outliers that we need to detect and fix.

Example 1: As shown in Table 1, a real-life relation Student (Age, Gender, FamilySupport, StudyTimeWeekly, Absence, GPA, GradeClass) is used for grade classification [88]. Attribute "GradeClass" records classification labels, including A (GPA>3.5), B (3.0<GPA<=3.5), C (2.5<GPA<=3.0), D (2.0<GPA<=2.5) and F (GPA<=2.0). FamilySupport ranges from 0 to 4, where higher values indicate greater support. Generally, more family support, longer study time per weak, and fewer absences are associated with higher GPA and better grades. Selecting FamilySupport, StudyTimeWeekly, Absence and GPA as important features during training, a well-trained ML classifier $\mathcal{M}$ (*e.g.,* XGBoost [31]) classifies tuples t_1, t_2, t_4 and t_8 as B, F, A and D, respectively. Among these, tuples t_1 and t_2 are misclassified while t_4 and t_8 are classified correctly by $\mathcal{M}$.

tid	Age	Gender	Family Support	StudyTime Weekly	Absence	GPA	Grade Class
t_1	15	Male	2	27.5	24	0.86	B
t_2	18	Female	3	18.9	1	3.61	F
t_3	17	Male	1	11.9	11	2.15	D
t_4	16	Male	3	18.4	0	4.00	A
t_5	15	Female	0	14.4	7	3.57	C
t_6	15	Male	2	4.2	26	0.11	F
t_7	17	Female	2	19.3	5	3.16	B
t_8	27	?	3	10.0	14	2.05	D
t_9	18	Male	3	18.7	10	2.85	C

Table 1: Example Student **relation**

Observe the following outliers (underlined) in Student.

(1) Tuple t_1 has moderate FamilySupport and relatively long study time, but its Absence is 24 and GPA is less than 2; it should obviously be classified as F. Thus, if the Absence of t_1 is correct, then its StudyTimeWeekly is an outlier as it is inconsistent with other features. Since this StudyTimeWeekly outlier appears in important features of $\mathcal{M}$, $\mathcal{M}$ misclassifies t_1 as B. This shows that outliers in critical features have significant impacts on classification accuracy. This is an "ugly" outlier, for its negative impact on the accuracy of $\mathcal{M}$.

(2) Tuple t_2 has the worst grade that contradicts with other features of high family support and good study habits. Thus, the grade F of t_2 is an outlier, which should be *A* instead. That is, not only attributes can be "ugly" outliers, but also classification labels.

(3) Tuple t_4 records a boy with good study habits and high family support. However, it can still be regraded as an outlier by supervised detectors since its label A is rare among all tuples. The presence of novel grade A in t_4 helps $\mathcal{M}$ better learn the distribution of grade A students. In other words, the grade A of t_4 is a "good" outlier.

(4) Attributes Age and Gender of t_8 are outliers as they significantly deviate from the normal domain. Since these outliers appear in features that were irrelevant to classification, $\mathcal{M}$ can correctly classify t_8 as D. That is, these outliers are "bad" but not "ugly". □

This example raises the following questions. How can we accurately detect different types of outliers? What should we do to fix outliers that sabotage the classification accuracy? Can we find right values to replace or repair ugly outliers in order to reserve data distribution, variable associations and the accuracy of $\mathcal{M}$?

Contributions & organization. In response to these questions, we propose a framework, named Outlier HUNTer (OHunt), which detects and fixes outliers in relational training data to improve the accuracy of ML classifiers. OHunt has the following features.

(1) Rules (Section 2). OHunt unifies logic reasoning and ML predictions by proposing a class of Outlier Management Rules of the form $X \to p_0$, denoted by OMRs. As opposed to previous data quality rules [11, 49, 50, 55], the precondition X of an OMR is a combination

of (a) logic predicates, (b) ML models for identifying outliers, and (c) statistical and loss functions to provide insights of downstream ML models. Using the functions and ML models, OMRs characterize different types of outliers. Moreover, they are able to reduce false positives (FPs) and false negatives (FNs) of ML detectors for outliers by incorporating logic conditions, and explain ML predictions in logic.

(2) Framework (Section 3). Based on OMRs, OHunt takes as input an ML classifier $\mathcal{M}$ and a dataset $\mathcal{D}$ for training $\mathcal{M}$. It first learns a set Σ of OMRs from (samples of) $\mathcal{D}$. It then catches and fixes ugly outliers in $\mathcal{D}$ with Σ, with accuracy guarantees (see below). It produces a dataset $\mathcal{D}_c$ in which ugly outliers are fixed, which improves the classification accuracy of $\mathcal{M}$ when trained with $\mathcal{D}_c$.

(3) Rule discovery (Section 4). We propose an approach for learning OMRs from real-life data, which selects ML, function and logic predicates in OMRs guided by the classification accuracy of the given model $\mathcal{M}$. We formulate the discovery of OMRs as a meta-learning problem and adopt a Bayesian-based optimization algorithm to identify valuable OMRs. This approach avoids the exponential enumeration of OMR predicates, selects hyperparameters through Bayesian optimization, and employs the accuracy of $\mathcal{M}$ as the criterion to identify effective OMRs for detecting or fixing outliers, instead of traditional metrics such as support and confidence [65, 81, 118].

(4) Detecting and fixing outliers (Section 5). OHunt not only distinguishes ugly outliers from the others, but also fixes the detected outliers. As opposed to simply removing tuples that contain outliers [24, 47, 90, 102, 138], OHunt deduces the right values by a combination of strategies, such as (a) logic reasoning with ground truth, (b) data distribution with ML models, and (c) referencing external data sources. It chases the training data $\mathcal{D}$ with the set Σ of OMRs learned, and accumulates ground truth Γ in the recursive process. It guarantees that the chase converges at the same result no matter what rules in Σ are applied and in what order they apply. Moreover, the fixes generated are logical consequences of the rules in Σ, the ground truth, and predictions of embedded ML predicates. Thus the fixes are correct as long as Σ, Γ and ML predicates are accurate.

(5) Experimental Study (Section 6). Using five classifiers and 38 real-world and synthetic datasets, we empirically find the following. (a) OHunt is accurate in detecting ugly outliers. Its average F1-score is above 0.960, which is 73.6% higher than the baselines, up to 78.4%. (b) OHunt effectively improves the accuracy of classifiers $\mathcal{M}$ by fixing ugly outliers, with an average enhancement of 6.9%, up to 21.6% on the real datasets. (c) OHunt is robust to various types of noise. It improves the F1-score of $\mathcal{M}$ by an average of 13.2% when injecting different types of noises, and up to 17.5% over the baselines on 20 noisy datasets. (d) OHunt is scalable. Its runtime and maximum memory consumption increase almost linearly as the parameters of datasets and Bayesian optimization grow.

Novelty. The novelty of OHunt includes (1) a categorization of outliers into the good, the bad and the ugly, distinguishing ugly ones that sabotage ML classifiers; (2) rules to detect different types of outliers by embedding ML models and statistical/loss functions as predicates, to improve the accuracy and explainability, (3) an approach for rule discovery based on meta learning, and (4) a method to fix outliers with right values, instead of simply removing tuples.

We discuss related work in Section 7 and future work in Section 8.

2 OUTLIER MANAGEMENT RULES

This section introduces Outlier Management Rules (OMRs). We start with preliminaries (Section 2.1), and then define OMRs (Section 2.2). We also show how OMRs distinguish different types of outliers and how they improve and explain ML predictions (Section 2.3).

2.1 Preliminaries

We start with basic notations and concepts.

Datasets. Consider a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, where R_i is a relation schema $R(A_1, \ldots, A_k, Y)$ and each A_i is an attribute (*a.k.a.* feature). A dataset $\mathcal{D}$ of $\mathcal{R}$ is $(D_1, \ldots, D_m)$, where D_i is a relation of R_i. Each tuple in D_i has a categorical label Y, for supervised classification model training. The active domain of attribute A_i in D_j, denoted as $dom(A_i)$, refers to discrete values or continuous ranges of A_i that are valid inputs for the ML classifier.

Accuracy. Following the closed-world assumption [72], we denote by $\text{acc}(\mathcal{M}, \mathcal{D})$ the accuracy of an ML classifier $\mathcal{M}$ that is trained and evaluated with splits of a dataset $\mathcal{D}$. We define $\text{acc}(\mathcal{M}, \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(\hat{y_i} = y_i)$, where $\mathcal{D}$ contains n samples s_i ($i \in [1, n]$), each s_i consists of a true label y_i and features X_i, $\hat{y_i}$ is the predicted label of s_i, and $\mathbb{I}(\cdot)$ is the function that returns 1 if true and 0 otherwise.

Influential features. We say that attribute A_i is *influential* if $\text{acc}(\mathcal{M}, \mathcal{D}) - \text{acc}(\mathcal{M}, \mathcal{D}\backslash A_i) > \alpha$, where α is a threshold, *i.e.,* A_i significantly affects the prediction accuracy of $\mathcal{M}$ on dataset $\mathcal{D}$.

Imbalanced features. We say that a feature/attribute A_j is *imbalanced* if the proportion of the most common value of A_j attribute exceeds that of the least common one by a given threshold β.

2.2 OMRs with ML and Function Predicates

The objective of OMRs is twofold: (a) to detect outliers in $\mathcal{D}$ and classify them into good, bad, and ugly based on their impact on $\mathcal{M}$, as indicated in Example 1; and (b) to fix ugly outliers with right values. Below we first define the predicates of OMRs.

Predicates. Over a database schema $\mathcal{R}$, a dataset $\mathcal{D}$ of $\mathcal{R}$, and an ML classifier $\mathcal{M}$, the predicates of OMRs have the following forms:

$$\begin{aligned} p \quad ::= \quad & R(t) \mid t.A \otimes c \mid t.A \otimes s.B \mid \mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M}) = \tau \mid \\ & F(t, A, \mathcal{D}, \mathcal{M}) = \tau \mid \text{Rk}(t.A), \end{aligned}$$

where $\otimes$ is one of $=, \neq, <, \leq, >, \geq$, and τ is true or false. We write $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M}) = \text{true}$ simply as $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$ when it is clear in the context; similarly for $F(t, A, \mathcal{D}, \mathcal{M}) = \text{true}$. We refer to $R(t)$, $t.A \otimes c$, $t.A \otimes s.B$ as *logic predicates.* Following tuple relational calculus [5], (1) $R(t)$ is a *relation atom* over $\mathcal{R}$, where $R \in \mathcal{R}$, and t is called a tuple variable *bounded by* $R(t)$. (2) When t is bounded by $R(t)$ and A is an attribute of R, $t.A$ denotes the A-attribute of t. (3) In $t.A \otimes c$, c is a constant in $dom(A)$. (4) In $t.A \otimes s.B$, A and B have the same type. (5) Rk is an indicator for good, bad and ugly.

In addition to logic predicates as in other data quality rules, OMRs support *ML predicates* $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$ and *function predicates* $F(t, A, \mathcal{D}, \mathcal{M})$, which are detailed as follows.

ML predicates. We consider two types of ML predicates $\mathcal{M}_o$.

(1) Detectors. OMRs leverage pre-trained ML-based detectors to identify outliers and influential features, such as the following:

- $\mathcal{M}_{\text{DetO}}(t, A, \mathcal{D})$, an existing ML classifier that returns true if $t.A$ is an outlier, and false otherwise, *e.g.,* unsupervised [18, 101, 114, 124, 141, 165] and label-based (semi-supervised, supervised and rule-based) [31, 34, 86, 116, 117, 123, 125, 133, 164] detectors.

- $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$, an ML classifier for determining whether the A-attribute of relation schema R is an influential feature for the given ML classifier $\mathcal{M}$. It returns true if so, and false otherwise.

(2) Fixes to outliers. OMRs also employ ML models to determine fixes to ugly outliers via distribution analysis and data extraction.

- $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$, where $t.A$ is an ugly outlier (either attribute or label), and v is a candidate value for $t.A$. It returns true if v is a fix to $t.A$ with a high confidence, where v is determined by, *e.g.,* a regression model that can learn attribute distribution, association analysis of attribute $t.A$ and other attributes of tuple t, or a classification model for labels and categorical features.
- $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$, where $t.A$ and v are the same as above. It returns true if v is a fix to $t.A$ with a high confidence, where v is extracted from an external data source KB.

The ML and statistical models learn column distributions and attribute association from the normal data, predict values or extract correct external data for the replacement of ugly outliers.

We will see (a) how the embedded $\mathcal{M}_o$ helps us detect ugly outliers, and (b) how OMRs reduce FPs and FNs of $\mathcal{M}_o$ in Section 2.3.

Function predicates. OMRs may embed statistical and loss functions F as predicates, to determine whether $t.A$ is an outlier, whether $t.A$ triggers substantial loss, or whether a dataset is imbalanced.

- $\mathsf{outlier}(D, R, t, A, \theta)$, where R is a relation schema in $\mathcal{R}$, D is an instance of R, A is an attribute of R, t is a tuple of D, and θ is a hyperparameter. It returns true if the value $t.A$ differs from $s.A$ for at least a factor θ of all tuples s in D, and false otherwise.
- $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$, where $\mathcal{D}$ is a dataset of schema $\mathcal{R}$, $\mathcal{M}$ is the classifier to be trained with $\mathcal{D}$, t is a tuple of $\mathcal{D}$, and A is an attribute of t. It is exactly the loss function of $\mathcal{M}$, *e.g.,* the hinge loss function [20] for SVM classifiers. We use $\mathsf{loss}(\mathcal{M}, D, t, A) \leq \lambda$ and $\mathsf{loss}(\mathcal{M}, D, t, A) > \lambda$ to check whether the loss is below a threshold λ or not. Intuitively, ugly outliers can hardly reduce their loss values during the model training, while the loss of good and other bad ones usually converge to small values eventually. Indeed, good outliers typically reduce the loss on normal samples from a higher value above λ at early training stage to a lower value below λ by the end, thereby enhancing the accuracy of $\mathcal{M}$. In contrast, the loss from the other outliers on normal samples remains relatively stable throughout the training process of $\mathcal{M}$.
- $\mathsf{imbalanced}(D, R, t, A, \delta)$, where R, D, A, t are as above, and δ is a configurable parameter. It returns true if the number of tuples in D grouped by $t.A$ is smaller than the counts of the other groups by A values, by at least a factor δ, and false otherwise.

We show how these functions are implemented in Section A.

Outlier Management Rules. For a database schema $\mathcal{R}$, a dataset $\mathcal{D}$ of $\mathcal{R}$, and an ML classifier $\mathcal{M}$, an OMR φ over $\mathcal{R}$ has the form of

$$\varphi = X \to p_0,$$

where X is a conjunction of predicates over $\mathcal{R}$, $\mathcal{M}$ and $\mathcal{D}$, and p_0 is a predicate such that all its tuple variables are bounded in X. We refer to X and p_0 as the *precondition* and *consequence* of φ, respectively.

We consider OMRs for a given ML classifier $\mathcal{M}$, and dataset $\mathcal{D}$ of schema $\mathcal{R}$ for training $\mathcal{M}$. We will see real-life OMRs in Section 2.3.

Semantics. Consider an instance $\mathcal{D}$ of database schema $\mathcal{R}$. A *valuation* of tuple variables of an OMR $\varphi = X \to p_0$ in $\mathcal{D}$, or simply *a valuation of* φ, is a mapping h that maps each t in relation atom $R(t)$ of φ to a tuple in the relation of schema R in $\mathcal{D}$.

We say that h satisfies a predicate p, denoted as $h \models p$, if h satisfies the following conditions. (1) If p is a relation atom $R(t)$, $t.A \otimes c$ or $t.A \otimes s.B$, then $h \models p$ is interpreted as in tuple relational calculus [5]. (2) If p is an ML predicate $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$, then $h \models p$ if $\mathcal{M}_o$ predicts true on tuple $t \in \mathcal{D}$ and its attribute A *w.r.t.* the given ML classifier $\mathcal{M}$; similarly for $p = F(t, A, \mathcal{D}, \mathcal{M})$.

For a set X of predicates p, we write $h \models X$ if $h \models p$ for all predicates $p \in X$. For an OMR $\varphi = X \to p_0$, we write $h \models \varphi$ such that if $h \models X$, then $h \models p_0$. An instance $\mathcal{D}$ of $\mathcal{R}$ satisfies φ if $h \models \varphi$ for all valuations h of φ in $\mathcal{D}$, denoted by $\mathcal{D} \models \varphi$. We say that $\mathcal{D}$ satisfies a set Σ of φ, written by $\mathcal{D} \models \Sigma$, if $\mathcal{D} \models \varphi$ for all $\varphi \in \Sigma$.

2.3 Catching and Fixing Outliers with OMRs

We showcase OMRs learned from real-world datasets Student [88] and Apple [45], and illustrate how they can detect (ugly) outliers.

(1) Catching outliers. We use OMRs of the form $X \to \mathsf{Rk(t.A)}$ to classify outlier $t.A$ as good, bad or ugly. Below are some examples.

(a) $\varphi_1 = \mathcal{M}_{\mathsf{DetO}}(t, \mathsf{Quality}, \mathsf{Apple}) \wedge \mathsf{outlier}(\mathsf{Apple}, R, t, \mathsf{Size}, 0.30) \wedge \mathsf{outlier}(\mathsf{Apple}, R, t, \mathsf{Sweetness}, 0.76) \wedge \mathsf{loss}(\mathcal{M}, \mathsf{Apple}, t, \mathsf{Quality}) \leq 0.52 \wedge \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{Size}, \mathcal{M}) \wedge \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{Sweetness}, \mathcal{M}) \wedge \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{Quality}, \mathcal{M}) \to \mathsf{good}(t.\mathsf{Quality})$. Specifically, in the Apple dataset, relation schema R is *Apple* and $\mathsf{loss}(\cdot)$ is the hinge loss function. It says that while t.Quality is an outlier (by $\mathcal{M}_{\mathsf{DetO}}(\cdot)$), it has positive impact on the SVM classifier $\mathcal{M}$. Intuitively, t.Size and t.Sweetness are identified as feature outliers by $\mathsf{outlier}(\cdot)$ and influential by $\mathcal{M}_{\mathsf{DetI}}(\cdot)$; moreover, they effectively contribute to the training of $\mathcal{M}$; as a consequence, when training $\mathcal{M}$, $\mathsf{loss}(\mathcal{M}, \mathsf{Apple}, t, \mathsf{Quality})$ can be reduced below threshold $\lambda = 0.52$ on the label t.Quality. Hence tuple t with the feature and label outliers benefit model $\mathcal{M}$.

(b) $\varphi_2 = \mathsf{outlier}(\mathsf{Apple}, R, t, \mathsf{Ripeness}, 0.10) \wedge \mathsf{loss}(\mathcal{M}, \mathsf{Apple}, t, \mathsf{Quality}) \leq 0.52 \wedge \mathsf{imbalanced}(\mathsf{Apple}, R, t, \mathsf{Ripeness}, 0.30) \wedge \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{Ripeness}, \mathcal{M}) = \mathsf{false} \to \mathsf{bad}(t.\mathsf{Ripeness})$. Here φ_2 says that t.Ripeness is an outlier, but it does not harm the training of $\mathcal{M}$ since $\mathsf{loss}(\cdot) \leq 0.52$. Intuitively, outlier t.Ripeness is not influential (by $\mathcal{M}_{\mathsf{DetI}}(\cdot) = \mathsf{false}$); it is an occasional erroneous value since both $\mathsf{outlier}(\cdot)$ and $\mathsf{imbalanced}(\cdot)$ are true. Hence it is bad but not ugly.

(c) φ_3 is $\mathsf{outlier}(\mathsf{Apple}, R, t, \mathsf{Size}, 0.30) = \mathsf{false} \wedge \mathsf{outlier}(\mathsf{Apple}, R, t, \mathsf{Sweetness}, 0.76) = \mathsf{false} \wedge \mathsf{outlier}(\mathsf{Apple}, R, t, \mathsf{Juiciness}, 0.16) \wedge \mathsf{imbalanced}(\mathsf{Apple}, R, t, \mathsf{Juiciness}, 0.2) \wedge \mathsf{loss}(\mathcal{M}, \mathsf{Apple}, t, \mathsf{Quality}) > 0.52 \wedge \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{Size}, \mathcal{M}) \wedge \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{Sweetness}, \mathcal{M}) \wedge \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{Juiceness}, \mathcal{M}) \to \mathsf{ugly}(t.\mathsf{Juiciness})$. It says that t.Juiceness is an ugly outlier, since it is an outlier, influential and imbalanced; moreover, the loss of $\mathcal{M}$ on tuple t constantly exceeds threshold $\lambda = 0.52$, and $\mathcal{M}$ cannot converge on t. In addition, other influential features t.Size and t.Sweetness are not outliers (by $\mathsf{outlier}(\cdot) = \mathsf{false}$).

(2) Fixing ugly outliers. We use OMRs of the forms $X \to t.A = c$ or $X \to t.A = s.B$ to fix ugly outlier $t.A$ with value c or $s.B$, which are determined by means of ML predicates $\mathcal{M}_{\mathsf{FixA}}$ or $\mathcal{M}_{\mathsf{FixE}}$, and logic reasoning in precondition X. We defer the details to Section 5.

(3) Improving existing ML detectors. We use OMRs of the form

$\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M}) \wedge X_1 \rightarrow p_0$ to improve the accuracy of ML model $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$ by incorporating logic conditions X_1. That is, while $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$ predicts true (resp. false), φ override the decision if condition X_1 is satisfied (resp. violated), to reduce FPs (resp. FNs) of $\mathcal{M}_o$. Here $\mathcal{M}_o$ can be $\mathcal{M}_{\mathsf{DetO}}$, $\mathcal{M}_{\mathsf{DetI}}$, $\mathcal{M}_{\mathsf{FixA}}$ or $\mathcal{M}_{\mathsf{FixE}}$.

Consider OMR φ_4 from Student: $\mathcal{M}_{\mathsf{DetI}}(R, \mathsf{FamilySupport}, \mathcal{M}) = \mathsf{false} \wedge X_1 \rightarrow \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{FamilySupport}, \mathcal{M}) = \mathsf{true}$, where X_1 is $t.\mathsf{Tutoring} = 1 \wedge t.\mathsf{ParentalEducation} \geq 1$. While ML detector $\mathcal{M}_{\mathsf{DetI}}(\cdot)$ classifies FamilySupport as not influential, if tuple t satisfies X_1, the prediction should be flipped. Indeed, if the parents are educated ($t.\mathsf{ParentalEducation} \geq 1$) and they spend time on supplemental education ($\mathcal{M}_{\mathsf{DetI}}(\cdot)$), then the student's performance can be improved if being tutored at home ($t.\mathsf{Tutoring} = 1$).

(4) Explaining ML predictions. We use OMRs of the form $X \rightarrow \mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ to explain predictions of ML detector $\mathcal{M}_{\mathsf{DetO}}$, by discovering precondition X to explain why $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ holds.

Consider OMR φ_5 learned from Apple: $\mathsf{outlier}(\mathsf{Apple}, \mathcal{R}, t, \mathsf{Juiciness}, 0.16) \wedge \mathsf{imbalanced}(\mathsf{Apple}, R, t, \mathsf{Juiciness}, 0.2) \wedge \mathcal{M}_{\mathsf{DetI}}(R, \mathsf{Juiceness}, \mathcal{M}) \rightarrow \mathcal{M}_{\mathsf{DetO}}(t, \mathsf{Juiceness}, \mathcal{D})$. The OMR explains why the ML detector $\mathcal{M}_{\mathsf{DetO}}(\cdot)$ identifies $t.\mathsf{Juiciness}$ as an outlier. It is because (a) the value of $t.\mathsf{Juiciness}$ differs significantly from the other tuples ($\mathsf{outlier}(\mathsf{Apple}, \mathcal{R}, t, \mathsf{Juiciness}, 0.16)$), (b) Juiciness is influential for $\mathcal{M}$ by $\mathcal{M}_{\mathsf{DetI}}(\cdot)$, and (c) $t.\mathsf{Juiciness}$ makes the dataset imbalanced ($\mathsf{imbalanced}(\mathsf{Apple}, R, t, \mathsf{Juiciness}, 0.2)$).

Remark. These OMRs cannot be expressed as previous data quality rules such as denial constraints (DCs) [11], matching dependencies (MDs) [49] and entity enhancing rules (REEs) [53, 55], since (1) DCs and MDs do not support ML models like $\mathcal{M}_{\mathsf{DetO}}(t, \mathsf{Quality}, \mathcal{D})$; and (2) REEs do not support functions like $\mathsf{loss}(\mathcal{M}, D, t) \otimes \lambda$. As opposed to prior rules, OMRs target outliers to improve the accuracy of a given ML classifier; they embed ML-based outlier detectors, and statistical/loss functions as predicates, to capture insights of downstream ML models. In light of these, OMRs require discovery methods different from discovery of the prior rules (see Section 4).

3 OHUNT: OUTLIER HUNTER

This section outlines OHunt, our system for detecting and fixing ugly outliers. OHunt works as follows. Given a database schema $\mathcal{R}$, a dataset $\mathcal{D}$ of schema $\mathcal{R}$, and an ML classifier $\mathcal{M}$ to be trained with $\mathcal{D}$, it first learns a set Σ of OMRs from (samples of) $\mathcal{D}$, guided by the given model $\mathcal{M}$. It then recursively applies the OMRs of Σ to $\mathcal{D}$, to catch and fix ugly outliers in place. OHunt returns the improved dataset as new training data for $\mathcal{M}$. It has two main modules.

(1) Rule discovery. This module targets the following problem.

- *Input*: $\mathcal{R}, \mathcal{D}, \mathcal{M}$ as above, a set $\mathcal{P}$ of potential OMR predicates, objective function $f_d(\cdot)$ (resp. $f_f(\cdot)$) for detection accuracy (resp. accuracy improvement of $\mathcal{M}$; see Section 4 for $\mathcal{P}, f_d(\cdot), f_f(\cdot)$).
- *Output:* A set $\Sigma = \Sigma_d \cup \Sigma_f$ of OMRs, where Σ_d is the set of OMRs mined for detecting (ugly) outliers, and Σ_f is the set of OMRs mined for fixing ugly outliers detected by Σ_d.

We develop an algorithm, denoted by OLeaner, for learning Σ in Section 4. It has the the following properties. (1) OLeaner models OMR discovery as a meta-learning problem, with accuracy improvement as the objective function. This avoids the exponential search space of levelwise rule discovery. (2) We employ Bayesian optimization to implement the meta-learning process, and identify hyperparameter combinations that genuinely improve the detection accuracy of ugly outliers and the classification accuracy of model $\mathcal{M}$.

(2) Outlier correction. The module focuses on the problem below.

- *Input*: $\mathcal{R}, \mathcal{D}, \mathcal{M}$ as above, and the set Σ of OMRs learned.
- *Output*: Dataset $\mathcal{D}_c$ by fixing ugly outliers in $\mathcal{D}$ with Σ.

It catches ugly outliers in $\mathcal{D}$ by applying OMRs of Σ_d in Σ, and fixes the outliers with right values with the OMRs of Σ_f in Σ. More specifically, it chases $\mathcal{D}$ with Σ_f [134], by accumulating a set Γ of ground truth and referencing Γ in the process. It guarantees that the fixes to outliers are correct as long as Σ_f and Γ are correct, and the ML and function predicates embedded in the OMRs are accurate. We present the algorithm for the module in Section 5.

4 DISCOVERING OMRS BY META LEARNING

This section develops algorithm OLeaner for learning OMRs.

Challenges. Rules are traditionally mined by levelwise enumeration [51, 65, 81, 118, 142]. However, the classical approach does not suffice for OMRs. (1) It cannot tell how to select hyperparameters in predicates, *e.g.,* how to find an optimal θ in $\mathsf{outlier}(D, R, t, A, \theta)$. Worse yet, hyperparameters that are continuous values can hardly be determined by enumeration. (2) The quality of OMRs is measured by their effectiveness in identifying good outliers that benefit $\mathcal{M}$, detecting ugly outliers that harm $\mathcal{M}$, and correcting ugly outliers for higher classification accuracy. Hence, classical rule quality metrics such as support and confidence [48, 53, 54] become invalid. For example, although an OMR for catching rare outliers has a low support, it may improve the accuracy of $\mathcal{M}$ and should be preserved. (3) Classical rule mining aims to find the entire set of rules above support/confidence thresholds, and conducts exponential enumeration. In contrast, we only need OMRs that improve the detection accuracy and classification accuracy of $\mathcal{M}$.

In view of these challenges, we formulate the discovery of OMRs as a meta learning problem and employ Bayesian optimization to efficiently discover valuable OMRs. We first briefly review basics of model-evaluation-based meta learning (Section 4.1). Then we formulate OMR discovery from the meta learning perspective, based on which we develop OLeaner for OMR discovery (Section 4.2).

4.1 Meta Learning based on Model Evaluation

Meta-learning involves learning new tasks by observing how ML models perform on existing ones [150]. It takes as input (a) a set of known tasks $t_j \in T$, (b) a set of learning algorithms with *configurations* $\theta_i \in \Theta$, where Θ covers hyperparameter settings like learning pipeline or neural network components, (c) the set $\mathbf{P}$ of all prior scalar evaluations $P_{i,j} = P(\theta_i, t_j)$ of configuration θ_i on task t_j measured by a predefined measure (*e.g.,* accuracy) and model evaluation technique (*e.g.,* cross validation), and (d) the set $\mathbf{P}_{\text{new}}$ of evaluations $P_{i,\text{new}}$ on a new task t_{new}. Meta-learning trains a *meta-learner* L to predict recommended configurations Θ^*_{new} for a new task t_{new} based on the *meta-data* $\mathbf{P} \cup \mathbf{P}_{\text{new}}$ [150].

Various meta-learning methods accelerate and improve hyperparameter selection of ML models in a data-driven way, classified as ranking-based [3, 22, 38, 97, 97], configuration space design [82, 122, 158], and configuration transfer [58, 63, 66, 143, 146, 159].

4.2 OMR Discovery as Meta Learning Problem

As shown in Algorithm 1, given a consequence predicate p_0 (Section 2), OLeaner for OMRs discovery is to find a combination of predicates along with their hyperparameters for precondition X, such that each $\varphi = X \to p_0$ either detects or fixes outliers.

Formulation. The discovery of OMR φ can be formulated as a meta learning problem as follows. The tasks $t_j \in T$ are either distinguishing outliers or fixing ugly ones in dataset $\mathcal{D}$ (line 1). The learning algorithms in meta-learning can be parametrized as: $\mathbb{S}(p_1, x_1) \wedge \mathbb{S}(p_2, x_2) \wedge \ldots \wedge \mathbb{S}(p_n, x_n) \to p_0$, where $p_1, \ldots, p_n$ are predicates of OMRs, and $\mathbb{S}(p, x)$ is a selection function defined as:

$$\mathbb{S}(p, x) = \begin{cases} p, & x = 1 \\ \mathsf{true}, & x = 0 \end{cases}$$

Thus, the configurations $\theta_i \in \Theta$ of a learning algorithm include the set $\{x_1, x_2, \ldots, x_n\}$ that controls whether each predicate is selected in precondition X, and hyperparameters in predicates, *e.g.,* the threshold λ in $\mathsf{loss}(\mathcal{M}, D, t, A) \leq \lambda$. The set $\mathbf{P}$ of all prior scalar evaluations $P_{i,j} = P(\theta_i, t_j)$ of configuration θ_i on task t_j is measured by the accuracy $f_d(\cdot)$ of outlier detection or the improvements $f_f(\cdot)$ of downstream classifiers (line 2). Based on this formulation, OLeaner trains a meta-learner L that recommends configurations Θ^* for tasks $t_j \in T$ based on the evaluation $\mathbf{P}$ (initialized as $\emptyset$). The predicted Θ^* specifies which predicates are selected in precondition X, and what values should be assigned to hyperparameters of those selected predicates, to make a useful OMR $X \to p_0$.

Objective functions $f_d(\cdot)$ and $f_f(\cdot)$ are defined as follows. (1) For $f_d(\cdot)$, we first train classifier $\mathcal{M}$ on the input dataset $\mathcal{D}$, and find the set U_1 of potential ugly outliers in $\mathcal{D}$, which consists of those samples whose loss function values exceed a threshold λ, where λ can be initialized as, *e.g.,* 1 under hinge loss, and learned during the optimization process. For an OMR φ_1 that detects ugly outliers, we find the set U_2 of ugly outliers deduced by φ_1. Then $f_d(\varphi_1) = \frac{TP+TN}{|\mathcal{D}|}$, where $TP = |U_1 \cap U_2|$ for true positives is the number of samples detected as ugly outliers by φ_1 that are real ugly outliers of $\mathcal{D}$, and $TN = |\mathcal{D}| - |U_1 \cap U_2| - |U_1 \setminus U_2| - |U_2 \setminus U_1|$ for true negatives is the number of samples caught as non-ugly samples that are real non-ugly samples. (2) To define $f_f(\cdot)$ of an OMR φ_2 for fixing ugly outliers, we first employ φ_2 to fix ugly outliers of $\mathcal{D}$, and obtain a cleaned dataset $\mathcal{D}_\varphi$. Next, we retrain $\mathcal{M}$ on $\mathcal{D}_\varphi$ and calculate the $\mathsf{acc}(\mathcal{M}, \mathcal{D}_\varphi)$ (Section 2.1). Then $f_f(\varphi_2) = \mathsf{acc}(\mathcal{M}, \mathcal{D}_\varphi) - \mathsf{acc}(\mathcal{M}, \mathcal{D})$. Guided by $f_d(\cdot)$ or $f_f(\cdot)$ on their corresponding task t_j, the well-trained meta-learner L produces optimized configurations Θ^*, where each $\theta_i \in \Theta^*$ specifies the selected predicates along with their hyperparameters for OMRs.

Algorithm. We employ Bayesian optimization [9] as the meta learning algorithm for OMRs, which initializes hyperparameter configurations $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$ by domain experts or randomly initialized (line 3). Based on Θ, it evaluates the target metric (*e.g.,* detection accuracy) on each task t_j (*e.g.,* detecting ugly outliers), and initializes the sets $\mathbf{P}$, Σ, Σ_d and Σ_f as empty sets (line 4).

Then the optimization for t_j iterates as follows. A surrogate model is employed to model the distribution of target metric values based on $\mathbf{P}$, which facilitates the Bayesian optimization to determine the next round hyperparameter configuration θ_{m+1} by selecting the point that maximizes the acquisition function [62, 139]. This further updates both $\mathbf{P} = \mathbf{P} \cup \{P(\theta_{m+1}, t_j)\}$ and the distribution of surrogate model guided by the target metric values. This iteration returns its optimal hyperparameter configuration Θ^* that achieves the highest target metric on task t_j, when the budget limit (*e.g.,* number of evaluations, runtime) is reached (lines 5-10).

Algorithm 1: Algorithm OLeaner

```
Input: $R$, $\mathcal{D}$, $\mathcal{M}$, $\mathcal{P}$, $f_d(\cdot)$ and $f_f(\cdot)$ as stated in Section 3.
Output: A set of detecting and fixing OMRs $\Sigma = \Sigma_d \cup \Sigma_f$
1  Define tasks $T$ as outlier detection and fixing tasks;
2  Define evaluation metric as detection accuracy $f_d(\cdot)$ for outlier
     detection or classification improvement $f_f(\cdot)$ for fixing;
3  Initialize configurations $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$ for hyperparameters;
4  Initialization of evaluations and OMR sets: $\mathbf{P} \leftarrow \emptyset$; $\Sigma, \Sigma_f, \Sigma_d \leftarrow \emptyset$;
5  foreach task $t_j \in T$ do
6      while not reaching budget limit do
7          Use Bayesian optimization to select configurations from $\Theta$;
8          Evaluate configuration $\theta_i$ based on function $f_d(\cdot)$ or $f_f(\cdot)$;
9          Update prior scalar evaluations: $\mathbf{P} \leftarrow \mathbf{P} \cup \{P(\theta_i, t_j)\}$;
10         Optimize the acquisition function using the surrogate
             model to determine the next $\theta_{m+1}$ and adds it to $\Theta$ ;
11 foreach configuration $\theta_i \in \Theta$ do
12     Identify $\varphi : X \to p_0$ based on configuration $\theta_i$ using $\mathbb{S}(p, x)$;
13     Use $\varphi$ to evaluate the objective function $f_d(\cdot)$ or $f_f(\cdot)$ ;
14     if the objective function $f_d(\cdot)$ or $f_f(\cdot)$ is enhanced then
15         Include the rule $\varphi : X \to p_0$ in $\Sigma_d$ or $\Sigma_f$;
16 Return the set of discovered OMRs: $\Sigma \leftarrow \Sigma_d \cup \Sigma_f$
```

When the configurations Θ are in place, OLeaner learns the predicates of rules in Σ_d and Σ_f, and determines OMRs (line 11). Specifically, the OMRs in Σ_f and Σ_d are learned by solving the meta-learning problem modeled above with the following procedures. (1) OLeaner selects predicates for $\varphi = X \to p_0$. For each configuration $\theta_i \in \Theta$, we select predicates $p_1, p_2, \ldots, p_n$ and their combinations from θ_i using $\mathbb{S}(p, x)$. For each $x_k \in x$, which is $x = \{x_1, x_2, \cdots, x_n\}$, OLeaner selects predicate $p_{i,k}$ from θ_i if $x_k = 1$, and ignores it otherwise. Thus, we obtain OMR $\varphi : X \to p_0$ using θ_i (line 12). (2) OLeaner applies φ to detect (resp. fix) ugly outliers, and evaluates the accuracy using $f_d(\cdot)$ (resp. $f_f(\cdot)$) (line 13). If the objective function $f_d(\cdot)$ (resp. $f_f(\cdot)$) is enhanced after applying φ, OLeaner adds φ to the set Σ_d (resp. Σ_f) (lines 14-15). Since the training of $\mathcal{M}$ and the computation of U_1 and U_2 typically have a cost of $O(|\mathcal{D}|)$, $f_d(\cdot)$ takes linear time to train. (3) After all iterations of Θ, OLeaner returns $\Sigma = \Sigma_d \cup \Sigma_f$ (line 16). The predicates in precondition X of OMRs $\varphi : X \to p_0$ in Σ_f are limited to logic predicates, $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$ and $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$, while those in Σ_d are logic predicates, ML detectors and function predicates. The consequence p_0 of OMRs in Σ_f fixes ugly outliers $t.A$; and p_0 of those in Σ_d is $\mathsf{Rk}(\mathsf{t.A})$ to classify $t.A$ as good, bad or ugly.

Remark. (1) We model rule discovery as a meta learning problem to avoid the exponential enumeration and find effective OMRs within limited computation cost. The number n of candidate predicates for precondition X is bounded by the active domain of $\mathcal{D}$, where continuous ranges are discretized into fixed intervals, and n grows linearly as $|\mathcal{D}|$ increases. Moreover, prior research [21, 40, 70, 110, 115, 148, 166] and our experiments show that most useful predicates in X are ML and function predicates as shown in Section 2.2, which further reduces the candidates of X. (2) We use Bayesian optimization for meta learning to discover OMRs because (a) gradient-descent-based meta learning methods [12, 59, 112] are not suitable since the discrete hyperparameters make the objective function non-differentiable, and (b) Bayesian optimization converges faster [26, 85, 160] than other gradient-free methods [78, 89, 111], by leveraging prior sampled information.

Complexity. OLeaner takes $O(l \times T + I \times (m^3 + K))$ time, where l is the size of the initial configuration set, T is the time to evaluate a configuration, I is the number of iterations, m is the number of attributes in $\mathcal{D}$, and K is the number of steps for optimizing the acquisition function (see Section B for more details).

5 FIXING THE UGLY OUTLIERS

This section develops the algorithm of OHunt for detecting and fixing ugly outliers, denoted by OFix, by employing OMRs. We first present a combination of strategies for finding right values for ugly outliers (Section 5.1). Based on the strategies, we then present OFix to "deep clean" outliers, with accuracy guarantees (Section 5.2).

5.1 Finding Right Values for Ugly Outliers

OFix takes as input a dataset $\mathcal{D}$ of schema $\mathcal{R}$, an ML classifier $\mathcal{M}$ to be trained with $\mathcal{D}$ and a set $\Sigma = \Sigma_d \cup \Sigma_f$ of learned OMRs (Section 4). It detects and fixes ugly outliers of $\mathcal{D}$ by employing OMRs of Σ, to obtain a cleaned dataset $\mathcal{D}_c$ as training data for $\mathcal{M}$.

As shown in Section 2.3, we apply OMRs of the form $X \rightarrow \text{ugly}(t.A)$ in Σ_d to identify ugly outliers. The question is how to fix the detected ugly outliers with right values? Prior work [1, 2, 56, 57, 68, 93–95, 104] often remove tuples that contain outliers. As remarked earlier, this approach may change data distribution and variable association, thus reducing $\mathcal{M}$'s classification accuracy.

OFix proposes to fix outliers caught by OMRs $X \rightarrow \text{ugly}(t.A)$ with a combination of three strategies as follows. It employs OMRs $X \rightarrow t.A = c$ or $X \rightarrow t.A = s.B$ in Σ_f to fix $t.A$ with a right value.

(1) Logic deduction. We can deduce value c for $t.A$ by logic reasoning. An example is $R(t) \wedge \text{ugly}(t.\text{GradeClass}) \wedge t.\text{FamilySupport} > 2 \wedge t.\text{StudyTimeWeekly} > 18 \wedge t.\text{Absence} < 5 \wedge t.\text{GPA} > 3.5 \rightarrow t.\text{GradeClass} = A$. That is, if FamilySupport, StudyHoursWeekly and GPA are high enough and Absence is not too high, then the student should be ranked high. This OMR fixes a label outlier.

Logic reasoning helps fix ugly outliers in features or labels. The fixes can be explained. However, this strategy often covers only a few cases in practice. To fix the other cases, we use ML models.

(2) Statistical analyses. We train an ML model $\mathcal{M}_{\text{FixA}}$ to find a value v for ugly outliers $t.A$. Then we can apply OMRs of the form $R(t) \wedge \text{ugly}(t.A) \wedge \mathcal{M}_{\text{FixA}}(\mathcal{D}, t, A, v) \rightarrow t.A = v$ to fix $t.A$.

(a) Upon the availability of normal tuples relevant to ugly outliers in $\mathcal{D}$, we train $\mathcal{M}_{\text{FixA}}$ to learn data distribution from these tuples and use it to fix ugly outliers in $\mathcal{D}$. Consider the contextual connection $C = (\mathcal{M}_{N \rightarrow U}, U, N)$ where $\mathcal{M}_{N \rightarrow U}$ identifies the set N of normal neighbors for the set U of ugly outliers. Consider ugly outliers in influential features or labels, while other features follow the correct data distribution. Then U and N share a similar distribution. To fix the ugly outliers, we train a KNN model $\mathcal{M}_{\text{knn}}$ on non-ugly tuples, setting the feature values in U as *nan*, and use $\mathcal{M}_{\text{knn}}$ to replace the *nan* values with the mean of neighbors' values. Moreover, $\mathcal{M}_{\text{FixA}}$ explores the association between $t.\bar{B}$ (attributes excluding A) and label $t.A$. For instance, a rich area is often associated with high income, and a female name is linked to a girl. We fix ugly outliers $t.A$ by the association between $t.A$ and $t.\bar{B}$ with high confidence [69].

(b) When training $\mathcal{M}_{\text{FixA}}$ is not feasible due to insufficient samples, we use statistical methods. We identify influential features A and the set U of ugly outliers. We compute statistical metrics (*e.g.,* mean or median) for normal tuples under A, and fix outliers in U with these statistical values. For ugly label outliers $t.Y$, we identify normal tuples with similar features and assign the most frequent label to $t.Y$.

(3) Value extraction from external sources. We train another ML model $\mathcal{M}_{\text{FixE}}$ to predict v using an external data source KB with overlapping information. Then we use OMRs $R(t) \wedge \text{ugly}(t.A) \wedge \mathcal{M}_{\text{FixE}}(\mathcal{D}, t, A, v, \text{KB}) \rightarrow t.A = v$ to fix outliers $t.A$.

For instance, when KB is a knowledge base, we match tuples t in $\mathcal{D}$ and vertices v in KB if they refer to the same real-world entity, by *e.g.,* heterogeneous entity resolution (HER) [52]. Model $\mathcal{M}_{\text{FixE}}(\mathcal{D}, t, A, v, \text{KB})$ determines the right values for ugly outliers $t.A$ with the corresponding features of v. Here KB can be extracted from Wikidata [154] and Kaggle [13], via query interfaces (*e.g.,* GraphQL API [121]), where HER can be enhanced [14, 75, 168].

For example, consider the ugly outlier $\text{ugly}(t.\text{Quality})$ detected by φ_3 on Apple (Section 2.3). Then OMR $R(t) \wedge \text{ugly}(t.\text{Quality}) = -5.96 \wedge \mathcal{M}_{\text{FixE}}(\text{Apple}, t, \text{Quality}, -3.25, \text{Sales}) \rightarrow t.\text{Quality} = -3.25$ corrects the outlier value by matching the tuple $t.\text{Quality} = -5.96$ in relation Apple with the vertex $v = -3.25$ in graph Sales.

If different values are suggested for fixing an ugly outlier $t.A$, we pick the one that maximally improves the accuracy of model $\mathcal{M}$.

5.2 A Recursive Algorithm for Fixing Outliers

Based on the strategies, we develop a "deep cleaning" algorithm OFix, to recursively fix ugly outliers, by chasing dataset $\mathcal{D}$ with the set Σ of OMRs learned. Below we first extend the chase [134].

The chase revised. We start with the notion of fixes.

Fixes. To keep track of fixes to ugly outliers, for each attribute $t.A$, we maintain relations $[t.A]_{\otimes}$, where $\otimes$ ranges over $=, \neq, <, \leq, >, \geq$. Each $[t.A]_{\otimes}$ consists of attributes $t'[B]$ or constants c such that $t.A \otimes t'[B]$ and $t.A \otimes c$ are either in the ground truth Γ (see below) or deduced during the chase. In particular, if $t.A$ is an outlier, then $[t.A]_{=}$ may include Rk (*i.e.,* good, bad, ugly) as a special value for label, and a constant in $[t.A]_{=}$ that makes a fix to $t.A$.

We ensure that each $[t.A]_{=}$ and $[t.A]_{\neq}$ are equivalence relations, *i.e.,* reflexive, symmetric and transitive, while each $[t.A]_{\otimes}$ is reflexive and transitive for the other $\otimes$. Moreover, each $[t.A]_{\otimes}$ is *valid*, *i.e.,* no c_1 and c_2 in the same $[t.A]_{\otimes}$ conflict with each other, *e.g.,* $[t.A]_{=}$ does not include constants c_1 and c_2 where $c_1 \neq c_2$.

Ground truth. The initial set of fixes is denoted as Γ, which are collected and validated by users, domain experts or crowd-sourcing.

The chase. A *chase step* of dataset $\mathcal{D}$ with Σ_f at a set $\mathcal{F}$ of fixes is

$$\mathcal{F} \Rightarrow_{(\varphi,h)} \mathcal{F}'.$$

Here $\varphi : X \rightarrow p_0$ is an OMR in Σ_f and h is a valuation of φ such that (a) all predicates in X are *validated* by $\mathcal{F}$; that is, if p is $t.A \otimes s.B$, then $h(s).B \in [t.A]_{\otimes}$ for $[t.A]_{\otimes}$ in $\mathcal{F}$; if p is $\mathcal{M}_o(t, A, \mathcal{D}, \mathcal{M})$, then $\mathcal{M}_o$ predicts true at $h(t).A$ and all the data involved in the prediction is in $\mathcal{F}$ (*e.g.,* neighbor features if $\mathcal{M}_o$ is KNN); similarly for $F(t, A, \mathcal{D}, \mathcal{M})$; if p is $\mathsf{Rk}(t.A)$, then Rk is in $[t.A]_\otimes$; and (b) the consequence p_0 extends $\mathcal{F}$ to $\mathcal{F}'$, by adding c (resp. $s.B$, Rk) to $[t.A]_\otimes$ in $\mathcal{F}'$ if p_0 is $t.A \otimes c$ (resp. $t.A \otimes s.B$, $\mathsf{Rk}(t.A)$). That is, the chase expands ground truth with validated fixes in the process.

A *chasing sequence* ξ of $\mathcal{D}$ by (Σ, Γ) is a sequence

$$\mathcal{F}_0, \ldots, \mathcal{F}_n,$$

where $\mathcal{F}_0$ is Γ and for each $i \in [1, n]$, there exist φ in Σ and valuation h of φ such that $\mathcal{F}_{i-1} \Rightarrow_{(\varphi,h)} \mathcal{F}_i$ is a valid chase step, *i.e.,* $\mathcal{F}_i$ is valid.

The sequence ξ *terminates* when no more OMRs in Σ can be applied to further extend $\mathcal{F}_n$ with new fixes. Such a sequence is referred to as a *terminal chasing sequence*. When a sequence terminates, we refer to the set $\mathcal{F}_n$ of fixes as the result of ξ.

Challenges. Directly implementing the chase may lead to conflicts when multiple rules are applicable to fixing the same outlier, with different values. It is also costly if we apply such rules one by one.

Algorithm. Based on the revised chase, we develop OFix as shown in Algorithm 2. It consists of initialization, chasing and correction.

(1) Algorithm OFix first employs OMRs in Σ_d to find all outliers. It stores the ugly ones in U (line 1). Then after trained with all kinds of outliers and normal tuples offline (line 2), $\mathcal{M}_{\mathsf{FixA}}$ and $\mathcal{M}_{\mathsf{FixE}}$ can predict correct value $t.A = v$ to fix $\mathsf{ugly}(t.A)$. The ground truth Γ is initialized with validated facts in $\mathcal{D}$ and high-quality KB (line 3).

(2) During chasing steps, OFix adopts batch processing when multiple OMRs are applicable simultaneously. Denote such rules as the subset $\Sigma_t \subseteq \Sigma_f$ when the same unverified $\mathsf{ugly}(t.A)$ appears in their precondition X (line 7). We initialize the candidate correction set C of Σ_t as $\emptyset$ (line 8). OMRs from Σ_t are activated in parallel to deduce fixes c for $t.A$ in U. If the fix c is already in Γ, $\mathsf{ugly}(t.A)$ is fixed directly using c (lines 10-11). Otherwise, if c is inferred, $\mathsf{ugly}(t.A)$ is added to the set $C = \{c_i \mid \exists\varphi \in \Sigma_t : \varphi \models X \rightarrow c_i\}$ (lines 12-13).

After the batch processing, OFix resolves conflicts in C as follows. When multiple fixes $t.A = c$ are deduced, OFix employs $\mathsf{imp}(\mathcal{M}, c)$ (*i.e.,* $\mathcal{M}$'s classification accuracy improvement) to decide the suitable value, which is computed as $c^* = \mathsf{argmax}_{c \in C} \mathsf{imp}(\mathcal{M}, c)$ (lines 15-16). Then, c^* is used to fix $t.A$, and added to $[t.A]_=$ (*i.e.,* Γ) to resolve conflicts in C (line 17). After $\mathsf{ugly}(t.A)$ is fixed, OFix removes it from U (line 17). The remaining $\mathsf{ugly}(t.A)$ in U and updated Γ activate more OMRs from Σ_f, and the chase continues (lines 4-5). When there are no changes can be made, the chase terminates.

(3) Upon termination, OFix deduces a new training set $\mathcal{D}_c$ from $\mathcal{D}$ by fixing all ugly outliers $t.A$ in U with values in $[t.A]_=$ (line 18).

Remark. OFix addresses chasing challenges by (1) using batch processing to infer multiple OMRs simultaneously, thus reducing time cost; and (2) resolving conflicts by objective function optimization. These ensure that each ugly outlier has a unique fix.

Complexity. OFix takes $O(|U| \times (\tau \times n + |\Sigma|) + n \times m \times w)$ time, where τ is the number of nearest neighbors in KNN, $|\Sigma|$ is the number of rules in Σ, n is the number of tuples in $\mathcal{D}$, m is the number of attributes in $\mathcal{D}$, w is the number of vertices in KB, and $|U|$ is the number of ugly outliers to be fixed (see Section B for details).

Termination and accuracy guarantees. OFix conducts deep cleaning since fixes at one chase step may help generate fixes in subsequent steps. Moreover, one can verify that wrong fixes cannot enter $\mathcal{F}_k$ if Σ and Γ are correct and ML/function predicates are

Algorithm 2: Algorithm OFix

```
   Input: $R$, $\mathcal{D}$, $\mathcal{M}$, KB as stated in above, $\Sigma = \Sigma_d \cup \Sigma_f$ of OMRs.
   Output: A dataset $\mathcal{D}_c$ by fixing ugly outliers in $\mathcal{D}$ with $\Sigma$.
1  Add all ugly outliers $t.A$ identified by $\Sigma_d$ to $U$;
2  Train $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$ and $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ offline;
3  $\Gamma \leftarrow \{[t.A] = v \mid t.A \in \mathcal{D}, \mathsf{KB}\}$;  flag := true;
4  while flag do
5      flag := false;
6      foreach ugly outlier $t.A \in U$ do
7          Identify the subset $\Sigma_t$ of rules from $\Sigma_f$ whose
             preconditions involve ugly$(t.A)$;
8          The candidate correction set $C \leftarrow \emptyset$;
9          foreach OMR $\varphi \in \Sigma_t$ in parallel do
10             if $\varphi$ produces a value $c \in \Gamma$ then
11                 Fix $t.A$ directly using $c$;
12             else if $\varphi$ deduces a new value $c \notin \Gamma$ then
13                 $C \leftarrow \{c_i \mid \exists\varphi \in \Sigma_t : \varphi \models X \rightarrow c_i\}$;
14         if multiple $c_i$'s appear in $C$ and $t.A$ has not been fixed then
15             Compute imp$(\mathcal{M}, c)$ for each candidate $c \in C$;
16             Optimal $c^* \leftarrow \mathsf{argmax}_{c \in C}\mathsf{imp}(\mathcal{M}, c)$;
17             Fix $t.A$ using $c^*$ and add $[t.A]_= c^*$ to $\Gamma$;
18         Remove ugly outlier $t.A$ from $U$; flag := true;
19 return Fixed dataset $\mathcal{D}_c$ with ugly outliers resolved;
```

accurate, by induction on chase steps. In addition, we show that OFix is Church-Rosser. A chase-based algorithm is *Church-Rosser* if for any dataset $\mathcal{D}$, set of rules Σ, and ground truth Γ, all chasing sequences of ξ by (Σ, Γ) terminate and converge at the same result, regardless of the rules used or their application order (cf. [5]).

Proposition 1: OFix *with* OMRs *is Church-Rosser.* □

Proof sketch: The proof has two steps. (1) *Any chasing sequence is finite.* Intuitively, each chasing step fixes at least one ugly outlier and the set U is finite; hence so is ξ. Specifically, for any terminal chasing sequence $\xi = (\mathcal{F}_0, \ldots, \mathcal{F}_n)$ of $\mathcal{D}$ by (Σ, Γ), we verify that $n \leq |\mathcal{D}|^2 + |\Gamma||\mathcal{D}|$, since chase step (a) assigns a constant c from Γ or deduced fixes to $t.A$, which makes at most $|\Gamma||\mathcal{D}|$ steps; or (b) sets two attributes equal, at most enumerating all tuple pairs ($|\mathcal{D}|^2$). The values extracted from KB via HER are bounded by $|\mathcal{D}|$.

(2) *All chasing sequences terminate at the same result.* Assume by contradiction that two terminal chasing sequences $\xi_1 = (\mathcal{F}_0, \ldots, \mathcal{F}_{n_1})$ and $\xi_2 = (\mathcal{F}'_0, \ldots, \mathcal{F}'_{n_2})$ of $\mathcal{D}$ with (Σ, Γ) have different results. Since ξ_1 and ξ_2 differ, there exists a chase step $\mathcal{F}'_i \Rightarrow_{(\varphi,h)} \mathcal{F}'_{i+1}$ in ξ_2 such that $\mathcal{F}'_{i+1}$ extends $\mathcal{F}'_i$ *w.r.t.* valuation $h(t').A$ of a tuple variable t' of φ but $h(t').A$ is not in $\mathcal{F}_{n_1}$ of ξ_1. However, we verify that $\mathcal{F}_{n_1} \Rightarrow_{(\varphi,h)} \mathcal{F}_{n_1+1}$ is a valid chase step expanding $\mathcal{F}_{n_1}$ *w.r.t.* $h(t').A$ by induction on the length of ξ_1, contradicting the assumption that ξ_1 is terminal. Once training completes, the predictions of ML models in OMRs are consistent across ξ_1 and ξ_2 (see Section C). □

6 EXPERIMENTAL STUDY

Using real-life and synthetic datasets, we empirically verified the accuracy, robustness and scalability of OHunt for detecting and fixing ugly outliers, and its impact on the accuracy of ML classifiers.

Experimental setting. We start with the setting.

Datasets. We used 18 real-life datasets with different sizes and outlier rates from different domains (*e.g.,* health); 11 are in Table 2 (see

Name	Domain	#Samples	#Features	%Outliers	$\#\lvert\Sigma_d\rvert$	$\#\lvert\Sigma_f\rvert$
Cardio [166]	health	2,114	21	22.04	38	89
Annthyroid [102]	health	7,200	6	7.42	21	35
Optdigits [7]	image	5,216	64	2.88	133	242
PageBlocks [107]	classification	5,393	10	9.46	35	93
Pendigits [136]	image	6,870	16	2.27	43	82
Satellite [144]	image	6,435	36	31.64	114	198
Shuttle [147]	classification	49,097	9	7.15	57	114
Yeast [109]	health	1,484	8	34.16	29	62
Census [115]	society	199,523	44	6.21	189	303
Covtype [41]	classification	581,012	55	0.47	158	269
Flights [41]	traffic	1,000,000	31	1.12	102	182

Table 2: Real-life datasets

$\mathcal{M}$	$\mathcal{D}$	ABOD	ECOD	Iforest	RePEN	SLAD	ICL	NeuTraL	uOMRs
SVM	Annthyroid	0.000	0.177	0.178	0.175	0.023	0.271	0.305	**0.981**
	Cardio	0.171	0.123	0.105	0.075	0.035	0.124	0.141	**0.970**
	Optdigits	0.500	0.273	0.778	0.268	0.439	0.512	0.280	**1.000**
RF	Shuttle	0.818	0.444	0.385	0.739	0.474	0.813	0.588	**0.955**
	Yeast	0.068	0.085	0.075	0.060	0.115	0.077	0.075	**1.000**
Softmax	PageBlocks	0.226	0.243	0.339	0.363	0.114	0.198	0.150	**0.994**
	Pendigits	0.615	0.333	0.692	0.818	0.250	0.667	0.333	**1.000**
	Shuttle	0.115	0.007	0.030	0.042	0.143	0.139	0.143	**1.000**

Table 3: Unsupervised methods for Ugly Outlier Detection

$\mathcal{M}$	$\mathcal{D}$	CatB	LGB	XGB	DevNet	DSAD	RoSAS	PReNeT	sOMRs
SVM	Annthyroid	0.042	0.075	0.073	0.251	0.162	0.295	0.358	**0.989**
	Cardio	0.034	0.026	0.044	0.121	0.081	0.136	0.071	**1.000**
	Optdigits	0.000	0.000	0.000	0.250	0.875	0.444	0.867	**0.995**
RF	Shuttle	0.000	0.000	0.000	0.381	0.522	0.640	0.800	**1.000**
	Yeast	0.003	0.017	0.018	0.072	0.057	0.119	0.122	**0.996**
Softmax	PageBlocks	0.009	0.017	0.015	0.325	0.242	0.259	0.331	**1.000**
	Pendigits	0.000	0.004	0.000	0.307	0.330	0.525	0.754	**1.000**
	Shuttle	0.011	0.037	0.033	0.109	0.156	0.015	0.007	**1.000**

Table 4: Label-based Algorithms for Ugly Outlier Detection

Section D for the other 7). Pendigits, PageBlocks, Optdigits, Cardio, Shuttle, Yeast, Satellite, Census and Annthyroid are from outlier detection benchmarks, and the rest come from public repositories.

Besides testing real outliers, we created 20 synthetic datasets based on real data to test the robustness of OHunt. Each dataset is injected with one of four types of noises: local, global, cluster and dependency [70]. Local noise refers to outliers deviating from their local neighborhoods [24, 70]. Global noise exhibits larger deviations from normal data [70, 80], generated from a uniform distribution. Cluster noises [96] are groups of outlier instances that share similar characteristics and appear close together in the feature space [46, 100]. Dependency noise involves deviations from the typical dependency structure of normal data [70, 108]. Among these experimental datasets, all real outliers are labeled.

ML classifiers. We tested five ML classifiers. (1) Support Vector Machine (SVM) [145] builds a maximum-margin decision boundary to classify samples and is effective in high-dimensional spaces. (2) Naive Bayes (NB) [17] assumes feature independence to compute class probabilities by Bayesian theorem. (3) Random Forest (RF) [23] adopts ensemble learning to aggregate predictions from decision trees through voting. (4) Softmax [25] assigns a sample to the category with the highest score, as commonly used as the final layer in neural networks for multi-class classification. (5) Decision Tree (DT) [125] employs a hierarchical structure to split data iteratively based on feature values, with interpretability through thresholds. Unless stated otherwise, SVM was selected as the default classifier.

Baselines. We compared OHunt with 22 outlier detection baselines following [40, 70, 162, 163], integrated with outlier correction methods in Exp-2. (1) Unsupervised ones: (a) Statistical: ECOD [99], COPOD [98], LODA [120] and RCA [101]. (b) Distance-based: ABOD [92]. (c) Ensemble-based: IForest [102]. (d) Deep leaning: DAGMM [173], SLAD [165], GOAD [18], DSADD [132], REPEN [114], ICL [141] and NeuTraL [124]. (2) Semi-supervised: PReNet [116], DevNet [117], DSAD [133] and ROSAS [164]. (3) Supervised: CatBoost [123], LGBoost [86] and XGBoost [31]. (4) Rule-based ones include ID3 [40, 125] and CART [40, 105].

Rules. We learned OMRs from 18 real-world datasets. The number of learned OMRs for detecting ($\lvert\Sigma_d\rvert$) and fixing outliers ($\lvert\Sigma_f\rvert$) are shown in Table 2. Based on the availability of labels during training, we classify OMRs learned into uOMRs and sOMRs. We compared uOMRs with unsupervised baselines, and sOMRs with label-based (supervised, semi-supervised, and rule-based) baselines.

Accuracy metrics. We adopt the following. (1) Accuracy acc($\mathcal{M}, \mathcal{D}$) defined in Section 2.1. (2) Recall (R), the ratio of actual outliers correctly predicted as positive. (3) F1-score = $2 \times \frac{P \times R}{P+R}$, where P is the ratio of actual outliers to all predicted positives. (4) PR-AUC [37], the area under the precision-recall curve at different thresholds. A higher PR-AUC indicates better classification, especially for imbalanced datasets. (5) Average Precision (AP) [153], the weighted average of P-values across different recall levels, typically used for imbalanced datasets. (6) ROC-AUC [71], a metric for binary classification, where a value closer to 1 indicates better performance. (7) Critical Difference (CD) Diagram [38, 84], a tool for visualizing and comparing the performance of multiple algorithms or models.

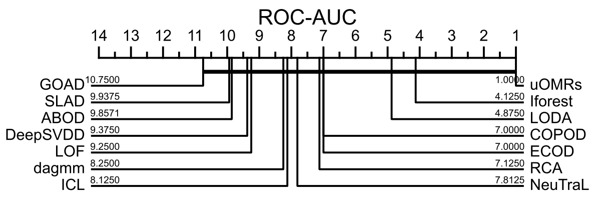
Environmental setting. We conducted all experiments on a server equipped with an Intel Core 2.90 GHz CPU and 32 GB of memory. OHunt and all the baselines were implemented in Python 3.8. The version of the PyTorch library is PyTorch 1.8.0. Each experiment was run 3 times, and the average result is reported here.

Experimental findings. We next report our findings.

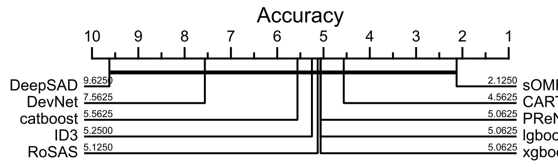
Exp-1: Outlier detection: Accuracy and robustness. We tested OHunt for (1) the proportions of good, bad and ugly outliers in real-life datasets and the recall/F1-score for ugly outliers. (2) Robustness in detecting ugly outliers in synthetic datasets with varying noise ratios and types. (3) Effectiveness in distinguishing outliers from all samples in binary classification datasets [21] using the CD diagrams/plots ($p \leq 0.05$, which is the significance level).

(1) Accuracy. We start with the detection of different outliers.

(a) The good, the bad, and the ugly. (i) Across all real-world datasets tested, outliers range from 0.47% to 49.90%, 14.08% on average. On average, good outliers make up 80.33% of all outliers, *e.g.,* 69.44% in the Apple dataset and 95.25% in Covtype. Bad outliers are 19.67% on average, in which the ratio of ugly outliers varies significantly. For instance, in Star, 97.06% of the 1,103 bad outliers are ugly, versus an average of 64.72% across all datasets. (ii) Predicates $\mathcal{M}_{\mathsf{DetO}}(\cdot)$ and outlier$(\cdot)$ are effective in identifying outliers, while loss$(\cdot)$ distinguishes good and bad ones. During the training of $\mathcal{M}$, loss$(\cdot)$ for good outliers decreases below a threshold λ learned via Bayesian optimization, while it remains stable for bad outliers. (iii) Model $\mathcal{M}_{\mathsf{DetI}}(\cdot)$ identifies influential features for ugly outliers, while function imbalanced$(\cdot)$ picks out imbalanced ugly outliers in bad ones.



(a) CD plot of ROC-AUC on unsupervised algorithms



(b) CD plot of Accuracy on label-based algorithms

Figure 1: Performance Comparison of OMRs and Baselines in Binary Outlier Detection Tasks

(b) Recall/F1-score of ugly outliers. We evaluated the recall and F1-score of ugly outliers from eight real datasets using three classifiers (SVM, RF, and Softmax). We compared the accuracy of uOMRs (resp. sOMRs) against unsupervised (resp. label-based) baselines. As shown in Tables 3 and 4. on average, (i) the recall of uOMRs is 0.988 (up to 1.000), 70.7% higher than unsupervised baselines. (ii) The recall of sOMRs is 0.998 (up to 1.000), 81.2% higher than label-based baselines. (iii) The average F1-score of uOMRs is 0.966 (up to 0.998), 68.8% higher than unsupervised baselines. (iv) For sOMRs, the average F1-score is 0.996 (up to 1.000), 78.4% higher than label-based baselines. These results verify that OHunt is more effective in detecting outliers that negatively impact ML classifiers.

(2) Robustness. Besides real outliers, we injected one of local, global, cluster and dependency noise into real datasets Cardio, PageBlocks (PB), Annthyroid (ATR) and Waveform (WF), by varying its ratio at 0.10/0.15/0.20/0.25/0.30, yielding 20 noisy datasets. Here k_l, k_g, k_c and k_d denote the ratio of the four types of noise, respectively.

As shown in Figure 2(a-d), (a) OHunt is most sensitive to dependency noise. In Waveform, as k_d increases from 0.10 to 0.30, the F1-score for detecting ugly outliers with uOMRs decreases by 0.194 (compared to 0.183 for local, 0.071 for global, and 0.106 for cluster), while the F1-score of sOMRs decreases by 0.260 (versus 0.140 for local, 0.080 for global, and 0.098 for cluster). (b) For high noise ratios, OHunt consistently outperforms the baselines; its average F1-score is 70.4% higher, up to 97.7%. Even in the noise-sensitive Waveform dataset (k_d is 0.3), the F1-score of OHunt is consistently above 0.6, while it is below 0.2 for the baselines; this demonstrates OHunt's robustness against noise. (c) For each noise type, when ratio k_l, k_g, k_c or k_d increases, the F1-score for detecting ugly outliers decreases for all methods, as expected. For instance, in PageBlocks with global noise, the F1-score of uOMRs drops from 0.968 to 0.897, while it drops from 0.977 to 0.897 for sOMRs as k_g increases from 0.1 to 0.3. This said, OHunt still outperforms all the baselines.

(3) Classical tasks. We evaluated the performance of OHunt versus 22 baselines on 18 real-world datasets for classical outlier detection, *i.e.,* distinguishing outliers from binary datasets composed of normal and outlier data. The analysis of Accuracy and ROC-AUC metrics is based on CD diagrams. As shown in Figure 1(a-b), uOMRs beat 13 unsupervised methods in terms of ROC-AUC. For Accuracy, sOMRs ranks the first among 10 label-based methods. These verify that OHunt is effective in classical outlier detection.

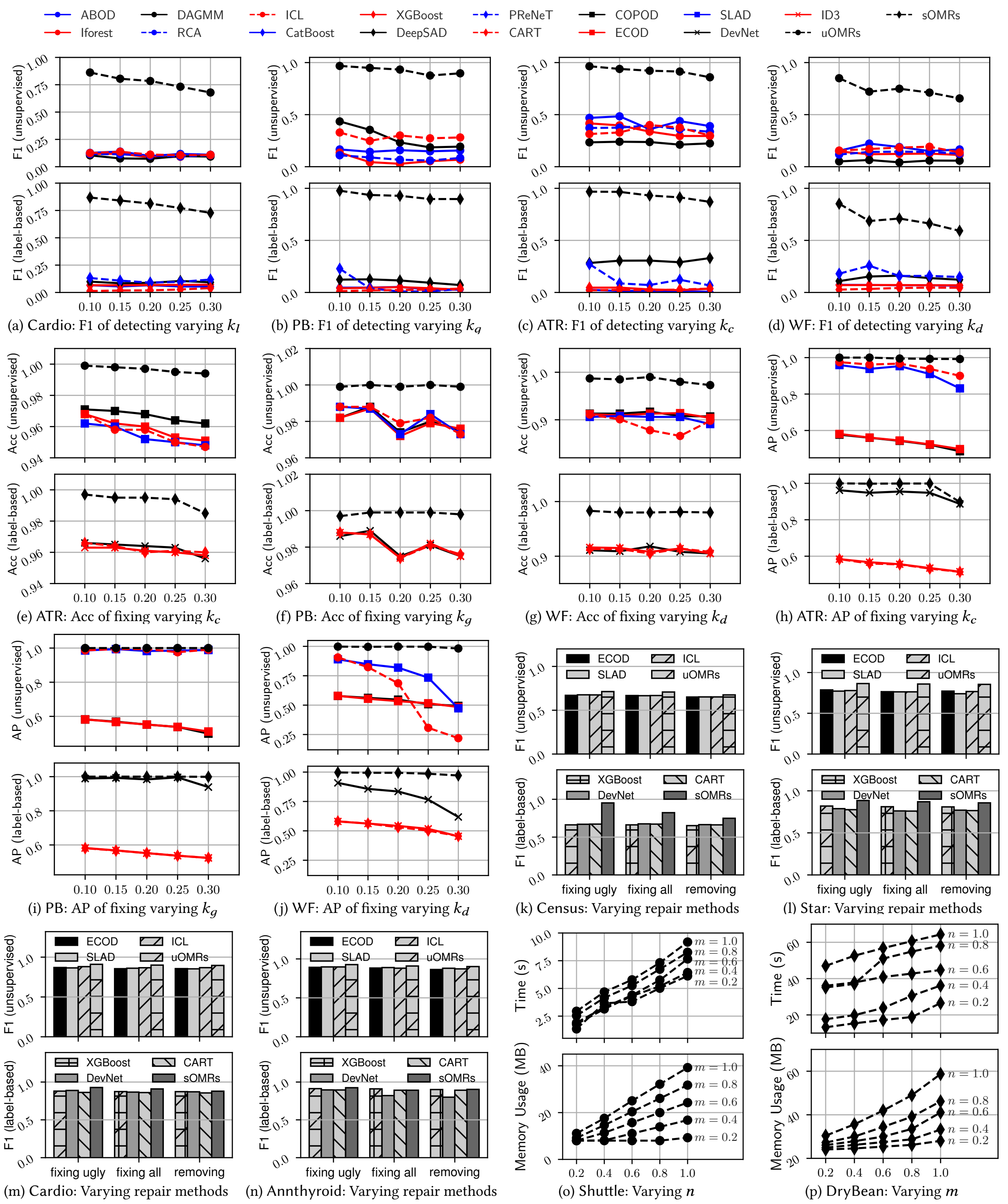
Exp-2: Fixing ugly outliers: Accuracy and robustness. We evaluated OHunt's effectiveness in fixing outliers and improving ML classifier accuracy, focusing on: (1) the impact on the F1-score of $\mathcal{M}$, (2) the impact of fixing outliers vs. removing tuples, and (3) OHunt's robustness to varying outlier ratios (0.10-0.30) and types (local/global/cluster/dependency). Since baselines either detect or correct outliers, we combine detectors with correction methods (see Section 7) for a fair comparison with OHunt. More specifically, we combine (1) elimination and statistics-based correction methods (*e.g.,* [83]) with unsupervised detectors (*e.g.,* SLAD); (2) ML-based (*e.g.,* [152]) and DB4AI methods (*e.g.,* [57]) for label-based correction, with semi-supervised, supervised, and rule-based detectors (*e.g.,* CART); and (3) for uOMRs and sOMRs in OHunt, we use the chasing process in the OFix algorithm for outlier correction.

(1) Accuracy. We used multi-class datasets (e.g., DryBean) and real outlier datasets (e.g., Shuttle), splitting them into 70% training and 30% testing sets. We trained five classifiers (SVM, Softmax, DT, RF, NB) on the training data and computed their F1-scores on the test set. We then detected and repaired ugly outliers in $\mathcal{D}$ using OHunt and baselines. After these, we retrained the classifiers $\mathcal{M}$ on the cleaned data $\mathcal{D}_c$ and recalculated their F1 scores.

As shown in Table 5, OHunt improves $\mathcal{M}$'s F1-score by an average of 6.9%, up to 21.6%, outperforming the baselines by an average of 6.6% and up to 21.1%. More specifically, uOMRs outperform the baselines by 5.1% on average, up to 21.1%, and sOMRs improve by 8.1%, up to 20.4%. For instance, the F1-score of SVM with uOMRs is 0.959 on DryBean, compared to 0.892 with NeuTraL for detection and [149] for repair; this makes a 6.7% improvement. We find that that detecting and fixing ugly outliers is more effective than repairing all outliers in baselines. This is because (a) the latter may incorrectly repair good and bad outliers that should not be fixed, introducing new errors and altering the normal data distribution of $\mathcal{D}$; and (b) the latter cannot guarantee the Church-Rosser property when repairing ugly outliers, degrading the data quality in D_c.

(2) Robustness. In the same setting as the robustness tests in Exp-1, we report acc$(\mathcal{M}, \mathcal{D}_c)$ and average precision (AP) of ML classifiers under different noise ratios and types. Notably, acc$(\mathcal{M}, \mathcal{D}_c)$ is used to evaluate the improvement in classification accuracy by OHunt compared to the baselines, while AP highlights OHunt's robustness relative to the baselines on imbalanced datasets.

As shown in Figure 2(e-j), (a) for different noise ratios and types, OHunt beats all the baselines in acc$(\mathcal{M}, \mathcal{D}_c)$ and AP by 13.2% and 38.3% on average, respectively, up to 17.5% and 55.9%. OHunt is less sensitive to noise than all the baselines. This is because it employs functions for influential features and loss when detecting ugly outliers, and logical reasoning to reduce false positives/negatives of ML-based detectors. When fixing ugly outliers, OHunt combines logical deduction, external data and statistical methods, rather than removing these outliers, which is crucial when noise ratio is high. (b) Both uOMRs and sOMRs exhibit robustness to noise ratios and types. For example, in Annthyroid with a 0.1 cluster noise ratio, acc$(\mathcal{M}, \mathcal{D}_c)$ (resp. AP) is 0.999 (resp. 1.000) for uOMRs, and 0.997 and (resp. 1.000) for sOMRs. When the noise ratio increases to 0.3, acc$(\mathcal{M}, \mathcal{D}_c)$ (resp. AP) only reduces to 0.994 (resp. 0.992) for uOMRs, and 0.985 (resp. 0.897) for sOMRs, verifying the robustness.


(a) Cardio: F1 of detecting varying k_l (b) PB: F1 of detecting varying k_g (c) ATR: F1 of detecting varying k_c (d) WF: F1 of detecting varying k_d

(e) ATR: Acc of fixing varying k_c (f) PB: Acc of fixing varying k_g (g) WF: Acc of fixing varying k_d (h) ATR: AP of fixing varying k_c

(i) PB: AP of fixing varying k_g (j) WF: AP of fixing varying k_d (k) Census: Varying repair methods (l) Star: Varying repair methods

(m) Cardio: Varying repair methods (n) Annthyroid: Varying repair methods (o) Shuttle: Varying n (p) DryBean: Varying m

Figure 2: Performance evaluation of OHunt **(uOMRs/sOMRs) and baselines**

<u>*(3) Ablation study.*</u> We evaluated the impact of (a) various detectors, (b) correction strategies: fixing only ugly outliers vs. all outliers, and removing ugly tuples vs. fixing ugly outliers in place, and (c) the impact of logic reasoning, ML predicates and function predicates.

(a-b) We tested eight outlier detectors and three cleaning strategies using four real datasets: Census, Star, Cardio and Annthyroid. We investigated the impacts of removing ugly tuples ($\text{OHunt}_{\text{rem}}$), fixing ugly outliers ($\text{OHunt}_{\text{ugly}}$), and fixing all outliers ($\text{OHunt}_{\text{all}}$) on the F1-score of the SVM classifier ($\mathcal{M}$). The selected detectors include OMRs (uOMRs and sOMRs), three unsupervised baselines (*e.g.,* SLAD), and three label-based baselines (*e.g.,* rule-based CART).

As shown in Figure 2(k-n), (i) by using uOMRs and sOMRs,

$\mathcal{M}$	$\mathcal{D}$	F1($\mathcal{M}$, $\mathcal{D}$)	ABOD	ECOD	LODA	DSADD	RePEN	SLAD	ICL	NeuTraL	**u**OMRs	DevNet	DSAD	RoSAS	PReNet	ID3	CART	**s**OMRs
SVM	DryBean	0.878	0.885	0.873	0.879	0.884	0.868	0.878	0.889	0.892	**0.959**	0.875	0.884	0.880	0.878	0.882	0.878	0.964
	Online	0.743	0.756	0.755	0.752	0.740	0.754	0.740	0.743	0.730	**0.807**	0.736	0.732	0.696	0.726	0.740	0.736	0.806
	Cardio	0.865	0.870	0.870	0.877	0.866	0.878	0.865	0.881	0.880	**0.897**	0.875	0.865	0.871	0.874	0.864	0.865	0.934
	PageBlocks	0.891	0.891	0.893	0.896	0.892	0.875	0.891	0.894	0.896	**0.928**	0.888	0.892	0.861	0.911	0.887	0.891	0.953
	Satellite	0.842	0.846	0.841	0.841	0.841	0.839	0.847	0.848	0.845	**0.861**	0.840	0.845	0.842	0.843	0.844	0.839	0.958
RF	Census	0.646	0.667	0.661	0.657	0.646	0.656	0.646	0.654	0.663	**0.700**	0.651	0.653	0.647	0.645	0.657	0.641	0.786
	Star	0.889	0.910	0.904	0.902	0.899	0.902	0.900	0.903	0.893	**0.930**	0.894	0.901	0.898	0.889	0.900	0.898	0.948
	Student	0.812	0.816	0.828	0.810	0.821	0.825	0.793	0.812	0.783	**0.881**	0.810	0.850	0.790	0.801	0.817	0.819	0.867
	PageBlocks	0.954	0.958	0.957	0.960	0.955	0.961	0.955	0.956	0.951	**0.964**	0.953	0.965	0.956	0.955	0.953	0.960	0.970
	Yeast	0.673	0.699	0.700	0.686	0.693	0.691	0.709	0.678	0.716	**0.729**	0.702	0.718	0.685	0.712	0.696	0.692	0.889
Softmax	DryBean	0.829	0.866	0.859	0.856	0.859	0.860	0.851	0.864	0.870	**0.951**	0.855	0.858	0.849	0.858	0.864	0.864	0.972
	Online	0.729	0.755	0.746	0.747	0.732	0.748	0.737	0.740	0.735	**0.790**	0.743	0.742	0.717	0.734	0.741	0.741	0.773
	Star	0.799	0.815	0.807	0.808	0.802	0.810	0.807	0.816	0.815	**0.851**	0.804	0.803	0.799	0.803	0.804	0.804	0.939
	Student	0.616	0.619	0.613	0.624	0.617	0.611	0.610	0.623	0.620	**0.821**	0.608	0.610	0.625	0.635	0.621	0.617	0.746
	Satellite	0.810	0.828	0.826	0.825	0.821	0.826	0.827	0.832	0.830	**0.846**	0.829	0.823	0.827	0.831	0.828	0.823	0.951
DT	Online	0.755	0.758	0.756	0.756	0.744	0.751	0.750	0.738	0.719	**0.783**	0.733	0.737	0.722	0.722	0.737	0.734	0.764
	Star	0.879	0.878	0.883	0.886	0.867	0.879	0.872	0.879	0.873	**0.910**	0.867	0.881	0.870	0.859	0.877	0.880	0.909
	Student	0.799	0.798	0.793	0.790	0.787	0.793	0.748	0.779	0.737	**0.847**	0.755	0.804	0.761	0.772	0.804	0.806	0.839
	Cardio	0.879	0.902	0.894	0.883	0.893	0.905	0.893	0.905	0.894	**0.916**	0.890	0.900	0.901	0.897	0.891	0.890	0.918
	Yeast	0.660	0.690	0.676	0.679	0.679	0.655	0.671	0.661	0.643	**0.704**	0.647	0.644	0.666	0.643	0.628	0.628	0.718
NB	DryBean	0.852	0.854	0.848	0.855	0.855	0.817	0.846	0.853	0.859	**0.904**	0.827	0.848	0.844	0.855	0.853	0.853	0.928
	Online	0.751	0.772	0.771	0.770	0.757	0.769	0.761	0.755	0.741	**0.821**	0.750	0.749	0.682	0.720	0.745	0.731	0.768
	Student	0.696	0.711	0.693	0.697	0.698	0.698	0.658	0.691	0.687	**0.834**	0.691	0.722	0.677	0.704	0.700	0.698	0.806
	Satellite	0.764	0.764	0.764	0.763	0.775	0.764	0.775	0.772	0.774	**0.785**	0.769	0.769	0.764	0.764	0.764	0.764	0.943
	Shuttle	0.989	0.989	0.988	0.989	0.989	0.970	0.988	0.988	0.987	**0.992**	0.989	0.989	0.975	0.988	0.987	0.987	0.999

Table 5: Comparison of OMRs **and Outlier Detection Baselines in Downstream Classification F1-Score**

$\text{OHunt}_{\text{rem}}$, $\text{OHunt}_{\text{ugly}}$ and $\text{OHunt}_{\text{all}}$ improve the F1-scores of $\mathcal{M}$ better than with all unsupervised and label-based baselines. For example, for $\text{OHunt}_{\text{ugly}}$ on Adult, the F1-score of uOMRs and sOMRs is 0.713 and 0.952, respectively, while it is 0.676 and 0.672 for the best unsupervised (SLAD) and label-based (CART) baselines. This suggests that OHunt is able to accurately detect and fix ugly outliers responsible for classification errors. (ii) Under the F1 metric, $\text{OHunt}_{\text{ugly}}$ and $\text{OHunt}_{\text{all}}$ with uOMRs improve the F1-score of $\mathcal{M}$ similarly (0.851 and 0.844 on average), both higher than $\text{OHunt}_{\text{rem}}$ (0.832 on average). (iii) For sOMRs, $\text{OHunt}_{\text{ugly}}$ has a greater impact on the F1-score of $\mathcal{M}$ than $\text{OHunt}_{\text{all}}$, with average F1-score of 0.918 and 0.874, respectively. Both $\text{OHunt}_{\text{ugly}}$ and $\text{OHunt}_{\text{all}}$ consistently beat $\text{OHunt}_{\text{rem}}$, with which the average F1-score of $\mathcal{M}$ is 0.860.

(c) We evaluated the impact of OMR predicates by removing fixing predicates ($\text{OHunt}_{\text{nofix}}$), ML predicates ($\text{OHunt}_{\text{noml}}$), logic reasoning ($\text{OHunt}_{\text{nolr}}$), or function predicates ($\text{OHunt}_{\text{nofun}}$). We fix ugly outliers and use uOMRs and sOMRs as detectors. The datasets, metrics and classifier are the same as in (a-b). We find that $\text{OHunt}_{\text{nofix}}$ and $\text{OHunt}_{\text{noml}}$ do not improve $\mathcal{M}$'s F1-score, since they are unable to conduct the chase in OFix or learn high-quality rules in OLeaner. Compared to the complete OHunt, (i) the F1-score of classifier $\mathcal{M}$ is 0.2% lower for uOMRs and 0.6% lower for sOMRs with $\text{OHunt}_{\text{nolr}}$ since it cannot reduce false positives/negatives of ML predictions; and (ii) it is 1.9% lower for uOMRs and 5.8% for sOMRs with $\text{OHunt}_{\text{nofun}}$ due to the inability to distinguish good, bad, and ugly outliers without function predicate loss($\mathcal{M}, \mathcal{D}, t, A$).

Exp-3: Scalability. We also studied the scalability of OHunt using real datasets, by varying (1) the sampling ratio n of tuples; (2) the sampling ratio m of attributes. We also evaluated (3) the scalability of rule discovery with $|\mathcal{D}|$, and Bayesian parameters $|\Theta|$ and $|T|$.

(1) Varying number of tuples in $\mathcal{D}$. Fixing the attribute sampling ratio m at 0.2, 0.4, 0.6, 0.8 and 1.0, we varied the tuple ratio n across 0.2, 0.4, 0.6, 0.8 and 1.0 via random tuple sampling, and tested the runtime and maximum memory usage of OHunt on the Shuttle dataset using the RF classifier. As shown in Figure 2(o), (a) the runtime and memory usage of uOMRs and sOMRs increase nearly linearly with n. (b) For both outlier detection and correction, sOMRs take longer and consume more memory than uOMRs. For instance, when both row and column sampling ratios are set to 1.0, uOMRs take 9.184s and use 39.231 MB, while sOMRs take 68.734s and use 65.870 MB.

(2) Varying number of attributes in $\mathcal{D}$. Fixing the tuple sampling ratio n at 0.2, 0.4, 0.6, 0.8 and 1.0, we varied the attribute ratio m across 0.2, 0.4, 0.6, 0.8 and 1.0 on the DryBean dataset using Softmax. As shown in Figure 2(p), (a) the runtime and memory usage of OHunt increase almost linearly with m; and (b) sOMRs use more time and memory than uOMRs for better accuracy, with $\text{acc}(\mathcal{M}, \mathcal{D}_c) = 0.972$ for sOMRs and $\text{acc}(\mathcal{M}, \mathcal{D}_c) = 0.951$ for uOMRs.

(3) Rule discovery. We tested the scalability of OMR discovery algorithm OLeaner, by varying (a) the average size $|\Theta|$ of configurations Θ, (b) the total number $|T|$ of tasks T, and (c) the data size $|\mathcal{D}|$ on Flights. (a) Fixing $|T|$ at 10 and $|\mathcal{D}|$ at 20,000, while varying $|\Theta|$ at 3, 4, 5, 6 and 7, the runtime of OLeaner ranges over 123.62, 112.70s, 122.28s, 112.88s and 115.28s. This shows that OLeaner is insensitive to $|\Theta|$, since it always searches for the optimal configuration within Θ. (b) Fixing $|\Theta|$ at 5 and $|\mathcal{D}|$ at 20,000, while varying $|T|$ at 10, 15, 20, 25 and 30, OLeaner takes 122.28s, 181.52s, 248.62s, 297.84s and 370.22s, respectively. This shows that OLeaner scales nearly linearly with $|T|$, since it sequentially executes tasks in T and leverages information from previous ones. (c) Fixing $|\Theta|$ at 5 and $|T|$ at 5, while varying $|\mathcal{D}|$ across 10,000, 20,000, 30,000, 40,000 and 50,000, OLeaner takes 11.96s, 61.14s, 154.82s, 292.12s and 477.81s, and finds 83, 165, 80, 87 and 144 useful OMRs, respectively. The runtime of OLeaner grows approximately linearly with $|\mathcal{D}|$.

The runtime of OLeaner scales well with $|\Theta|$, $|T|$ and $|\mathcal{D}|$, primarily due to the precise objective function in Bayesian optimization, which reduces irrelevant predicate selection and facilitates rapid convergence to the optimal Bayesian hyperparameters.

In contrast, we tested two SOTA levelwise DCFinder [119] and PRMiner [53] for discovering DCs [11] and REEs [55], respectively; as remarked in Section 4, the baselines are unable to determine hyperparameters and mine OMRs. We find that the baselines mined very few helpful rules when varying $|\mathcal{D}|$ across 10,000, 20,000, 30,000, 40,000 and 50,000 on Flights. (1) DCFinder found a total of 9,509 rules at support of $0.00002|\mathcal{D}|$ and confidence of 0.95. It took from 4,258s to 8,463s on five datasets on average. (2) When we set the support to $0.00002|\mathcal{D}|$ and confidence to 0.95, PRMiner mined 991 REEs, and took from 385.24s to 1941.77s on the five datasets. Worse yet, all the REEs (resp. DCs) discovered by PRMiner (resp. DCFinder) are logic reasoning rules, and most of which are not very useful. All the helpful REEs and DCs are also found by OLeaner, and their contribution to the accuracy improvement of classifiers $\mathcal{M}$ is at most 0.4%. These justify the need for the meta-learning approach for discovering OMRs, as evidenced by both its efficiency/scalability in rule discovery and its effectiveness in improving $\mathcal{M}$'s accuracy.

Summary. We find the following. (1) In real-life datasets, 14.08% of samples are outliers on average, among which 80.33% and 19.67% are good and bad, respectively, and among the bad ones 64.72% are ugly. OHunt accurately detects ugly outliers, with average F1-scores of 0.966 and 0.996 for uOMRs and sOMRs on SVM, respectively, 73.6% higher than the baselines, up to 78.4%. (2) By fixing ugly outliers with OMRs, OHunt improves F1-scores of $\mathcal{M}$ by 6.9% on average, up to 21.6%, 6.6% better than the baselines. (3) OHunt is robust to local, global, dependency and cluster noise. Its recall for ugly outliers is 0.656 even with 30% dependency noise. After fixing ugly outliers, it improves the F1-score of $\mathcal{M}$ by 13.2% on average, up to 17.5% on 20 noisy datasets. (4) The combination of logic reasoning and ML/function predicates in OMRs improves the F1-score of ML classifiers $\mathcal{M}$ by 0.4% and 3.9% on average, compared to using each alone. (5) OHunt is scalable in outlier detection and fixing, with runtime and memory consumption increasing almost linearly with the number of tuples and attributes. (6) OLeaner scales almost linearly with $|\mathcal{D}|$ and $|T|$, and is insensitive to $|\Theta|$.

7 RELATED WORK

We categorize the related work as follows.

Outlier detection. Such methods can be classified as follows [29, 39, 74, 74, 135]. (1) Unsupervised methods find outliers by heuristically calculating their score of outlyingness; *e.g.,* [138] distinguishes outliers from normal data by identifying a decision boundary; [102] progressively isolates outliers with random trees; [135] measures the similarity between data points and their neighbors; [74] adopts clustering; [98] (resp. [99]) employs empirical copula models (resp. cumulative distribution functions) to estimate outliers; and [28] suggests data-driven detector selection. (2) Semi-supervised methods train neural-network-based detection models [114, 117, 170] on partially labeled datasets [70]; *e.g.,* [114] adopts random distance-based detectors; [117] computes statistical deviation scores; [170] integrates DAGMM's architecture [173] with DevNet's deviation loss; [169] enriches feature representations by unsupervised detectors, and concatenates them to the original features for classification in an augmented feature space. (3) Supervised methods train outlier detectors [17, 23, 29, 35, 67, 129] using fully labeled data [70].

There has also been rule-based outlier detectors for relations, *e.g.,* decision trees [16, 125], iterative rule generation [10, 34], if-then-elseif rules [155], and Explanation Table [44]. However, these detectors can hardly be integrated with ML models, and their embedded functions for classification are highly customized, making it hard to extend their predicates to various types of outliers [40].

Departing from the prior methods, this work studies the impact of outliers on the downstream classification task. (a) Based on the impact, we categorize outliers as good, bad or ugly, and handle them differently. (b) We unify logic reasoning and ML predictions by embedding ML outlier-detectors and loss/statistical functions as predicates in OMRs, to improve the detection accuracy. (c) Besides, we reduce FPs and FNs of ML models, and explain ML predictions.

Outlier correction. Such methods are classified as follows. (1) Elimination methods directly remove the detected outliers [24, 47, 90, 102, 138]. This approach is effective when outliers have a low rate and are not representative [8]. (2) Statistics-based methods estimate replacement values for outliers [8], such as percentile-based estimation [83], deterministic-function-based transformation [149], and Bayesian statistics [64]. (3) ML-based methods, *e.g.,* [43] learns the joint distribution of clean data and fixes outliers by maximizing a posteriori [19] inference via generative models, [152] reconstructs the original data and replaces outliers accordingly, and [157] builds an uncertainty model to estimate replacement of outliers.

There have also been work on data cleaning for AI [1, 56, 57, 68, 93–95, 104], by error detection, correction and data augmentation.

In contrast, (a) we distinguish outliers and fix only those that have negative impact on the downstream ML classifiers. (b) We unify logic reasoning and ML models to correct ugly outliers. We fix ugly outliers via "deep cleaning" by chasing training data with OMRs, which is Church-Rosser and generates fixes as the logical consequences of rules, ground truth and ML predictions. (c) We also reference external data sources for accurate fixes. (d) To measure the effectiveness of data cleaning for downstream ML tasks on imbalanced data, we suggest ROC-AUC, AP and CD plots as criteria.

Rule discovery. The prior methods are classified as follows. (1) Levelwise search, *e.g.,* [81] and [113] for mining FDs; [51] and [65] for CFDs; and [142] for MDs. (2) Depth-first search, *e.g.,* [4, 161] for FDs, [51] for CFDs, and [33, 119] for DCs using evidence sets. (3) Hybrid, *e.g.,* [137] combines levelwise search with depth-first search to mine MDs, and [126] mines CFDs by integrating FD mining and itemset mining. (4) ML-based methods, *e.g.,* inductive learning [61], reinforcement learning [53], and generative models [48].

In contrast, we propose an approach by modeling OMR discovery as a meta-learning problem, where the objective is to improve the classification accuracy of a given ML model. Then we resolve this meta-learning problem via Bayesian optimization to determine hyperparameters in function/ML predicates and skip exponential enumeration, without using classical support and confidence.

8 CONCLUSION

This paper aims to reduce the negative impact of outliers on $\mathcal{M}$. (1) We classify outliers into the good, the bad and the ugly, where only the ugly ones sabotage the classification accuracy. (2) We propose OMRs to detect and fix ugly outliers, by unifying logic reasoning, ML predicates and function predicates to reveal the insights of $\mathcal{M}$. (3) We develop OHunt to prepare cleaned $\mathcal{D}_c$ for $\mathcal{M}$ by fixing ugly outliers. (4) We propose a meta-learning-based rule discovery algorithm. (5) We provide a "deep cleaning" algorithm for recursively detecting and fixing ugly outliers, with accuracy guarantees. Experiments show that OHunt is effective in practice.

One topic for future work is to extend OHunt to ML models beyond classifiers. Another topic is to study the impact of outliers on the fairness and robustness of ML models, besides the accuracy.

REFERENCES

[1] Mohamed Abdelaal, Christian Hammacher, and Harald Schöning. 2023. REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines. In *EDBT*. OpenProceedings.org, 499–511.

[2] Mohamed Abdelaal, Rashmi Koparde, and Harald Schöning. 2023. AutoCure: Automated Tabular Data Curation Technique for ML Pipelines. In *aiDM@SIGMOD*. ACM, 1:1–1:11.

[3] Salisu Mamman Abdulrahman, Pavel Brazdil, Jan N van Rijn, and Joaquin Vanschoren. 2018. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine learning* 107 (2018), 79–108.

[4] Ziawasch Abedjan, Patrick Schulze, and Felix Naumann. 2014. DFD: Efficient functional dependency discovery. In *CIKM*. 949–958.

[5] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[6] Charu C Aggarwal and Charu C Aggarwal. 2013. Applications of Outlier Analysis. *Outlier analysis* (2013), 373–400.

[7] Charu C Aggarwal and Saket Sathe. 2015. Theoretical foundations and algorithms for outlier ensembles. *ACM SIGKDD Explorations Newsletter* 17, 1 (2015), 24–47.

[8] Herman Aguinis, Ryan K Gottfredson, and Harry Joo. 2013. Best-practice recommendations for defining, identifying, and handling outliers. *Organizational research methods* 16, 2 (2013), 270–301.

[9] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-Generation Hyperparameter Optimization Framework. In *SIGKDD*. 2623–2631.

[10] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. 2018. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research* 18, 234 (2018), 1–78.

[11] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.

[12] Sébastien M R Arnold, Praateek Mahajan, Debajyoti Datta, Ian Bunner, and Konstantinos Saitas Zarkias. 2020. Learn2learn: A Library for Meta-Learning Research. (2020). arXiv:2008.12284 [cs.LG] http://arxiv.org/abs/2008.12284

[13] Konrad Banachewicz and Luca Massaron. 2022. *The Kaggle Book: Data analysis and machine learning for competitive data science*. Packt Publishing Ltd.

[14] Xianchun Bao, Zian Bao, Qingsong Duan, Wenfei Fan, Hui Lei, Daji Li, Wei Lin, Peng Liu, Zhicong Lv, et al. 2024. Rock: Cleaning Data by Embedding ML in Logic Rules. In *SIGMOD (industrial track)*.

[15] V Barnett. 1994. Outliers in statistical data. *John Wiley & Sons Google Scholar* 2 (1994), 705–708.

[16] Mridula Batra and Rashmi Agrawal. 2018. Comparative analysis of decision tree algorithms. In *Nature Inspired Computing: Proceedings of CSI 2015*. Springer, 31–36.

[17] Thomas Bayes. 1991. An essay towards solving a problem in the doctrine of chances. 1763. *MD computing: Computers in medical practice* 8, 3 (1991), 157–171.

[18] Liron Bergman and Yedid Hoshen. 2020. Classification-based anomaly detection for general data. *arXiv preprint arXiv:2005.02359* (2020).

[19] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.

[20] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Annual Workshop on Computational Learning Theory*. 144–152.

[21] Azzedine Boukerche, Lining Zheng, and Omar Alfandi. 2020. Outlier detection: Methods, models, and classification. *ACM Computing Surveys (CSUR)* 53, 3 (2020), 1–37.

[22] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. 2003. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning* 50 (2003), 251–277.

[23] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[24] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: Identifying density-based local outliers. In *SIGMOD*. 93–104.

[25] John S Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*. Springer, 227–236.

[26] Adam D Bull. 2011. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research* 12, 10 (2011).

[27] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. 2016. On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study. *Data mining and knowledge discovery* 30 (2016), 891–927.

[28] Lei Cao, Yizhou Yan, Yu Wang, Samuel Madden, and Elke A Rundensteiner. 2023. AutoDo: Automatic outlier detection. *Proc. ACM Manag. Data* 1, 1 (2023), 1–27.

[29] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-loop Outlier Detection. In *SIGMOD*.

[30] Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. 2021. Bias in Machine Learning Software: Why? How? What to do? *CoRR* abs/2105.12195 (2021). arXiv:2105.12195 https://arxiv.org/abs/2105.12195

[31] Tianqi Chen and Carlos Guestrin. 2016. XFBoost: A scalable tree boosting system. In *SIGKDD*. 785–794.

[32] Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Vol. 1. IEEE, 539–546.

[33] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *PVLDB* 6, 13 (2013), 1498–1509.

[34] William W Cohen. 1995. Fast effective rule induction. In *Machine learning proceedings 1995*. Elsevier, 115–123.

[35] Corinna Cortes. 1995. Support-Vector Networks. *Machine Learning* (1995).

[36] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[37] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *International Conference on Machine Learning*. 233–240.

[38] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research* 7 (2006), 1–30.

[39] Yuhao Deng, Deng Qiyan, Chengliang Chai, Lei Cao, Nan Tang, Ju Fan, Jiayi Wang, Ye Yuan, and Guoren Wang. 2024. IDE: A System for Iterative Mislabel Detection. (2024), 500–503. https://doi.org/10.1145/3626246.3654737

[40] Yuhao Deng, Yu Wang, Lei Cao, Lianpeng Qiao, Yuping Wang, Jingzhe Xu, Yizhou Yan, and Samuel Madden. 2024. Outlier Summarization via Human Interpretable Rules. *PVLDB* 17, 7 (2024), 1591–1604.

[41] D. Dua and C. Graff. 2019. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml Irvine, CA: University of California, School of Information and Computer Science.

[42] Richard O Duda, Peter E Hart, et al. 1973. *Pattern classification and scene analysis*. Vol. 3. Wiley New York.

[43] Simao Eduardo, Alfredo Nazábal, Christopher KI Williams, and Charles Sutton. 2020. Robust variational autoencoders for outlier detection and repair of mixed-type data. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4056–4066.

[44] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and informative explanations of outcomes. *PVLDB* 8, 1 (2014), 61–72.

[45] Nidula Elgiriyewithana. 2024. Apple Quality Dataset. https://doi.org/10.34740/kaggle/dsv/7384155

[46] Andrew Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. 2015. A meta-analysis of the anomaly detection problem. *arXiv preprint arXiv:1503.01158* (2015).

[47] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, Vol. 96. 226–231.

[48] Lihang Fan, Wenfei Fan, Ping Lu, Chao Tian, and Qiang Yin. 2024. Enriching Recommendation Models with Logic Conditions. *Proc. ACM Manag. Data* (2024).

[49] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.

[50] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.

[51] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2010. Discovering conditional functional dependencies. *TKDE* 23, 5 (2010), 683–698.

[52] Wenfei Fan, Liang Geng, Ruochun Jin, Ping Lu, Resul Tugey, and Wenyuan Yu. 2022. Linking Entities across Relations and Graphs. In *ICDE*. IEEE, 634–647.

[53] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*. ACM, 384–398.

[54] Wenfei Fan, Ruochun Jin, Ping Lu, Chao Tian, and Ruiqi Xu. 2022. Towards event prediction in temporal graphs. *PVLDB* 15, 9 (2022), 1861–1874.

[55] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying Logic Rules and Machine Learning for Entity Enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).

[56] Anna Fariha, Ashish Tiwari, Alexandra Meliou, Arjun Radhakrishna, and Sumit Gulwani. 2021. CoCo: Interactive Exploration of Conformance Constraints for Data Understanding and Data Cleaning. In *SIGMOD*. ACM, 2706–2710.

[57] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *NIPS*. 2962–2970.

[58] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. 2018. Scalable meta-learning for Bayesian optimization. *stat* 1050, 6 (2018).

[59] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*. PMLR, 1126–1135.

[60] Ronald A Fisher. 1922. On the interpretation of χ 2 from contingency tables, and the calculation of P. *Journal of the royal statistical society* 85, 1 (1922), 87–94.

[61] Peter A Flach and Iztok Savnik. 1999. Database dependency discovery: A machine learning approach. *AI communications* 12, 3 (1999), 139–160.

[62] Peter I Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[63] Johannes Fürnkranz and Johann Petrak. 2001. An evaluation of landmarking variants. In *ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*. 57–68.

[64] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. 1995. *Bayesian data analysis*. Chapman and Hall/CRC.

[65] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB* 1, 1 (2008), 376–390.

[66] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. 2017. Google Vizier: A service for black-box optimization. In *SIGKDD*. 1487–1495.

[67] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* 34 (2021), 18932–18943.

[68] Rihan Hai, Christos Koutras, Andra Ionescu, Ziyu Li, Wenbo Sun, Jessie van Schijndel, Yan Kang, and Asterios Katsifodimos. 2023. Amalur: Data Integration Meets Machine Learning. IEEE, 3729–3739.

[69] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM SIGMOD Record* 29, 2 (2000), 1–12.

[70] Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. 2022. ADBench: Anomaly detection benchmark. *Advances in Neural Information Processing Systems* 35 (2022), 32142–32159.

[71] James A Hanley and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36.

[72] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE international conference on computer vision*. 1026–1034.

[73] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*. 770–778.

[74] Zengyou He, Xiaofei Xu, and Shengchun Deng. 2003. Discovering cluster-based local outliers. *PATTERN RECOGNITION LETTERS* (2003), 1641–1650. Issue No.9-10.

[75] Ahmed Helal, Mossad Helali, Khaled Ammar, and Essam Mansour. 2021. A demonstration of KGLac: A data discovery and enrichment platform for data science. *PVLDB* 14, 12 (2021), 2675–2678.

[76] Mauricio A Hernández and Salvatore J Stolfo. 1995. The merge/purge problem for large databases. *ACM Sigmod Record* 24, 2 (1995), 127–138.

[77] Aidan Hogan and Aidan Hogan. 2020. SPARQL query language. *The Web of Data* (2020), 323–448.

[78] John H Holland. 1992. Genetic algorithms. *Scientific American* 267, 1 (1992), 66–73.

[79] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression*. John Wiley & Sons.

[80] Hao Huang, Hong Qin, Shinjae Yoo, and Dantong Yu. 2014. Physics-based anomaly detection defined on manifold space. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9, 2 (2014), 1–39.

[81] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal* (1999).

[82] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2014. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*. PMLR, 754–762.

[83] Boris Iglewicz and David C Hoaglin. 1993. *Volume 16: How to detect and handle outliers*. Quality Press.

[84] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: A review. *Data mining and knowledge discovery* 33, 4 (2019), 917–963.

[85] Kenji Kawaguchi, Leslie P Kaelbling, and Tomás Lozano-Pérez. 2015. Bayesian optimization with exponential convergence. *Advances in neural information processing systems* 28 (2015).

[86] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[87] Hufsa Khan, Muhammad Tahir Rasheed, Shengli Zhang, Xizhao Wang, and Han Liu. 2024. Empirical study of outlier impact in classification context. *Expert Systems with Applications* 256 (2024), 124953.

[88] Rabie El Kharoua. 2024. Students Performance Dataset. https://doi.org/10.34740/KAGGLE/DS/5195702

[89] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.

[90] Edwin M Knox and Raymond T Ng. 1998. Algorithms for mining distance-based outliers in large datasets. In *VLDB*. 392–403.

[91] R Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Morgan Kaufman Publishing* (1995).

[92] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. 2008. Angle-based outlier detection in high-dimensional data. In *SIGKDD*. 444–452.

[93] Sanjay Krishnan, Michael J. Franklin, Ken Goldberg, and Eugene Wu. 2017. BoostClean: Automated Error Detection and Repair for Machine Learning. *CoRR* abs/1711.01299 (2017).

[94] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive Data Cleaning for Statistical Modeling. *PVLDB* 9, 12 (2016), 948–959.

[95] Sanjay Krishnan and Eugene Wu. 2019. AlphaClean: Automatic Generation of Data Cleaning Pipelines. *CoRR* abs/1904.11827 (2019). http://arxiv.org/abs/1904.11827

[96] Meng-Chieh Lee, Shubhranshu Shekhar, Christos Faloutsos, T Noah Hutson, and Leon Iasemidis. 2021. Gen2out: Detecting and ranking generalized anomalies. In *IEEE International Conference on Big Data (Big Data)*. IEEE, 801–811.

[97] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. 2012. Selecting classification algorithms with active testing. In *International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*. Springer, 117–131.

[98] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. 2020. COPOD: Copula-based outlier detection. In *IEEE international conference on data mining (ICDM)*. IEEE, 1118–1123.

[99] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George H Chen. 2022. ECOD: Unsupervised outlier detection using empirical cumulative distribution functions. *TKDE* 35, 12 (2022), 12181–12193.

[100] Boyang Liu, Pang-Ning Tan, and Jiayu Zhou. 2022. Unsupervised anomaly detection by robust density estimation. In *AAAI*, Vol. 36. 4101–4108.

[101] Boyang Liu, Ding Wang, Kaixiang Lin, Pang-Ning Tan, and Jiayu Zhou. 2021. RCA: A deep collaborative autoencoder approach for anomaly detection. In *IJCAI*, Vol. 2021. 1505.

[102] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *International Conference on Data Mining*. IEEE, 413–422.

[103] Huan Liu and Hiroshi Motoda. 2007. *Computational methods of feature selection*. CRC press.

[104] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2020. Picket: Self-supervised Data Diagnostics for ML Pipelines. *CoRR* abs/2006.04730 (2020). https://arxiv.org/abs/2006.04730

[105] Wei-Yin Loh. 2011. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery* 1, 1 (2011), 14–23.

[106] Oded Maimon and Lior Rokach. 2005. *Data mining and knowledge discovery handbook*. Vol. 2. Springer.

[107] Donato Malerba. 1994. Page Blocks Classification. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5J590.

[108] Rafael Martinez-Guerra and Juan Luis Mata-Machuca. 2014. Fault detection and diagnosis in nonlinear systems. *Understanding Complex Systems, Springer International Publishing, Cham* (2014).

[109] Kenta Nakai. 1991. Yeast. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5KG68.

[110] Ali Bou Nassif, Manar Abu Talib, Qassim Nasir, and Fatima Mohamad Dakalbab. 2021. Machine learning for anomaly detection: A systematic review. *IEEE Access* 9 (2021), 78658–78700.

[111] John A Nelder and Roger Mead. 1965. A simplex method for function minimization. *The computer journal* 7, 4 (1965), 308–313.

[112] A Nichol. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999* (2018).

[113] Noel Novelli and Rosine Cicchetti. 2001. Fun: An efficient algorithm for mining functional and embedded dependencies. In *International Conference on Database Theory*. Springer, 189–203.

[114] Guansong Pang, Longbing Cao, Ling Chen, and Huan Liu. 2018. Learning representations of ultrahigh-dimensional data for random distance-based outlier detection. In *SIGKDD*. 2041–2050.

[115] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. 2021. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)* 54, 2 (2021), 1–38.

[116] Guansong Pang, Chunhua Shen, Huidong Jin, and Anton van den Hengel. 2023. Deep weakly-supervised anomaly detection. In *SIGKDD*. 1795–1807.

[117] Guansong Pang, Chunhua Shen, and Anton Van Den Hengel. 2019. Deep anomaly detection with deviation networks. In *SIGKDD*. 353–362.

[118] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In *SIGMOD*.

[119] Eduardo HM Pena, Eduardo C De Almeida, and Felix Naumann. 2019. Discovery of approximate (and exact) denial constraints. *PVLDB* 13, 3 (2019), 266–278.

[120] Tomáš Pevnỳ. 2016. Loda: Lightweight on-line detector of anomalies. *Machine Learning* 102 (2016), 275–304.

[121] Eve Porcello and Alex Banks. 2018. *Learning GraphQL: Declarative data fetching for modern web apps*. " O'Reilly Media, Inc.".

[122] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. 2019. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research* 20, 53 (2019), 1–32.

[123] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Doro-

gush, and Andrey Gulin. 2018. CatBoost: Unbiased boosting with categorical features. *Advances in neural information processing systems* 31 (2018).
[124] Chen Qiu, Timo Pfrommer, Marius Kloft, Stephan Mandt, and Maja Rudolph. 2021. Neural transformation learning for deep anomaly detection beyond images. In *International conference on machine learning*. PMLR, 8703–8714.
[125] J. Ross Quinlan. 1986. Induction of decision trees. *Machine learning* 1 (1986), 81–106.
[126] Joeri Rammelaere and Floris Geerts. 2019. Revisiting conditional functional dependency discovery: Splitting the "C" from the "FD". In *Machine Learning and Knowledge Discovery in Databases: European Conference (ECML PKDD)*. Springer, 552–568.
[127] Shebuti Rayana. 2016. ODDS library. *Stony Brook University, Department of Computer Sciences* (2016).
[128] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?" Explaining the predictions of any classifier. In *SIGKDD*. 1135–1144.
[129] Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958), 386.
[130] Peter J Rousseeuw and Christophe Croux. 1993. Alternatives to the median absolute deviation. *Journal of the American Statistical association* 88, 424 (1993), 1273–1283.
[131] Peter J Rousseeuw and Bert C Van Zomeren. 1990. Unmasking multivariate outliers and leverage points. *Journal of the American Statistical association* 85, 411 (1990), 633–639.
[132] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep one-class classification. In *International conference on machine learning*. PMLR, 4393–4402.
[133] Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. 2019. Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694* (2019).
[134] Fereidoon Sadri and Jeffrey D. Ullman. 1980. The Interaction between Functional Dependencies and Template Dependencies. In *SIGMOD*.
[135] Markus M. Breunig;Hans-Peter Kriegel;Raymond T. Ng;Jörg Sander. 2000. LOF: Identifying density-based local outliers. *SIGMOD Record* (2000), 93–104. Issue No.2.
[136] Saket Sathe and Charu Aggarwal. 2016. LODES: Local density meets spectral outlier detection. In *SIAM international conference on data mining*. SIAM, 171–179.
[137] Philipp Schirmer, Thorsten Papenbrock, Ioannis Koumarelas, and Felix Naumann. 2020. Efficient discovery of matching dependencies. *ACM Trans. on Database Syst. (TODS)* 45, 3 (2020), 1–33.
[138] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13, 7 (2001), 1443–1471.
[139] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.
[140] Wei Shen, Jianyong Wang, and Jiawei Han. 2014. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (2014), 443–460.
[141] Tom Shenkar and Lior Wolf. 2022. Anomaly detection for tabular data with internal contrastive learning. In *International conference on learning representations*.
[142] Shaoxu Song and Lei Chen. 2009. Discovering matching dependencies. In *CIKM*. 1421–1424.
[143] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. 2016. Bayesian optimization with robust Bayesian neural networks. *Advances in neural information processing systems* 29 (2016).
[144] Ashwin Srinivasan. 1993. Statlog (Landsat Satellite). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C55887.
[145] Shan Suthaharan and Shan Suthaharan. 2016. Support vector machine. *Machine learning models and algorithms for big data classification: Thinking with examples for effective learning* (2016), 207–235.
[146] Kevin Swersky, Jasper Snoek, and Ryan P Adams. 2013. Multi-task bayesian optimization. *Advances in neural information processing systems* 26 (2013).
[147] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. 2011. Fast anomaly detection for streaming data. In *International Joint Conference on Artificial Intelligence*. Citeseer.
[148] Srikanth Thudumu, Philip Branch, Jiong Jin, and Jugdutt Singh. 2020. A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data* 7 (2020), 1–30.
[149] John W Tukey. 1977. Exploratory data analysis. *Reading/Addison-Wesley* (1977).
[150] J Vanschoren. 2018. Meta-Learning: A Survey. *arXiv preprint arXiv:1810.03548* (2018).
[151] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
[152] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*. 1096–1103.
[153] Ellen M Voorhees and Dawn M Tice. 2000. Building a question answering test collection. In *SIGIR*. 200–207.
[154] Denny Vrandečić. 2012. Wikidata: A new platform for collaborative data collection. In *International Conference on World Wide Web*. 1063–1064.
[155] Fulton Wang and Cynthia Rudin. 2015. Falling rule lists. In *Artificial intelligence and statistics*. PMLR, 1013–1022.
[156] Hu Wang, Guansong Pang, Chunhua Shen, and Congbo Ma. 2019. Unsupervised representation learning by predicting random distances. *arXiv preprint arXiv:1912.12186* (2019).
[157] Jianwei Wang, Ying Zhang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2024. Missing Data Imputation with Uncertainty-Driven Network. *Proc. ACM Manag. Data* 2, 3 (2024), 1–25.
[158] Hilde JP Weerts, Andreas C Mueller, and Joaquin Vanschoren. 2020. Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588* (2020).
[159] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Learning hyperparameter optimization initializations. In *IEEE international conference on data science and advanced analytics (DSAA)*. IEEE, 1–10.
[160] Kaiwen Wu, Kyurae Kim, Roman Garnett, and Jacob Gardner. 2024. The behavior and convergence of local bayesian optimization. *Advances in neural information processing systems* 36 (2024).
[161] Catharine Wyss, Chris Giannella, and Edward Robertson. 2001. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *DaWaK*.
[162] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. 2023. Deep Isolation Forest for Anomaly Detection. *TKDE* (2023), 1–14.
[163] Hongzuo Xu, Yijie Wang, Songlei Jian, Qing Liao, Yongjun Wang, and Guansong Pang. 2024. Calibrated one-class classification for unsupervised time series anomaly detection. *TKDE* (2024).
[164] Hongzuo Xu, Yijie Wang, Guansong Pang, Songlei Jian, Ning Liu, and Yongjun Wang. 2023. RoSAS: Deep semi-supervised anomaly detection with contamination-resilient continuous supervision. *Information Processing & Management* 60, 5 (2023), 103459.
[165] Hongzuo Xu, Yijie Wang, Juhui Wei, Songlei Jian, Yizhou Li, and Ning Liu. 2023. Fascinating supervisory signals and where to find them: Deep anomaly detection with scale learning. In *International Conference on Machine Learning*. PMLR, 38655–38673.
[166] Xiaodan Xu, Huawen Liu, Li Li, and Minghai Yao. 2018. A comparison of outlier detection techniques for high-dimensional data. *International Journal of Computational Intelligence Systems* 11, 1 (2018), 652–662.
[167] Huimin Zhao and Sudha Ram. 2008. Entity matching across heterogeneous data sources: An approach based on constrained cascade generalization. *Data & Knowledge Engineering* 66, 3 (2008), 368–381.
[168] Liang Zhao, Qingcan Li, Pei Wang, Jiannan Wang, and Eugene Wu. 2020. ActiveDeeper: A model-based active data enrichment system. *PVLDB* 13, 12 (2020), 2885–2888.
[169] Yue Zhao and Maciej K Hryniewicki. 2018. XGBOD: Improving supervised outlier detection with unsupervised representation learning. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.
[170] Yingjie Zhou, Xucheng Song, Yanru Zhang, Fanxing Liu, Ce Zhu, and Lingqiao Liu. 2021. Feature encoding with autoencoders for weakly supervised anomaly detection. *IEEE Transactions on Neural Networks and Learning Systems* 33, 6 (2021), 2454–2465.
[171] Xingquan Zhu and Xindong Wu. 2004. Class noise vs. attribute noise: A quantitative study. *Artificial intelligence review* 22 (2004), 177–210.
[172] Ryszard Zieliński. 1983. PJ Huber; Robust statistics. *Mathematica Applicanda* 11, 23 (1983).
[173] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*.

A IMPLEMENTING OMR PREDICATES

We show how to implement ML and function predicates of OMRs.

A.1 Implementation of ML Predicates

We start with ML predicates, namely, $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ for outlier detection and $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ for determining influential features.

$\underline{\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})}$. We implement $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ using the tabular outlier detection models from [70, 162], and adopt the PReNet detector [165] as $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$. These ML-based models are classified as (1) unsupervised: Deep SVDD [132], REPEN [114], RDP [156], RCA [101], GOAD [18], NeuTraL [124], ICL [141] and DIF [162]; and (2) label-based: DevNet [117], PReNet [116], CART [40, 105], XGBoost [31], RoSAS [164], and SLAD [165], based on the availability of labels of outliers. We experimentally evaluate these outlier detectors using five-fold cross-validation [91] on 18 real-world outlier detection datasets, which are listed in Table 2 and Table 6. We find that PReNet performs the best compared to other unsupervised and label-based outlier detectors.

$\underline{\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})}$. We implement $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ to identify influential features A for $\mathcal{M}$. Feature A is considered influential if it is strongly correlated with the label or is widely involved in model training. We identify such features using both statistical and model-based methods, and take the intersection of the results from both.

(1) Statistical implementation of $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$. We consider both categorical and continuous features. (a) For categorical features, we use Fisher's exact test [60] to evaluate the impact of differences between the categories in the feature columns and the label column. This helps us assess the correlation and importance of the categorical features to the label column, for us to select those features that have a significant impact on the label. (b) For continuous features, we use Local Interpretable Model-agnostic Explanations (LIME) [128] to assess the linear and nonlinear correlation between the continuous feature columns and the label column. We select continuous ones that are ranked high in terms of feature importance, as they are considered to have a significant impact on label. (c) For the label column, it serves as the output (target variable) in a classification task, while the categorical feature columns provide the input for $\mathcal{M}$'s predictions on labels. Thus, the label column naturally plays an influential role in the training process of model $\mathcal{M}$, which serves as a basis for evaluating $\mathcal{M}$'s performance.

We take the union of influential categorical, continuous features and labels to serve as statistically influential features A for $\mathcal{M}$.

(2) Model-based implementation of $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$. We then filter influential features based on their involvement in the training process of $\mathcal{M}$. We first train $\mathcal{M}$ on all the original features and label column in the training set. Then, we iteratively remove one feature from all features and retrain the classifier $\mathcal{M}'$ using the remaining features and label column. After that, by testing the classification accuracy of $\mathcal{M}$ and $\mathcal{M}'$ on the test set and setting a threshold α, we consider features such that $\mathrm{acc}(\mathcal{M}, \mathcal{D}) - \mathrm{acc}(\mathcal{M}', \mathcal{D}') > \alpha$ as influential features for $\mathcal{M}$. Moreover, for certain downstream classifiers such as XGBoost [31], we take ranking based on, *e.g.,* importance scores in scikit-learn, as a basis for identifying influential features.

$\underline{\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)}$. KNN is used to fix the ugly outliers in the dataset. Below we detail the process of how to train and apply KNN for the correction. The training process for KNN includes selecting non-ugly outliers, constructing the training data, setting up the KNN model, and training the model. Non-ugly outliers are those captured by OMRs in Σ_d from the OLeaner algorithm. The features (attributes) of these non-ugly samples are used as input in the training data. The labels and feature data of these samples are known. For the KNN model, the number of neighbors K and appropriate distance metric d need to be set [42]. The optimal value of K can be determined via five-fold cross-validation [91]. During this process, d is chosen as Euclidean distance, Manhattan distance or Cosine distance. In our case, we use Euclidean distance by default. To further refine the selection of K and d, we treat these as discrete hyperparameters in OLeaner's process of learning Σ_f, and optimize them together using Bayesian optimization. Since KNN is an instance-based learning method, it does not require an explicit training process. Instead, it simply stores all the training samples [36] (see Section 5.1 for more details).

Moreover, we elaborate on what association rules need to be mined and explain why we use association rules to assist in label correction. More specifically, FP-growth [69] is applied to uncover strong relationships between the target attribute $t.A$ and other features $t.\bar{B}$ in the ugly outliers detected by OMRs in Σ_d. Association rules with high confidence, support, and lift are selected with default thresholds set to 0.9, $0.1|\mathcal{D}|$, and 3, since they indicate reliable correlations between $t.A$ and $t.\bar{B}$. For each ugly label outlier $t.A$, the corresponding association rules are used to impute the $\mathsf{ugly}(t.A)$ values based on the values of $t.\bar{B}$. The key advantage of using association rules to repair ugly labels is that it captures strong and interpretable patterns between features and labels in $\mathcal{D}$. These patterns are often more reliable than purely statistical or model-based imputation methods [106] in practice.

$\underline{\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})}$. The implementation of $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ includes three parts: (1) training model for entity matching; (2) entity matching process; and (3) data extraction from KB.

(1) The training process for entity matching involves using both the attributes $t.\bar{B}$ from $\mathcal{D}$ and the corresponding features from KB, such as textual descriptions, categories, and other metadata. To facilitate matching, similarity metrics such as string matching, numeric similarity, and categorical similarity, are calculated between the ugly tuple t in $\mathcal{D}$ and entities in KB [140]. ML models, such as decision trees [125], logistic regression [79] or deep learning models like Siamese networks [32], are then trained on labeled pairs of matching and non-matching entities to predict the probability that two entities represent the same real-world entity. The resulting model is used to match tuples in $\mathcal{D}$ with entities in KB, allowing for the extraction of accurate values for correcting ugly outliers in $\mathcal{D}$.

(2) Heterogeneous entity resolution (HER) refers to matching objects from two datasets that refer to the same real-world entity but may differ in structure or content. To resolve differences between $\mathcal{D}$ and KB, we use HER techniques [52]. The HER process identifies common attributes across datasets and uses them for matching. For example, if both $\mathcal{D}$ and KB contain entity names and geographical locations, these can be used to align records. HER involves the following techniques: (a) String matching: techniques such as Levenshtein distance (edit distance) or cosine similarity between string

representations of entity names. (b) Fuzzy matching: for data that may have small errors (e.g., typos in names), fuzzy matching algorithms can identify similar entities. Moreover, (c) attribute overlap: comparing shared attributes like location, date, or other categorical features to identify potential matches [76]. HER can be enhanced using ML models that learn from labeled data to predict whether two entities refer to the same real-world object.

(3) Once entities are matched, the relevant data from KB is extracted and used to correct the ugly outliers in $\mathcal{D}$. The matched entities in KB (denoted as v) provide the values for $t.A$ that need to be fixed. These values could include: (a) textual information (*e.g.,* descriptions, names, or titles), (b) numerical data (*e.g.,* price, rating, quantity) and (c) categorical data (*e.g.,* tags, categories) [167]. KB data can be accessed through query interfaces such as the following: (a) GraphQL [121] allows retrieving structured data efficiently. For example, users can query Wikidata to retrieve an entity's attributes based on its ID or name; (b) SPARQL [77] is used for querying RDF-based knowledge graphs, like Wikidata or DBpedia; and (c) APIs from sources such as Kaggle [13] or other datasets that provide structured data about entities. After successful entity matching, the model $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ uses the features extracted from KB (such as related entity descriptions, property values, etc.) to predict or impute the correct values for ugly outliers $t.A$.

Complexity. We outline complexity analysis of the ML models.

(1) The cost of $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ depends on the outlier detectors. For example, when using the SLAD outlier detector, the time complexity of $\mathcal{M}_{\mathsf{DetO}}(t, A, \mathcal{D})$ is typical $O(m \times n)$, where m is the number of network parameters and n is the number of training samples.

(2) The time complexity of $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ is $O(n \times (t^2 + l))$ when implemented using statistical methods on $\mathcal{M}$, where n is as above, l is cost of predicting each neighboring sample, and t is the feature dimension of training data. When $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ is implemented using the model-based methods on SVM classifier $\mathcal{M}$, the cost of $\mathcal{M}_{\mathsf{DetI}}(R, A, \mathcal{M})$ is $O(n^2 \times t \times (t-1))$, where n and t are as above.

(3) The time complexity of $\mathcal{M}_{\mathsf{FixA}}(\mathcal{D}, t, A, v)$ is $O(n^2)$ with n as defined earlier, which consists of the costs of ML methods and statistical methods. It is dominated by $O(n \log n)$ for KNN and $O(n^2)$ for association rule mining in ML methods. The time complexity is $O(n)$ for both fixing feature values and labels in statistical methods.

(4) The time complexity of $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ is $O(n \times |\mathsf{KB}| + n \times f)$, where $|\mathsf{KB}|$ is the size of external data source KB, and f refers to the average number of features (or attributes) associated with each entity in the knowledge base KB. This includes the cost of entity matching, feature extraction and model prediction. More specifically, the time complexity of entity matching is $O(n \times |\mathsf{KB}|)$, where the matching process requires comparing each sample in $\mathcal{D}$ with the entities in KB to find the most suitable match. Once we match an entity from the knowledge base KB, the time complexity of feature extraction in that entity is $O(n \times f)$. After extracting the features from KB, it is in $O(n)$ to pass these features as input to the model $\mathcal{M}_{\mathsf{FixE}}(\mathcal{D}, t, A, v, \mathsf{KB})$ for prediction.

A.2 Implementation of Function Predicates

We now show how function predicates $\mathsf{outlier}(D, R, t, A, \theta)$, $\mathsf{loss}(\mathcal{M}, \mathcal{D}, t, A)$ and $\mathsf{imbalanced}(D, R, t, A, \delta)$ are implemented.

$\underline{\mathsf{outlier}(D, R, t, A, \theta).}$ We implement $\mathsf{outlier}(D, R, t, A, \theta)$ directly based on its definition with the specified threshold θ, if it is possible to determine θ with reasonable accuracy by incorporating domain knowledge. If it is challenging to specify the parameter θ accurately, we replace the parameter θ with statistical metrics and implement this predicate at a statistical level. We present both methods below.

(1) θ-based implementation for $\mathsf{outlier}(D, R, t, A, \theta)$. First, we sort the values within the domain of attribute A to facilitate comparison. Then, we calculate the absolute difference between each data point and its neighbors. For the first (minimum) value and the last (maximum) value, they are only compared with the second smallest and second largest values, respectively. For middle ones (*i.e.,* those that are neither the minimum nor the maximum), the predicate $\mathsf{outlier}(D, R, t, A, \theta)$ is true if $|t.A - t_{\text{right}}.A|$ and $|t.A - t_{\text{left}}.A|$ both exceed the given threshold θ. If either $|t.A - t_{\text{right}}.A|$ or $|t.A - t_{\text{left}}.A|$ is within the threshold θ, $\mathsf{outlier}(D, R, t, A, \theta)$ is considered false. The criterion for determining outliers for the first and last elements is the same as that used for the middle tuples.

(2) θ-free implementation for $\mathsf{outlier}(D, R, t, A, \theta)$. Since it is difficult to determine the exact value of the threshold θ, we also employ statistical methods to estimate $\mathsf{outlier}(D, R, t, A, \theta)$ without specifying θ. More specifically, (a) we use the modified Z-score method [83] and the 2MAD_e statistical method [130] to identify outliers in the column for A. These methods define the range of normal values based on statistical indicators, and $t.A$ falling outside the range is considered outliers. Since these statistical methods do not require manually setting the threshold θ, the identification of outlier is not influenced by human factors. (b) Additionally, to reduce misclassification by $\mathsf{outlier}(D, R, t, A, \theta)$, we use filter fitting [130] and dist [131] techniques to model the data distribution of values within the domain of attribute A. By using the fitted data distribution, we identify $t.A$ outside the distribution as outliers, thus effectively estimating $\mathsf{outlier}(D, R, t, A, \theta)$. (c) To minimize the probability of missing outlier candidates, we take the union of the outliers detected by the modified Z-score, 2MADe, filter fitting, and dist methods. If a tuple t under attribute A falls within this union, then the predicate $\mathsf{outlier}(D, R, t, A, \theta)$ is true; otherwise, it is false.

$\underline{\mathsf{imbalanced}(D, R, t, A, \delta).}$ Along the same lines as implementing $\mathsf{outlier}(D, R, t, A, \theta)$, we implement $\mathsf{imbalanced}(D, R, t, A, \delta)$ using δ-based and δ-free methods. The former implements this predicate based on its definition and requires specifying the threshold δ, while the latter uses statistical methods and does not require specifying δ.

(1) δ-based implementation for $\mathsf{imbalanced}(D, R, t, A, \delta)$. When the parameter δ can be effectively determined through domain knowledge, we adopt δ-based methods as $\mathsf{imbalanced}(D, R, t, A, \delta)$. More specifically, we first count the occurrences of each distinct value in the domain of A. If the interpolation between the maximum and minimum counts exceeds the threshold δ, then $\mathsf{imbalanced}(D, R, t, A, \delta)$ is true; otherwise, $\mathsf{imbalanced}(D, R, t, A, \delta)$ is false.

(2) δ-free implementation for $\mathsf{imbalanced}(D, R, t, A, \delta)$. When it is difficult to effectively determine the parameter δ, we use the δ-free method to reduce the impact of δ. More specifically, we first normalize the values in the column of attribute A. Next, based on whether A is a categorical or continuous feature, we correspond-

ingly estimate the value of imbalanced(D, R, t, A, δ), as follows. (a) If A is a categorical feature, we first count the number N of distinct values in the domain of A. We then compute the average number of supporting tuples for each distinct value in the domain of A, denoted as N_{mean}, by dividing the total number of tuples in the training set by N. If the number of supporting tuples for $t.A$ is less than N_{mean}, imbalanced(D, R, t, A, δ) is true; otherwise, it is false. (b) If A is a continuous feature, we first bin the values in the A column at fixed intervals (*e.g.,* 0.05) and count the number of tuples supporting each bin. We then calculate the average value N_{mean} of these counts. Next, we determine which bin $t.A$ falls into and the number N of supporting tuples for that bin. If N is less than N_{mean}, imbalanced(D, R, t, A, δ) is true; otherwise, it is false.

loss$(\mathcal{M}, \mathcal{D}, t, A)$. We train the classifier $\mathcal{M}$ on the dataset $\mathcal{D}$ and specify a threshold λ for the loss predicate loss$(\mathcal{M}, \mathcal{D}, t, A)$. The form of loss is determined by the classifier $\mathcal{M}$. For instance, when $\mathcal{M}$ is an SVM classifier, loss$(\mathcal{M}, \mathcal{D}, t, A)$ becomes the hinge loss function. More specifically, the hinge loss function is hinge$(\mathcal{M}, D, t, j) = \max(0, 1 - d(t, y_t) + d(t, j))$, where y_t is the true label of sample t and $d(t, j)$ is the SVM's decision function value for sample t on label j. After training $\mathcal{M}$, if the possible number of labels for a sample is n, we compute the hinge loss functions hinge$(\mathcal{M}, D, t, 1), \cdots,$ hinge$(\mathcal{M}, D, t, n)$ for the sample t across labels 1 to n. The loss function for sample t is computed as loss$(\mathcal{M}, \mathcal{D}, t, A) = \max($hinge$(\mathcal{M}, D, t, 1), \cdots,$ hinge$(\mathcal{M}, D, t, n))$. Here the threshold λ should be set to 1. If loss$(\mathcal{M}, \mathcal{D}, t, A)$ exceeds 1, the decision function value for the true label y_t is less than that for a non-true label j, indicating that sample t is misclassified by $\mathcal{M}$ as label j, thus negatively impacting $\mathcal{M}$. Conversely, if loss$(\mathcal{M}, \mathcal{D}, t, A)$ is less than 1, the true label corresponds to the maximum decision function value, meaning that $\mathcal{M}$ correctly classifies t as y_t, and so t does not negatively affect $\mathcal{M}$. In other words, samples with loss$(\mathcal{M}, \mathcal{D}, t, A) > 1$ are likely ugly outliers harming $\mathcal{M}$, while samples with loss$(\mathcal{M}, \mathcal{D}, t, A) < 1$ are likely to be good outliers or bad outliers with no negative impact on $\mathcal{M}$'s accuracy.

Complexity. We next analyze the cost of the function predicates.

(1) outlier(D, R, t, A, θ). The time complexity of θ-based methods for outlier(D, R, t, A, θ) is $O(nlog(n))$, where n denotes the number of training samples, as we only need to compare $t.A$ with the maximum value A_{max} and minimum value A_{min} in the domain of attribute A, and the cost of sorting the values under attribute A is $O(nlog(n))$. If we implement outlier(D, R, t, A, θ) using θ-free methods, the time complexity is $O(n^2)$, where n is as above. The Z-score and 2MAD$_e$ methods both have a time complexity of $O(n)$. This is because Z-score involves calculating values for each tuple in the training set, and 2MAD$_e$ requires traversing each tuple to compute the median and MAD. In contrast, the time complexity for outlier detection using the filter fitting and dist methods is $O(n^2)$.

(2) imbalanced(D, R, t, A, δ). The time complexity of δ-based methods for imbalanced(D, R, t, A, δ) is $O(n)$, where n is as above. This is because counting the occurrences of each distinct attribute value can be done using a hash table in $O(n)$ time. For δ-free imbalanced(D, R, t, A, δ), the complexity is also $O(n)$, as the main cost of this process is determined by the normalization, bucketing, and counting steps, each of which is $O(n)$.

(3) loss$(\mathcal{M}, \mathcal{D}, t, A)$. The time complexity of loss$(\mathcal{M}, \mathcal{D}, t, A)$ is $O(n^2 \times d)$ when $\mathcal{M}$ is an SVM classifier, where n is the number of tuples in the training data $\mathcal{D}$ and d is the number of features in $\mathcal{D}$.

B COMPLEXITY OF OLeaner AND OFix

(1) OLeaner. OLeaner takes at most $O(l \times T + I \times (m^3 + K))$ time, where l is the size of the initial set of configurations, T is the time to evaluate a configuration, I is the number of iterations before stopping, and K is the number of steps for optimizing the acquisition function, often smaller than m^3. This cost consists of three parts: (1) $O(l \times T)$ is the cost of evaluating the initial set of configurations; (2) $O(I \times m^3)$ is for surrogate model training and (3) $O(I \times K)$ is for acquisition function optimization during each iteration.

(2) OFix. The time complexity of OFix is $O(|U| \times (\tau \times n + |\Sigma|) + n \times m \times l)$, where τ is the number of nearest neighbors considered in the KNN algorithm, $|\Sigma|$ is the number of rules in Σ, n is the number of tuples in $\mathcal{D}$, m is the number of attributes in $\mathcal{D}$, l is the number of vertices in KB, and $|U|$ is the number of ugly outliers to be repaired in U. This complexity includes the training cost of $\mathcal{M}_{\text{FixA}}(\mathcal{D}, t, A, v)$ and $\mathcal{M}_{\text{FixE}}(\mathcal{D}, t, A, v, \text{KB})$, and the cost of the chasing process. Specifically, the training complexity of $\mathcal{M}_{\text{FixA}}(\mathcal{D}, t, A, v)$ is $O(\tau \times n \times |U| + n \times m \times log(m))$, while the complexity for $\mathcal{M}_{\text{FixE}}(\mathcal{D}, t, A, v, \text{KB})$ is $O(n \times m \times l + n \times r)$, where r is the average number of relevant neighbors in KB for ugly$(t.A)$. The complexity for the chasing process is $O(|\Sigma| \times |U| + |U| \times log(|U|))$.

C PROOF OF PROPOSITION 1

We show that OFix is Church-Rosser. The proof has two steps.

(1) *Any chasing sequence is finite.* Intuitively, each chasing step fixes at least one ugly outlier and the set U is finite; hence so is ξ. More specifically, for any terminal chasing sequence $\xi = (\mathcal{F}_0, \ldots, \mathcal{F}_n)$ of $\mathcal{D}$ by (Σ, Γ), we can verify that $n \leq |\mathcal{D}|^2 + |\Gamma||\mathcal{D}|$, as a chase step (a) assigns a constant c from Γ or deduces fixes to $t.A$, which makes at most $|\Gamma||\mathcal{D}|$ steps; or (b) sets of two attributes equal, at most enumerating all tuple pairs ($|\mathcal{D}|^2$). Note that the fixes given by $\mathcal{M}_{\text{FixA}}$ and $\mathcal{M}_{\text{FixE}}$ are values (or mean/median) from either normal samples in $\mathcal{D}$ or values from external KB, while the number of values from KB is bounded by the number of outliers in $\mathcal{D}$, and the size of such values do not exceed outlier values. Thus, the chase sequence introduces limited number of values to Γ and ξ is finite.

(2) *All chasing sequences terminate at the same result.* Assume by contradiction that there exist two terminal chasing sequences $\xi_1 = (\mathcal{F}_0, \mathcal{D}_0) \Rightarrow_{\varphi_1, h_1} (\mathcal{F}_1, \mathcal{D}_1) \Rightarrow \cdots \Rightarrow (\mathcal{F}_{n_1-1}, \mathcal{D}_{n_1-1}) \Rightarrow_{\varphi_{n_1}, h_{n_1}} (\mathcal{F}_{n_1}, \mathcal{D}_{n_1})$ and $\xi_2 = (\mathcal{F}'_0, \mathcal{D}'_0) \Rightarrow_{\varphi'_1, h'_1} (\mathcal{F}'_1, \mathcal{D}'_1) \Rightarrow \cdots \Rightarrow (\mathcal{F}'_{n_2-1}, \mathcal{D}'_{n_2-1}) \Rightarrow_{\varphi'_{n_2}, h'_{n_2}} (\mathcal{F}'_{n_2}, \mathcal{D}'_{n_2})$ of $\mathcal{D}$ with (Σ, Γ), with different results. Note that $\mathcal{D}_{n_1}$ (resp. $\mathcal{D}'_{n_2}$) is generated from $\mathcal{D}$ by employing fixes in $\mathcal{F}_{n_1}$ (resp. $\mathcal{F}_{n_2}$). Since ξ_1 and ξ_2 differ, there exists a variable set $S = \mathcal{F}'_{n_2} \setminus \mathcal{F}_{n_1} \neq \emptyset$. Let $\mathcal{F}_i$ and $\mathcal{F}'_i$ be the first pair of sets including different fixes for ugly$(t.A)$ in ξ_1 and ξ_2, such that $\mathcal{F}'_i$ contains a fix p_0 in T, that is, $\mathcal{F}'_{i-1} = \mathcal{F}_{i-1}$, $p_0 \in \mathcal{F}'_i \setminus \mathcal{F}_i$ and $p_0 \in \mathcal{F}'_{n_2} \setminus \mathcal{F}_{n_1}$. This first pair must exist since ξ_1 and ξ_2 chase starting with the same fix sets $\mathcal{F}_0 = \mathcal{F}'_0$, which are initialized with the same ground truth set Γ. In addition, the validated values added to Γ are incremental, with no existing value in $\mathcal{F}_l$ will

Name	Domain	#Samples	#Features	%Outliers	$\#\lvert\Sigma_d\rvert$	$\#\lvert\Sigma_f\rvert$
Apple [45]	architecture	4,000	8	49.90	33	69
DryBean [13]	architecture	13,611	16	3.84	51	125
Obesity [13]	health	1,000	6	14.30	21	55
Balita [13]	classification	120,999	3	11.42	26	61
Star [13]	astronomy	100,000	17	18.96	69	163
Online [13]	network	40,034	12	25.79	54	188
Student [88]	education	2,392	14	4.47	38	88

Table 6: The other seven real-life datasets

be updated or removed, *i.e.,* $\mathcal{F}_l \subseteq \mathcal{F}_{l+1}$ for $l \in [0, n_1]$ (see Section 5.2). This ensures that after the $i-1$ chase steps prior to $\mathcal{F}'_i$ (resp. $\mathcal{F}_i$), $\mathcal{F}_{i-1} = \mathcal{F}'_{i-1}$ must hold. Thus, $\mathcal{F}_i$ and $\mathcal{F}'_i$ are the first pair of different sets. Assume that $(\mathcal{F}'_{i-1}, \mathcal{D}'_{i-1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}'_i, \mathcal{D}'_i)$ is the chase step in ξ_2 that causes the different value in p_0. We show that $(\mathcal{F}_{n_1}, \mathcal{D}_{n_1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}_{n_1+1}, \mathcal{D}_{n_1+1})$ is also a valid chasing step; thus ξ_1 is not terminal, which contradicts the assumption. Here $\varphi = X \rightarrow p_0$ is an OMR in Σ, $\mathcal{F}_{n_1+1}$ expands $\mathcal{F}_{n_1}$ with the fix $h(t').A$, and updated data $\mathcal{D}_i$ is obtained from $\mathcal{D}_{i-1}$ with $h(t').A$.

To show that $(\mathcal{F}_{n_1}, \mathcal{D}_{n_1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}_{n_1+1}, \mathcal{D}_{n_1+1})$ is a valid chase step, it suffices to prove the following two properties: (1) h'_i is also a valuation of φ'_i in $\mathcal{D}_{n_1}$ and (2) each predicate p in the precondition X of φ'_i is validated by $\mathcal{F}_{n_1}$. If these properties hold, since $h(t').A$ does not appear in $\mathcal{F}_{n_1}$ and h is a valuation of φ in $\mathcal{F}_{n_1}$, then $(\mathcal{F}_{n_1}, \mathcal{D}_{n_1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}_{n_1+1}, \mathcal{D}_{n_1+1})$ is a valid chase step.

Below we outline the key ideas behind the two properties. Formally, these can be verified by induction on the length of ξ_1.

(1) We show that h'_i is a valuation of φ'_i in $\mathcal{D}_{n_1}$. This is because (a) h'_i is a valuation in $\mathcal{D}_{i-1} = \mathcal{D}'_{i-1}$, since the two chasing sequences ξ_2 and ξ_1 still have the same fix values in their prior $i-1$ chase steps. (b) Dataset $\mathcal{D}_{n_1}$ is derived from $\mathcal{D}_{i-1}$ by applying fixes; more specifically, it is by fixing ugly$(t.A)$ guided by $\mathcal{F}_l$ for $l \leq n_1$, without deleting any tuples from $\mathcal{D}_{i-1}$. Hence all the tuples involved in h'_i remain existent in dataset $\mathcal{D}_{n_1}$ (despite possibly changed values). That is, h'_i is still a valuation of OMR φ'_i in dataset D_{n_1}.

(2) Each predicate p in the precondition X of φ'_i is validated in $\mathcal{F}_{n_1}$, since p is validated by $\mathcal{F}'_{i-1}$ in our assumption and $\mathcal{F}'_{i-1} = \mathcal{F}_{i-1} \subseteq \mathcal{F}_{n_1}$. More specifically, we adopt logic reasoning, and "robust" ML models and functions as predicates in precondition X to predict the correct values for fixing ugly outliers and updating $\mathcal{F}_{j-1}$. Indeed, observe the following. First, logic predicates in X remain true in D_{n_1} since their involved values are already in $\mathcal{F}_{j-1}$ and $\mathcal{F}_{j-1} \subseteq F_{n_1}$; this is because $\mathcal{F}$ is incremental, without deleting or updating any values that are already in it in the chase process; hence $\mathcal{F}_{j-1} \subseteq F_{n_1}$. Second, the parameters and values involved in the ML or function predicates of φ'_i are already added to $\mathcal{F}_{i-1} = \mathcal{F}'_{i-1}$ since $(\mathcal{F}'_{i-1}, \mathcal{D}'_{i-1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}'_i, \mathcal{D}'_i)$ is a valid chase step (see chase step in Section 5.2). In addition, since these models are trained before the chase begins, they remain unchanged throughout the chase process. Hence, if an ML/function predicate is true at chase step $i-1$ in ξ_2 and ξ_1, it remains true in step n_1 for ξ_1. That is, all predicates in X are validated in $\mathcal{F}_{n_1}$, and h'_i is a valuation of φ'_i and can be applied to $\mathcal{D}_{n_1}$.

Putting these together, we can see that the chasing sequence ξ_1 can be extended with $(\mathcal{F}_{n_1}, \mathcal{D}_{n_1}) \Rightarrow_{\varphi'_i, h'_i} (\mathcal{F}_{n_1+1}, \mathcal{D}_{n_1+1})$, which contradicts the assumption. Thus the chase is Church-Rosser. □

D EXPERIMENTAL DETAILS

Datasets. As shown in Table 6, we summarize the information for the remaining seven real-life datasets used in our experiments, in addition to the details provided in Table 2.

Baselines. We compared OHunt with 22 outlier detection baselines following [40, 70, 162, 163], which are integrated with outlier correction methods detailed in Exp-2. (1) Unsupervised ones in four categories: (a) Statistical: ECOD [99] (Empirical CDF functions), COPOD [98] (based on copulas), LODA [120] (projection-based), and RCA [101] (using covariance matrices and principal component analysis). (b) Distance-based: ABOD [92] (Angle-based Outlier Detection). (c) Ensemble-based: IForest [102] (with decision trees). (d) Deep leaning: DAGMM [173] (based on autoencoders and Gaussian mixture models), SLAD [165] (self-supervised by contrastive learning), GOAD [18] (generative model-based), DSADD [132] (combining deep neural networks with support vector machines), REPEN [114] (learning latent representations of data), ICL [141] and NeuTraL [124] (using contrastive learning techniques). (2) Semi-supervised: PReNet [116] (by reconstruction error), DevNet [117] (based on deep neural networks and graphical techniques), DSAD [133] and ROSAS [164] (both based on self-supervised learning and deep neural networks). (3) Supervised: CatBoost [123] (by gradient boosting trees), LGBoost [86] (both based on gradient boosting trees), and XGBoost [31] (based on efficient gradient boosting machines, GBM). (4) Rule-based ones include ID3 [40, 125] and CART [40, 105].