Uni-Detect: A Unified Approach to Automated Error Detection in Tables

Pei Wang*
Simon Fraser University
peiw@sfu.ca

Yeye He
Microsoft Research
yeyehe@microsoft.com

ABSTRACT

Data errors are ubiquitous in tables. Extensive research in this area has resulted in a rich variety of techniques, each often targeting a specific type of errors, e.g., numeric outliers, constraint violations, etc. While these diverse techniques clearly improve data quality, it places a significant burden on humans to configure these techniques with suitable rules and parameters for each data set. For example, an expert is expected to define suitable functional-dependencies between column pairs, or tune appropriate thresholds for outlier-detection algorithms, all of which are specific to one individual data set. As a result, users today often hire experts to cleanse only their high-value data sets.

We propose UniDetect, a unified framework to automatically detect diverse types of errors. Our approach employs a novel "what-if" analysis that performs local data perturbations to reason about data abnormality, leveraging classical hypothesis-tests on a large corpus of tables. We test UniDetect on a wide variety of tables including Wikipedia tables, and make surprising discoveries of thousands of FD violations, numeric outliers, spelling mistakes, etc., with better accuracy than existing algorithms specifically designed for each type of errors. For example, for spelling mistakes, UniDetect outperforms the state-of-the-art spell-checker from a commercial search engine.

ACM Reference Format:
Pei Wang and Yeye He. 2019. Uni-Detect: A Unified Approach to Automated Error Detection in Tables. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3319855

*Work done at Microsoft Research.

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3319855

Type of error	Sample methods
Numeric outliers	Median-absolute-deviation [48], Distance-based outliers [57], Density-based outlier factor [24], …
Spelling mistakes	Fuzzy group-by [8, 9], Spell-checkers [1, 6], Knowledge-based [35], …
Uniqueness constraint	Unique-row ratio [37], Unique-value ratio [48], …
FD constraint	Unique-projection ratio [53], Conforming-row ratio [56], Conforming-pair ratio [56], …

Table 1: Existing methods for common errors

1 INTRODUCTION

Data errors in tables are extremely common. Studies suggest that 24% of professionally-produced spreadsheets (by firms like KPMG) have errors [39, 70], and approximately 20% of spreadsheets accompanying scientific publications have quality issues [64]. It is estimated that between 1% to 5% of data cells have errors [70, 74]. Such data errors can have grave financial and societal consequences, as evidenced by the ever-growing list of "horror-stories" [12].

Consulting approach to error-detection. The importance of data quality and the prevalence of data errors has inspired a long and fruitful line of research. The general focus in the literature so far has been on the "*consulting*" scenario, where companies use in-house experts or outside consultants, to clean data sets that are of high business values. These experts can often leverage tools and algorithms developed from the research community to detect different types of quality issues, such as violations of Uniqueness or Functional Dependency (FD) constraints, numeric outliers, spelling mistakes, etc. Sample techniques to detect these quality issues are shown in Table 1.

Automated error-detection as software features. While the consulting approach to error-detection is successful and often a must-have for high-value data sets, the need of expert also makes the process expensive which limits adoption. For the purpose of error-detection, for instance, for *each data set*, human experts are expected to inspect and define what rules (e.g., Uniqueness or FD constraints) should hold, and

what parameters (e.g., z-scores in numeric outliers) should be used, etc.

While the consulting approach is suitable for high-value data sets, we observe that there is also a long tail of use cases and a plethora of less valuable data sets, where hiring data quality experts for per-table customization is not feasible – e.g., think of a mom-and-pop shop that uses Excel spreadsheets to keep track of their sales figures and supplier information. There are in fact millions of spreadsheet users who cannot directly benefit from the availability of data quality experts.

We argue that a *software* approach to error-detection would complement the *consulting* approach, and benefits an entirely different segment of the market. Specifically, we envision an automated error-detection feature that can be embedded in software like Excel, Google Sheets, Tableau, etc. This error-detection feature would scan user data in the background, and flag likely data errors for users to inspect/verify. Such a feature would clearly benefit a broad audience of less-technical users – detecting a missing decimal-point in spreadsheets from a mom-and-pop shop may not be as glamorous as multi-million dollar consulting projects, but is still immensely beneficial given the size of the audience this approach can potentially reach.

We would like to note that the software approach to error-detection is similar in spirit to a trend in the industry called "*self-service data preparation*" [11], which is an area that the research community also actively contributes to (examples include Transform-data-by-example [44, 45, 80], Auto-Join [46, 86], Auto-Split [31, 59], Auto-Type [83], Auto-EM [84], among many other things).

Error-detection: the "APR" desiderata. We postulate that for error-detection, there are three desiderata: *automation*, *high-precision*, and *high-recall* (collectively referred to as APR). We further speculate that with today's technologies, it may be possible to get any two of APR at the same time, but not all three simultaneously – we call this an "APR-conjecture" that is analogous to the CAP-theorem [25].

In the consulting-based approach, the focus has mostly been on high-precision and high-recall. In a software setting, we introduce the new dimension of automation, and argue that the focus should be on automation and high-precision:
(1) Automation is a must, because less-technical users typically do not understand things like constraints or z-scores, they would not be able to program rules or set parameters appropriately for their data sets. Error-detection needs to work out-of-box for any input data sets.
(2) High-precision is also a must. Since users need to inspect errors detected by the system, the vast majority of detection has to be correct, or otherwise a supposedly-intelligent feature will quickly become a nuisance. Note that this is particularly challenging given the diverse variety of possible input data.
(3) Recall. Given our APR-conjecture above, we argue that it would be difficult to achieve high-recall, on top of automation and high-precision (or the consulting business would not need to exist). In practice, having any recall for "free" (automated, with no false-positives) is better than no recall at all, as evidenced by features in existing commercial systems that employ a small number of high-precision rules to detect few errors. Microsoft Excel [3] for example, employs 9 simple but high-precision rules to find errors (shown in Figure 1). We will review related error-detection features in existing commercial systems (Trifacta, OpenRefine, etc.) in Appendix B.

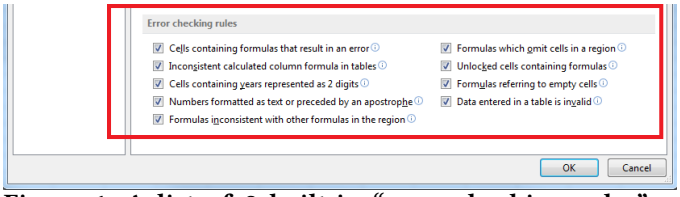


Figure 1: A list of 9 built-in "error-checking rules" used in Excel 2016. Examples include "year represented as 2-digits", "Formula inconsistent with other formulas in the region", etc.

Detect common classes of errors. Given this software setting and the new requirements, we looked at ways to adapt existing methods (currently designed for human experts in consulting scenarios), to automatically detect four common types of errors: numeric-outilers, misspellings, uniqueness and FD constraints.

In order to ensure robustness of algorithms at handling diverse real data, we extracted millions of real tables from the Web and Enterprise spreadsheets, and use them as test cases[1]. We find that existing methods produce many false positives resulting in a low precision, when tested against the large variety of real tables in the wild.

We will review each of these types of errors in turn.

Uniqueness constraint violations. Certain columns are semantically required to be unique (e.g., an ID column), where duplicate values can be flagged as errors. While such constraints are straightforward to enforce once defined by human experts, detecting violations automatically turns out to be non-trivial.

To detect violations of uniqueness, the conventional wisdom in the literature [37, 48] is to flag columns that are almost unique (e.g., 99% unique). While such a method appears intuitive, to our surprise, it produces a large number of false-positives when tested on real tables "in the wild" – our experiments (in Section 4) show that around half of real

[1]We are going through an internal release process for the labeled data. Once approved it will be available at https://github.com/peiwangdb/Uni-detect.

Name	Age	Hometown
Katavelos, Mr. Vassilios G.	19	Agios Sostis
Keane, Mr. Andrew "Andy"	23	Athenry, Galway
Keefe, Mr. Arthur	39	Rahway, New Jersey
Kelly, Mr. James	19	Unknown
Kelly, Mr. James	44	Leixlip, Kildare

(a) False-positive uniqueness error

Published*	Title
2015-04-01	In Perpetuity
2015-05-26	ScareScapes Book Two
2015-05-26	ScareScapes Book One
2015-06-02	AntiBio 2: The Control War
2015-06-30	ScareScapes Book Three
2015-07-07	Reign of Four: I

(b) False-positive uniqueness error

County	2013 Pop	Core Based Statistical Area
Jackson County	61,044	Jefferson, GA Micropolitan Statistical Area
Jasper County	13,601	Atlanta-Sandy Springs-Roswell, GA Metropolitan Statistical Area
Pamlico County	12,953	New Bern, NC Metropolitan Statistical Area
Pasquotank County	39,981	Elizabeth City, NC Micropolitan Statistical Area
Perquimans County	13,601	Elizabeth City, NC Micropolitan Statistical Area

(c) False-positive FD error

City	Country
Sakata	Japan
Sakura, Chiba	Japan
Salamanca	Mexico
Salamanca	Spain
Salavat	Russia
Salé	Morocco

(d) False-pos FD error

Candidate	% of total votes
David Miller	43.26
John Tory	22.12
Barbara Hall	9.21
John Nunziata	5.20
Tom Jakobek	0.76
Douglas Campbell	0.32
Ahmad Shehab	0.30

(e) False-positive outlier

Name	axis
2MASS J01225093-2439505	52
BD+15 2940 b	0.539
COROT-25b	0.0578
Gliese 163 b	0.0607
Gliese 163 c	0.1254
Gliese 504 b	43.5

(f) False-positive outlier

Species	formula
Bromine	Br_2
Bromide	Br^-
Water	H_2O
Hydrogen peroxide	H_2O_2
Sulfur dioxide	SO_2
Sulfur trioxide	SO_3

(g) False-pos spelling-error

Super Bowl	Season
Super Bowl XX	1985
Super Bowl XXI	1986
Super Bowl XXII	1987
Super Bowl XXV	1990
Super Bowl XXVI	1991
Super Bowl XXVII	1992

(h) False-pos spelling-error

Figure 2: Sample cells that are incorrectly detected as errors (in dashed rectangles), when applying existing algorithms to real Wikipedia tables. False positives like these lead to low precision.

Company Name	Sector
Asian Paints Ltd	Consumer Goods
Axis Bank	Banking
GAIL	Energy - Oil & Gas
Grasim Industries	Cement
HCL Technologies	Information Technology
Bharti Airtel	Telecommunication

(a) False-positive: (incorrectly changed to "GMAIL")

County Seat	County
Tahoka	Lynn
Throckmorton	Throckmorton
Tilden	McMullen
Tulia	Swisher
Tyler	Smith

(b) False-positive: (changed to "Trulia", a popular site)

City of license / Market	Station
Mobile, AL – Pensacola, FL	WALA-TV
Kingman	KMOH-TV
Phoenix – Mesa	KTVK ++
	KASW ++
Tucson	KOLD-TV
Little Rock	KARK-TV §§

(c) False-positive: (changed to "Kingsman", a popular movie)

Num.	Nickname	position
1	FEDE	Goalkeeper
12	FABIO	Goalkeeper
23	PIQUERAS	Defender
26	MATTEUS	Defender
5	ELÍAS	Wing
6	GARCIA	Wing

(d) False-positive error: (changed to "FEDEX")

Figure 3: Sample cells incorrectly detected as spelling errors, using Speller from a commercial search engine.

tables from Wikipedia and Web that are over 99% unique are in fact not violations of uniqueness constraints (which amounts to a precision of less than 50%). Figure 2(a) and 2(b), show two such examples. Both tables have hundreds of rows (omitted for space reasons), with only a pair of duplicate values each (marked in dashed rectangles). The percentage of unique values are thus over 99%, which would be predicted as violations using techniques like [37, 48]. Both of these predictions however, are incorrect, as the first table lists all Titanic passengers, where two happen to have the same name; while the second has a list of books, where two happen to have the same publication date. Cases like these are surprisingly common given the sheer number of real tables in the wild – one can imagine that from a large list of people names, a small fraction will inevitably be identical by chance for instance. Detecting errors using assumed uniqueness constraints in such cases is clearly problematic.

Functional-dependency (FD) violations. Functional dependencies are defined over two groups of columns, the left-hand-side (lhs) and right-hand-side (rhs), where values in lhs should uniquely determine the rhs. While there is a large literature on detecting approximate FD *efficiently* (e.g., [51, 56]), in terms of inferring what approximate FD are likely to hold, existing methods in the literature [41, 54, 56] leverage a similar heuristic that FD candidates that almost hold (e.g., for 99% rows) are likely true FDs.

When testing these methods on a large variety of real tables, we again find many false-positive detection (Section 4). Figure 2(c) and 2(d) show two example tables (with hundreds of rows). It is detected that in the first case, because the "population" column almost uniquely determines "statistical area" (for over 99% rows, except the two rows marked that happen to have the same population), this is a likely FD-violation. Similarly in the second case, the "city" column almost uniquely determines "country" (except the two values marked) and is predicted as a violation. Both of these cases are clearly false-positives.

Numeric outliers. Numeric outliers are values that deviate significantly from the underlying distribution (e.g., values that are orders of magnitude larger than the rest) [48, 52]. Existing approaches identify numeric outliers as ones that are k (e.g., 3) *Standard Deviations (SD)* away from the mean [52], or in the case of robust statistics, k *Median-Absolute-Deviation (MAD)* from the median [48]. Recent work on declarative data quality [78] expose k as a parameter for expert users (e.g., developers) to specify. In general, the conventional wisdom is that a larger k indicates a higher likelihood of outliers [48].

While seemingly intuitive, this again leads to frequent false-positives. Figure 2(e) and 2(f) show two examples that are incorrectly identified as outliers. Figure 2(e) lists 44 candidates of an election, most of whom receive less than 1% votes. As such the top candidate seems like a numeric outlier as measured by SD or MAD, which however is a false-positive. Similarly Figure 2(f) lists recently discovered planets. Since most (omitted) have small axis values (below 1), the first value is again incorrectly detected as an outlier.

Spelling mistakes. Spelling mistakes (e.g., "Mississippi" and "Missisipi") can lead to issues in downstream results (e.g., a group-by query that produces two groups representing "Mississippi"). Existing commercial systems such as Paxata [9] and OpenRefine [8] employ a feature known as fuzzy-group-by (shown in Appendix B), which clusters together values in the same column that are syntactically similar (e.g., of Edit-distance 1). Users are expected to select an appropriate fingerprint method for clustering, and then inspect similar values grouped in the same clusters, to tell what are true spelling errors, and what are false-positives (e.g., "H2O" and "H2O2" in Figure 2(g) may be clustered together, so are "Super Bowl XXI" and "Super Bowl XXII" in Figure 2(h)).

For fully-automatic detection of spelling mistakes, one may expect that state-of-the-art Spell-Checkers from commercial search engines (Google [6], or Bing [1]) to perform well. We programmatically invoke Spell-Checker from a commercial search engine to check real Wikipedia tables, which however also produces a large number of false-positives. Figure 3 shows four such examples – a company named "GAIL" is incorrectly changed to "GMAIL", and a city called "Tulia" gets changed incorrectly to "Trulia" (a popular website), etc. In retrospect the poor performance of Speller on tables actually makes sense – values in table cells are often idiosyncratic (e.g., abbreviations, codes, or just employee alias like "JenniferA"), which can all cause troubles for Spellers.

Our approach. In light of the difficulty of automated error-detection, we in this work propose a new and unified approach called UniDetect, which produces promising results when tested for these four types of common errors.

In a nutshell, UniDetect employs principled reasoning of errors using hypothesis-tests, which are based on analysis of large corpus of background tables. Specifically, we propose a *perturbation*-based framework with a "what-if" analysis, using over 100M (mostly clean [50]) tables crawled from the Web, henceforth collectively referred to as **T**. Intuitively, given a new data set D, we hypothetically "perturb" D by removing a sufficiently small subset $O \subset D$, for all possible O. We then compare the likelihood that D, and the "perturbed" $D^{\mathcal{P}}_O = D \setminus O$, are drawn from **T**, denoted as $P(D|\mathbf{T})$ and $P(D^{\mathcal{P}}_O|\mathbf{T})$, respectively. In most cases, because O is small relative to D, $P(D|\mathbf{T})$ and $P(D^{\mathcal{P}}_O|\mathbf{T})$ should not be substantially different. However, if we observe $P(D^{\mathcal{P}}_O|\mathbf{T})$ to be significantly larger than $P(D|\mathbf{T})$, then we can hypothesize (and test) that O is actually an abnormal subset in D, for removing only a small O (surprisingly) makes D statistically more likely. Samples of real errors discovered from Wikipedia tables can be seen in Figure 4.

We show that predictions using the data-driven method correspond well with the human intuition. For instance, in Figure 4(a), UniDetect can intuitively reason that the values involved look like unique code-values (ICAO airport codes), and are thus likely violation of uniqueness. (In comparison, the false-positives in Figure 2(a) and 2(b) are common names/dates that are more likely to have duplicates by chance). In Figure 4(e), UniDetect reasons that "8.716" (which incorrectly uses a "." in place of ",") is likely a true outlier, because removing this single row would make the column a lot more "similar" to tables in **T** (which is not the case for Figure 2(e) and 2(f), as they would require removing multiple rows). Finally, in Figure 4(g), UniDetect detects "Kevin Doeling" and "Kevin Dowling" as likely spelling errors, because they have a very small edit distance (1), relative to other value-pairs in the same column. (In comparison, in Figure 2(g) and 2(h), there are many pairs with small edit distances, making us less confident that the two cases are spelling errors). We show that with suitable featurization, such intuitions can be "learned" from the corpus **T** without explicit human labels.

While we demonstrate the generality of UniDetect in four types of common errors, we recognize that this will likely work for data quality constructs with simple structures that are easy to learn on **T**. For instance it is unlikely to extend to complex constructs such as CFD [23, 71] and general DC [32], as these often encode very specific constraints that only hold on select data, which would require human expertise and understanding of data semantics to be programmed appropriately, and are difficult to be learned from **T** in a general fashion.

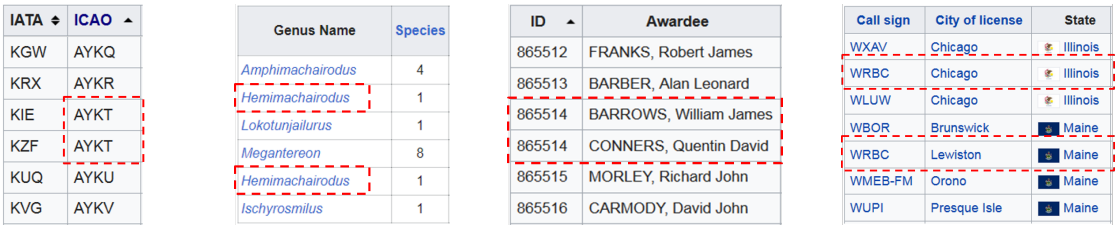
2 THE UNI-DETECT APPROACH

In this section, we will first give a general problem statement before describing the proposed approach UniDetect.

2.1 Problem Statement

One possible approach to automatic error detection is to use the classical supervised machine learning (ML). This, however, would require humans to label a large number of tables *cell-by-cell*, before supervised ML can be trained.

Though we have witnessed recent successes of supervised ML in fields such as image classification and NLP, in part because of large labeling efforts such as ImageNet [58] and SQuAD [75], so far large-scale labeling of tables for supervised ML has not taken off.



IATA	ICAO
KGW	AYKQ
KRX	AYKR
KIE	AYKT
KZF	AYKT
KUQ	AYKU
KVG	AYKV

(a) Uniqueness error

Genus Name	Species
Amphimachairodus	4
Hemimachairodus	1
Lokotunjailurus	1
Megantereon	8
Hemimachairodus	1
Ischyrosmilus	1

(b) Uniqueness error

ID	Awardee
865512	FRANKS, Robert James
865513	BARBER, Alan Leonard
865514	BARROWS, William James
865514	CONNERS, Quentin David
865515	MORLEY, Richard John
865516	CARMODY, David John

(c) FD error

Call sign	City of license	State
WXAV	Chicago	Illinois
WRBC	Chicago	Illinois
WLUW	Chicago	Illinois
WBOR	Brunswick	Maine
WRBC	Lewiston	Maine
WMEB-FM	Orono	Maine
WUPI	Presque Isle	Maine

(d) FD error

Banua	Population
1861	8,011
1871	8.716
1881	9,954
1901	11,895
1911	13,329
1921	11,352
1931	11,709

(e) Outlier error ("8.716" uses "." in place of ",")

Name and surname	Height
Katarina Zec	1.78
Bojana Stevanovic	183
Aleksandra Katanic	175
Jovana Subašić	187
Snezana Bogicevic	177
Kristina Arsenic	189

(f) Outlier error

Author	Director
Joshua Ravetch	Steve Gomer
David Grae	Kevin Doeling
Tom Garrius	Alan Myerson
Sibyl Gardner	James Hayman
Joy Gregory	Kevin Dowling
Antoinette Stella	Rob Morrow

(g) Spelling error

Title	Directed by
"Ratters"	Jet Wilkinson
"Last Seen"	Jet Wilkinson
"No Smoke"	Pino Amenta
"In Harm's Way"	Pino Amanta
"The Hit"	Nicholas Bufalo
"Just Desserts"	Nicholas Bufalo

(h) Spelling error

Figure 4: True-positive errors in Wikipedia tables. Tens of thousands of real errors are detected using UniDetect.

One possible reason is that labeling tables is still prohibitively expensive, because error rate at the cell-level is low. Furthermore, such labeling of tables often require domain-specific understanding of the data (whereas labeling common objects from images is a lot more straightforward). Finally, tables of different characteristics (e.g., in languages other than English, in proprietary enterprise domains, etc.) would typically require additional labeling, making supervised ML for error-detection exceedingly expensive.

These observations motivate us to study error-detection in an *unsupervised setting*, without using human labels. The high-level problem can be stated as follows.

Definition 1. **Unsupervised Error-Detection in Tables**. Given target tables $\mathbf{D} = \{D_j\}$, and classes of data errors $\mathbf{E} = \{E_i\}$. Let $\mathbf{E}(\mathbf{D}) = \bigcup_{i,j} E_i(D_j)$ be errors in $\mathbf{D}$ of type $\mathbf{E}$. Automatically detect errors in $\mathbf{E}(\mathbf{D})$ with high precision.

For the purpose of this work, we instantiate $\mathbf{E}$ as four common classes of errors: {Uniqueness, FD, numeric-outliers, spelling-errors}. As we illustrated earlier, existing methods in the literature typically focus on one class of errors at a time, and can often produce many false-positives when tested on real tables in the wild.

The problem in Definition 1 is general; in this work we propose one possible approach named as UniDetect.

2.2 UniDetect: A Data-driven Approach

In order to enable automatic and unsupervised detection of data errors, in this work we propose a data-driven approach that leverages a large corpus of (mostly clean [50]) tables $\mathbf{T}$. Specifically, we use a large corpus of over 100M web tables extracted from a commercial search engine. Intuitively, this allows us to "learn" statistically what clean tables should look like.

There are two key aspects in our approach: hypothesis-tests using perturbation; and featurization by data-subsetting. We describe the two in turn below.

2.2.1 *Hypothesis Tests: Surprisingness from Perturbation.*

We propose a unified framework that leverages $\mathbf{T}$ to reason about errors in D, based on the idea of *perturbing* D with small changes.

Definition 2. An *ϵ-perturbation* of a table D, is generated by removing a small subset of rows $O \subset D$ of size up to ϵ ($|O| \leq \epsilon$). The perturbed version of D is denoted by $D_O^{\mathcal{P}} = D \setminus O$.

The set of all such perturbations up to size ϵ can be denoted as $\mathcal{P}(D, \epsilon) = \{D_O^{\mathcal{P}} | \forall O \in D, |O| \leq \epsilon\}$.

In UniDetect, the amount of perturbation ϵ can be parameterized as up to ϵ number of rows, or ϵ fraction of rows in D (e.g., 1 row or 1% of the rows). Note that we introduce this hypothetical perturbation O in D, in order to reason whether O may be anomalous relative to D.

We note that using perturbation for error-detection draws an interesting parallel to the well-known *Differential-Privacy (DP)* [66], which uses perturbation to reason about data privacy (privacy can be assured if no query results change significantly before and after perturbing a small fraction of data).

In UniDetect, we detect errors based on hypothesis tests. Specifically, we propose two competing hypotheses.

- H_0 **(Null-hypothesis)**: *D is "normal" (no errors), defined as statistically "like" tables we draw from* $\mathbf{T}$*;*
- H_1 **(Alternative-hypothesis)**: *D is not "normal" due to an abnormal subset $O \in D$; however after removing O, the remaining $D_O^{\mathcal{P}}$ becomes "normal" and statistically "like" tables we draw from* $\mathbf{T}$.

Note that as in statistical hypothesis testing, our default position is to assume D has no error, which is more likely to be true and thus used as the null hypothesis. Unless we have overwhelming evidence in $\mathbf{T}$ to reject the null hypothesis, we assume D to be clean and would not detect it as errors.

With these hypotheses formulated, we can test whether to reject the null-hypothesis H_0 using hypothesis tests [61]. In this work, we use a particular form of hypothesis tests called the *likelihood-ratio (LR) test* [27][2].

Definition 3. In the *likelihood-ratio (LR) test* [27], we estimate the likelihood of two hypotheses based on observed evidence, denoted as $P(H_0|\text{evidence})$ and $P(H_1|\text{evidence})$. The likelihood ratio is simply:

$$LR = \frac{P(H_0|\text{evidence})}{P(H_1|\text{evidence})}$$

Given a fixed significance level α, we can reject the null-hypothesis H_0, if $LR < \alpha$.

In our setup, we use the table in question D, and the corpus $\mathbf{T}$ as the "evidence" for reasoning. We rewrite LR as:

$$LR = \frac{P(H_0|D,\mathbf{T})}{P(H_1|D,\mathbf{T})} = \frac{P(D|H_0,\mathbf{T})}{P(D|H_1,\mathbf{T})}\frac{P(H_0|\mathbf{T})}{P(H_1|\mathbf{T})} \propto \frac{P(D|H_0,\mathbf{T})}{P(D|H_1,\mathbf{T})} \quad (1)$$

Note that the first derivation follows the Bayes Rule [60] and the Chain Rule, while the last step comes from the fact that $\frac{P(H_0|\mathbf{T})}{P(H_1|\mathbf{T})}$ is a fixed constant relating only to the prior probability of having errors in the corpus.

Given Equation (16), we can estimate the target likelihood ratio LR from $\frac{P(D|H_0,\mathbf{T})}{P(D|H_1,\mathbf{T})}$. Specifically, since H_0 states that D is "like" tables drawn from $\mathbf{T}$, we can estimate $P(D|H_0,\mathbf{T})$ as the likelihood of drawing tables "like" D from $\mathbf{T}$, for some definition of "likeness", denoted as $P(D|\mathbf{T})$. On the other hand, H_1 states that after removing some subset O, $D_O^{\mathcal{P}} = D \setminus O$ is "like" tables drawn from $\mathbf{T}$, so we can estimate $P(D|H_1,\mathbf{T})$ as $P(D_O^{\mathcal{P}}|\mathbf{T})$. With these, we can rewrite Equation (16) as:

$$LR = \frac{P(H_0|D,\mathbf{T})}{P(H_1|D,\mathbf{T})} = \frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})}\frac{P(H_0|\mathbf{T})}{P(H_1|\mathbf{T})} \propto \frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})} \quad (2)$$

Recall that in LR test, if $LR \le \alpha$ for some predetermined significance level α, then we could reject the null hypothesis H_0. We would effectively try to minimizie LR over all possible $O \in D$, and would predict the minimizer O^* as the anomaly.

$$O^* = \underset{O\in D, |O|\le\epsilon}{\arg\min} \frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})} \quad (3)$$

The reasoning here is quite intuitive. When we hypothetically perturb D by removing a small subset O, if $P(D_O^{\mathcal{P}}|\mathbf{T}) \gg P(D|\mathbf{T})$, or the likelihood of seeing the resulting $D_O^{\mathcal{P}}$ in $\mathbf{T}$ is substantially larger than that of D, then this is *surprising*, because a small change in D should generally not make $D_O^{\mathcal{P}}$ very different or substantially more likely in $\mathbf{T}$. This would only happen if O is an abnormality that does not initially belong to D, whose removal makes the remaining $D_O^{\mathcal{P}}$ statistically more consistent with the corpus $\mathbf{T}$. Intuitively, $\frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})}$ quantifies the "surprisingness" that a small change O leads to – the more surprising it is, the more likely O is abnormal and erroneous.

Note that $P(D|\mathbf{T})$ and $P(D_O^{\mathcal{P}}|\mathbf{T})$ cannot be evaluated directly, as we are unlikely to draw tables identical to D or $D_O^{\mathcal{P}}$ from $\mathbf{T}$. Instead, we will introduce *metric functions* m, which formalizes the aforementioned notion of "likeness", by mapping tables D and $T \in \mathbf{T}$ to numeric quantities so that $P(D|\mathbf{T})$ and $P(D_O^{\mathcal{P}}|\mathbf{T})$ can be estimated directly.

We describe a concrete example below to explain the idea.

Example 1. [Spelling errors.] We explain the intuition of using perturbation to detect spelling errors.

Existing algorithms (such as the Fuzzy Clustering features in OpenRefine [8] and Paxata [9]) detect spelling errors by finding pairs of values in the same column that are of low distance (e.g., edit-distance = 1). This would detect true spelling errors, such as ("Kevin Doeling" and "Kevin Dowling") in Figure 4(g). Unfortunately however, it also produces a large number of false-positive, such as the value pairs ("Bromine" and "Bromide") and ("H2O" and "H2O2") in Figure 2(g), and ("Super Bowl XXI" and "Super Bowl XXII") in Figure 2(h).

Our perturbation-based reasoning can improve a simple edit-distance-based approach. Suppose we use ϵ-perturbation with $\epsilon = 1$, which removes at most one row. For a column C, let $MPD(C)$ be its *minimum pair-wise edit-distance (MPD)* defined as:

$$MPD(C) = \min_{u\in C, v\in C, u\neq v} Edit(u,v)$$

As we will see, this is our *metric-function* to map $C \in \mathbf{T}$ to numeric quantities.

Observe that in Figure 2(h), removing any row in the suspected pair ("Super Bowl XXI" and "Super Bowl XXII") will not change the *MPD* of the column, because there are many more pairs also with an edit distance of 1. Similarly, in Figure 2(g), when we remove one value in ("H2O" and "H2O2"), the *MPD* stays at 1. When we remove one value in ("Bromine" and "Bromide"), the *MPD* grows from 1 to 2 (the distance between "Sulfur dioxide" and "Sulfur trioxide").

In general, we observe that for certain types of data (e.g., chemical formula, names with roman numerals, etc.), the values are inherently of small distances. Because of this, intuitively it is "normal" to expect that for such columns with small *MPD*, removing one row would not increase the *MPD* much. What would be "surprising" is the case in Figure 4(g), where initially *MPD* is 1, but when we remove one value in ("Kevin Doeling" and "Kevin Dowling"), *MPD* grows significantly to 9 (between "Alan Myerson" and "Rob Morrow"). Intuitively, ("Kevin Doeling" and "Kevin Dowling") is the only pair with small distance in the column, indicating that

[2] The Neyman-Pearson lemma [69] shows that the LR test can in some cases be most statistically efficient among all tests at the same significance level.

one of the values may be misspelled and does not "belong" to the column. We test this hypothesis using data in **T**.

As discussed, we need to estimate $P(D|\mathbf{T})$ and $P(D_O^{\mathcal{P}}|\mathbf{T})$, yet we are unlikely to observe tables identical to D and $D_O^{\mathcal{P}}$ in **T**. In this case we use the function $MPD(D)$ to "describe" D using a number in order to produce the estimate.

Let D be the column with ("Super Bowl XXI" and "Super Bowl XXII") in Figure 2(h). Using MPD, D is described as $MPD(D) = 1$, and $MPD(D_O^{\mathcal{P}}) = 1$. Given over 100M tables and 600M table columns from **T**, we find a total of 2M columns with such properties. Similarly $P(D_O^{\mathcal{P}}|\mathbf{T})$ can be estimated as the fraction of columns whose MPD is 1, and we find 5M such columns out of 600M. Overall, we compute $\frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})}$ to be $\frac{|\{D|D\in\mathbf{T}, MPD(D)=1, MPD(D_O^{\mathcal{P}})=1\}|}{|\{D|D\in\mathbf{T}, MPD(D)=1\}|} = \frac{2}{5}$. The computation is identical for the second column in Figure 2(g).

For the first column in Figure 2(g), we can compute similarly as $\frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})} = \frac{|\{D|D\in\mathbf{T}, MPD(D)=1, MPD(D_O^{\mathcal{P}})=2\}|}{|\{D|D\in\mathbf{T}, MPD(D)=2\}|}$. Suppose we have 3M and 10M such columns, respectively, the ratio is $\frac{3}{10}$.

Finally, for the case in Figure 4(g), we would compute the ratio as $\frac{|\{D|D\in\mathbf{T}, MPD(D)=1, MPD(D_O^{\mathcal{P}})=9\}|}{|\{D|D\in\mathbf{T}, MPD(D)=9\}|}$. It is very uncommon to find $D \in \mathbf{T}$ whose $MPD(D) = 1$ but $MPD(D_O^{\mathcal{P}}) = 9$, and we find a total of 1K such columns in **T**. In comparison there are 50M columns with $MPD(D) = 9$. We compute the ratio as $\frac{1}{50000}$, a very small and surprising value. It can be roughly interpreted as removing "Kevin Doeling" makes the column 50000 times more likely in **T** from an MPD perspective, compared to the original data (where the smallest pair-wise distance is 1 and second smallest is 9). Based on this we can predict ("Kevin Doeling", "Kevin Dowling") as a likely error. □ Note that this approach outperforms a direct application of Fuzzy-Clustering [8] (which blindly picks value pairs with small edit-distances), and the state-of-the-art Speller from a commercial search engine. A formal treatment of spelling errors can be found in Section 3.2.

As we will see, the use of $\frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})}$ applies to other types of errors.

2.2.2 *Featurization by Subsetting.*

The perturbation-based approach introduces a small change to a test data set D, and reasons about the surprisingness of the resulting $D_O^{\mathcal{P}}$ using a large corpus **T**.

While **T** is large which is clearly beneficial for a data-driven method, we find that using the statistics derived from *all* of **T** can be sub-optimal. It is often better to identify tables in **T** that are most relevant or "similar" to the test data D, and only use statitistics derived from this subset. Intuitively, **T** is already big enough so that sparsity is not an issue. Instead, we want to select a subset of **T** most relevant to D for a more accurate analysis.

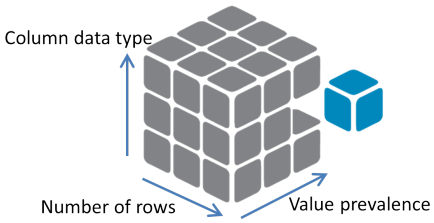


Figure 5: A cube diagram showing possible dimensions for subsetting/featurization of data columns.

Case Number	Party Name Matches
DN35828	RABE, DEBORAH R
D183826	RABE, DEBRA ANN
DN35828	RABE, DONALD BRAD
D130434	RABELLO, DIANA LYN
D128216	RABELLO, DONNA NOEL
D284360	RABIESON, DANNY

Part No.
KV214-310B8K2
MP2492DN
B32671L1272J
MFI341S2500
KV214-310B8K2
P1087

Product Code
S042091
S042093
S041264
S041264
S041908
S042095

Figure 6: Real uniqueness violations from Web tables

For example, for table columns, we could differentiate them based on data types (e.g., string vs. integer vs. floating-point numbers vs. mixed-alphanumeric, etc.). When we reason about test data D, it would be a good idea to only use data in **T** that are of matching type. For example, if D is a string column, then statistics derived from floating-point-number columns in **T** may not be as useful (and may in fact be misleading) compared to that from string columns in **T**. The same is true for other features of D, for example, the number-of-rows (big table vs. small table); or how unique are the values (e.g., long and unique integers, vs. short and common ones; or unique mixed-alphanumeric strings vs. commonly-occurring ones). Such sub-setting can be viewed as a type of featurization.

Figure 5 shows a cube diagram and a few dimensions along which we can featurize data in **T**. For test data D, we can perform analysis using a sub-cube that is the subset of **T** most relevant to D, as opposed to the full **T**. This approach creates disjoint subsets of data, that can be thought of as leaf nodes in decision trees.

Let $S_D^F(\mathbf{T}) = \{T|T \in \mathbf{T}, F(D) = F(T)\}$ be a subset function that selects tables in **T** like D based on a featurization function F. The ratio we would like to compute $\frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})}$ can be rewritten as $\frac{P(D|S(\mathbf{T}))}{P(D_O^{\mathcal{P}}|S(\mathbf{T}))}$ (we omit the superscript and subscript of S when the context is clear).

We use a concrete example of detecting uniqueness violations to intuitively explain this idea.

Example 2. [Uniqueness violation.] Existing approaches (e.g. [37, 48]) detect almost-unique columns (e.g. 99%-unique) as possible violations to uniqueness constraints. While these approaches are intuitive and can detect true errors such as Figure 4(a) and 4(b), they also incorrectly flag many false-positives such as tables in Figure 2(a) and 2(b), making them

unable to meet the high-precision requirement for error-detection in software settings.

An observation here is that the false-positives tend to be relatively common strings (e.g., people's names in Figure 2(a), for names can be identical by chance), as well as number-related strings such as floating-points or date-time (from a large list of numbers, a few can be identical by chance as in Figure 2(b)).

In comparison, for columns with "rare" tokens (measured by an idf-like metric) such as Figure 4(a) and 4(b), or columns with mixed-alphanumeric values (which also likely contain rare tokens), such as ones shown in Figure 6 (taken from real web tables), our human intuition is that these are a lot more likely to be true violations of uniqueness. If we can featurize columns based on these characteristics, and find relevant evidence in data to support such intuition, then it should lead to better predictions.

Recall that in our framework, we use $\frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})}$ to reason about the surprisingness of a perturbation O, and its abnormality. In this case we use the *uniqueness-ratio (UR)*, or the fraction of unique values in D, as the metric-function to "describe" D. Since the duplicate values are suspected errors, for perturbation we can naturally drop duplicate values.

Suppose given a column D with 100 values, where only two of which are duplicates, we can compute $UR(D) = 0.99$, and $UR(D_O^{\mathcal{P}}) = 1$. Given this description of D, we can compute $P(D|\mathbf{T})$ as $\frac{|\{D|D\in\mathbf{T}, UR(D)=0.99, UR(D_O^{\mathcal{P}})=1\}|}{|\mathbf{T}|}$. Similarly $P(D_O^{\mathcal{P}}|\mathbf{T})$ can be computed as $\frac{|\{D|D\in\mathbf{T}, UR(D)=1\}|}{|\mathbf{T}|}$. Overall, the ratio $\frac{P(D|\mathbf{T})}{P(D_O^{\mathcal{P}}|\mathbf{T})}$ can be computed as $\frac{|\{D|D\in\mathbf{T}, UR(D)=0.99, UR(D_O^{\mathcal{P}})=1\}|}{|\{D|D\in\mathbf{T}, UR(D)=1\}|}$.

Note that this calculation makes intuitive sense – out of all columns in $\mathbf{T}$ (~600M), a large fraction of columns are either exactly unique with $UR = 1$ (~150M), or not at all unique (e.g., $UR < 0.5$) (~250M). There is only a small fraction of columns (~200K) that are 99% unique, where the removal of one value would make the column 100% unique. The ratio above can thus be calculated as $\frac{200K}{150M} = \frac{1}{750}$, a surprisingly small number, suggesting that a duplicate value in such cases is suspicious.

In comparison, if we featurize all columns in $\mathbf{T}$, and only use the subset of $S_D(\mathbf{T})$ relevant to D, we would instead compute $\frac{P(D|S_D(\mathbf{T}))}{P(D_O^{\mathcal{P}}|S_D(\mathbf{T}))} = \frac{|\{D|D\in S_D(\mathbf{T}), UR(D)=0.99, UR(D_O^{\mathcal{P}})=1\}|}{|\{D|D\in S_D(\mathbf{T}), UR(D)=1\}|}$.

Suppose we featurize based on value-types and value-prevalence (like in Figure 5). Based on this featurization, for the example in Figure 6, the subset of columns selected in $S_D(\mathbf{T})$ are (1) of type mixed-alphanumeric, and (2) have rare-tokens / low token-prevalence (e.g., on average values in D occur in less than 100 other tables in $\mathbf{T}$). Intuitively, there are likely "ID"-type columns (unique-identifiers, code, etc.), and should likely be unique. In fact, $S_D(\mathbf{T})$ selects a total of 60M columns from $\mathbf{T}$, out of which 30M are unique (or $|\{D|D \in S_D(\mathbf{T}), UR(D) = 1\}| = 30M$). From the subset $S_D(\mathbf{T})$, it is even more uncommon to see columns that are 99% unique, because these ID-like columns are indeed intended to be unique, any columns in $S_D(\mathbf{T})$ that are 99% unique are likely unintended errors. Within $S_D(\mathbf{T})$, we find around 5K such columns, or $|\{D|D \in S_D(\mathbf{T}), UR(D) = 0.99, UR(D_O^{\mathcal{P}}) = 1\}| = 5K$. Combining, the ratio $\frac{P(D|S_D(\mathbf{T}))}{P(D_O^{\mathcal{P}}|S_D(\mathbf{T}))}$ can be computed as $\frac{5K}{30M} = \frac{1}{6000}$, a even smaller number than $\frac{1}{750}$, suggesting a higher likelihood of error for cases in Figure 6. The examples in Figure 4(a) and 4(b) can be computed similarly.

Finally, for the case in Figure 2(a), the featurization suggests that these columns are (1) of type string, and (2) have high token-prevalence. Such columns are often not key columns, and it is common to see such columns having 99% uniqueness yet without quality issues. We compute $\frac{P(D|S_D(\mathbf{T}))}{P(D_O^{\mathcal{P}}|S_D(\mathbf{T}))} = \frac{200K}{20M} = \frac{1}{100}$, suggesting a much less confident prediction compare to previous cases. □

We note that the idea of featurization and data subsetting applies similarly to other types of errors.

2.2.3 *Putting It Together: UniDetect for Error-Detection.*

With intuitions of how perturbation and featurization help to identify errors, we put them together to formally define UniDetect as follows.

Definition 4. **UniDetect: Perturbation-based Error-Detection**. Given a target table D, a large corpus of tables $\mathbf{T}$, and a class of target error E. Using suitable metric-function m, Featurization F, and perturbation $\mathcal{P}$, identify errors $O^* \subset D$ as the minimizer of the LR ratio:

$$O^* = \underset{O\in D, |O|\leq\epsilon}{\arg\min} \frac{P_m(D|S_D^F(\mathbf{T}))}{P_m(D_O^{\mathcal{P}}|S_D^F(\mathbf{T}))} \quad (4)$$

Where ϵ specifies the maximum amount of perturbation, $S_D^F(\mathbf{T}) = \{T|T \in \mathbf{T}, F(D) = F(T)\}$ is the subset function that selects tables like D in $\mathbf{T}$ based on featurization F, and $P_m(D|\mathbf{T})$ uses the metric function m to estimates $P(D|\mathbf{T})$, defined as $P_m(D|\mathbf{T}) = \frac{\{T|T\in\mathbf{T}, m(D)=m(T)|\}}{|\mathbf{T}|}$. Note that for a fixed significance level α, we can use Equation (2) to determine if we should predict O^* as an error.

In the Example 2 for uniqueness violation above, we would instantiate $m()$ as the uniqueness-ratio function $UR()$, F as the featurization with {column-value-types, value-prevalence}, and the perturbation $\mathcal{P}$ as simply dropping duplicate values.

In the Example 1 for spelling errors, we can instantiate $m()$ as the $MPD()$, F similarly as above, and the perturbation $\mathcal{P}$ as dropping a value with the smallest pair-wise distance.

Note that using UniDetect in Definition 4 to solve the general problem in Definition 1 (which is defined for all classes of errors such as {Uniqueness, FD, numeric-outlier, misspelling}), we only need to instantiate UniDetect for

each class of error, and then produce a union of all errors as a ranked list, since each prediction is associated with a (comparable) statistical significance value computed from LR-ratio in Equation (2).

In this work we instantiate UniDetect for each class of errors, using appropriate configurations $(m, F, \mathcal{P})$ that are often inspired by existing methods in the literature. Our aspiration is to model the general UniDetect as a search problem: given a space of metric functions **M**, featurizations **F**, and perturbations **P**, find appropriate configuration $m \in \mathbf{M}$, $F \subset \mathbf{F}$, and $\mathcal{P} \in \mathbf{P}$ to instantiate UniDetect.

Definition 5. **Configuration Search for UniDetect**. Given a target error class E, target tables **D**, training corpus **T**, and a configuration space of metric functions **M**, featurizations **F**, and perturbation **P**. For a fixed significance level α, find the configuration $(m, F, \mathcal{P}) \in (\mathbf{M}, \mathbf{F}, \mathbf{P})$ that maximizes surprising discoveries in UniDetect, defined as

$$\underset{(m,F,\mathcal{P})\in(\mathbf{M},\mathbf{F},\mathbf{P})}{\arg\max} |\{D|D \in \mathbf{D}, \min_{O\in D, |O|\leq\epsilon} \frac{P_m(D|S_D^F(\mathbf{T}))}{P_m(D_O^{\mathcal{P}}|S_D^F(\mathbf{T}))} < \alpha\}| \quad (5)$$

The intuition is that only when $(m, F, \mathcal{P})$ are configured properly, can we find a large number of statistically surprising results. For if it is not the case, e.g. when m and F do not "project" D the right way, we would not observe surprising LR-ratios, because in general a small perturbation $\mathcal{P}$ would not make D and $D_O^{\mathcal{P}}$ substantially different. For example, suppose we use $\mathcal{P}$ that drops duplicate values as in Example 2, but for m instead of using UR we use MPD from Example 1. Clearly this $\mathcal{P}$ would not affect MPD, and would not produce any surprising LR-ratios, resulting in an empty result set in Equation (5), suggesting that this combination is not a good configuration.

Another variant of the search problem is to label tables for errors, and then evaluate predictions of each configuration $(m, F, \mathcal{P})$ using the labeled data. The best configuration can then be selected based on optimization objectives (e.g., maximizing recall, with a precision greater than 0.95%).

We believe that this opens up a new space for designing error-detection methods, with non-trivial open challenges such as controlling False Discovery Rate (FDR) [85][3], because in a naive implementation of Equation (5), we would use the same **T** repeatedly to test a large number of hypotheses.

In this work we take the first step to show that simple instantiation of UniDetect is already promising without exploring the full search problem.

System Architecture. Our UniDetect has two main components, the first is an expensive and offline "learning" component, which performs hypothesis testing on a large corpus **T** as described above. We implemented it as MapReduce-like jobs in order to crunch **T**. Surprising discoveries are pre-computed and "memorized" as rules: e.g., if $MPD(D)$ increases 1 to 9 with a one-row perturbation, then the LR ratio computed on **T** is $\frac{1}{50000}$ (Example 1). This is a surprising result and can be used to detect future errors, so we memorize it using materialization.

At on-line prediction time, we only need to compute relevant metric for a new table D, and perform a lookup to find relevant predictions without computing LR from scratch using **T**. This makes it possible to have real-time predictions at interactive speeds.

Current Limitations. While we show how to instantiate UniDetect for four common classes of errors (and we will show how related work like Auto-Detect [50] is consistent with it in Section 3.5), these errors have relatively simple structures with ample examples in **T** for UniDetect to "learn".

We would like to point out that there are more expressive mechanisms in the literature (e.g., Denial Constraints [33], CFD [23], Matching Dependencies [38]), that can be more complex (e.g., first-order logic). These often require human understanding of data semantics before they can be programmed correctly (e.g., tax rates are different for different jurisdictions). We suspect that it would be difficult to extend the automated reasoning in UniDetect to these formalism in their most general forms.

3 UNI-DETECT FOR DIFFERENT ERRORS

In this section, we demonstrate how UniDetect in Definition 4 can be instantiated to detect seemingly unrelated errors: misspellings, numeric-outliers, uniqueness-violations, and FD-violations.

We note that while these instantiations can be quite involved, we note that as software features they only need to be programmed once *for each type of error*, compared to the consulting approach which would require separate configurations *for each data set.*

3.1 Numeric outliers.

Numeric outliers is a common and important class of data errors that can arise due to entry errors, scale mismatch, etc.

Existing algorithms such as [48] detect values that are outside of k (e.g., 3) standard-deviations (SD) or median-absolute-deviations (MAD) [65] as outliers. Both SD and MAD measure statistical dispersion [27], which is a measure of how "spread out" a distribution is. The SD and MAD of a column C are defined as follows.

$$SD(C) = \sqrt{\frac{\sum_{v\in C}(v - mean(C))^2}{N-1}} \quad (6)$$

$$MAD(C) = median_{v\in C}(|v - median(C)|) \quad (7)$$

Since SD is well known, we show an example of calculating MAD below.

[3]Note that FDR only applies to automatic configuration search, and does not apply to our current instantiation of UniDetect, where configurations are fixed a priori, and each data point in **T** is only used once by a sub-model.

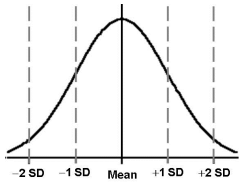


Figure 7: Example SD scores in distributions.

Example 3. Given the column in Figure 2(e), denoted as $C^- = \{43, 22, 9, 5, 0.76, 0.32, 0.30\}$, the median of C^- is 5, and $MAD(C^-) = median(\{38, 17, 4, 0, 4.24, 4.68, 4.70\}) = 4.68$.

For the column in Figure 4(e), denoted as $C^+ = \{$"8,011", "8.716", "9,954", "11,895", "13,329", "11,352", "11,709"$\}$ – note that the second value "8.716" is an outlier that incorrectly uses "." in place of ",". $Median(C^+)$ can be computed as 11352. and the $MAD(C^+)$ is calculated as $median(\{3341, 11344, 1398, 543, 1977, 0, 357\}) = 1398$. □

Given SD/MAD that measures dispersion of a distribution C, the "outlier-ness" of a data point $v \in C$ can then be calculated using the dispersion. For example, the SD-score of a value v is the number of SDs v lies away from the $mean(C)$:

$$score_{SD}(v, C) = \frac{|v - mean(C)|}{SD(C)} \tag{8}$$

Figure 7 shows a probability density function of a distribution. A point that lies at "+1 SD" or "-1 SD" has an outlier-ness SD-score of 1, since it is 1 SD away from the mean, and a point at "+2 SD" or "-2 SD" has SD-score of 2, etc. A larger SD-score indicates a higher degree of outlier-ness.

The MAD-score is defined similarly using $median(C)$:

$$score_{\text{MAD}}(v, C) = \frac{|v - median(C)|}{MAD(C)} \tag{9}$$

Example 4. Continue with Example 3, the value with the highest $score_{\text{MAD}}$ in C^- is the most outlying value 43, where the score is $\frac{43-median(C^-)}{MAD(C^-)} = \frac{43-5}{4.68} = 8.1$.

In the case of C^+, the value with the highest $score_{\text{MAD}}$ is 8.716, which has a score of $\frac{|8.716-median(C^+)|}{MAD(C^+)} = \frac{|8.716-11352|}{1398} = 8.1$, which is a score comparable to C^-. □

Note that these scores are non-parametric (i.e. they do not rely on assumptions of underlying distributions), and are thus broadly applicable. In the influential work by Hellerstine [48], it was shown that MAD from the robust statistics [65] can more reliably predict outliers. In the following discussion we will focus on MAD, but the analysis is equally applicable to SD, or other measures of statistic dispersion (e.g., Interquartile-range/IQR [65]).

We instantiate UniDetect in Definition 4 for numeric-outliers as follows. We use max-MAD of a numeric column C as the metric function m, defined as

$$\text{max-MAD}(C) = \max_{v \in D} Score_{MAD}(v, C) \tag{10}$$

Effectively we use the most outlying value in C to measure the outlier-ness of the C. For perturbation $\mathcal{P}$, we naturally drop the suspected outlier, which is the value with the highest $Score_{MAD}$. Finally, for featurization F, we use (1) data types (integer vs. floating-point numbers, etc.), (2) number of rows (bucketized as: {(0-20] , (20-50], (50-100], (100-500], (500-1000], and (1000-∞)}), and (3) whether logarithm-transform better fits the data [68].

Recall that we find outliers O^* by minimizing LR. Using the aforementioned $(m, F, \mathcal{P})$, we can further rewrite the ratio as $\frac{P_m(D|S_D^F(\mathbf{T}))}{P_m(D_O^{\mathcal{P}}|S_D^F(\mathbf{T}))}$

$$= \frac{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D)=\theta_1, \text{max-MAD}(D_O^{\mathcal{P}})=\theta_2\}|}{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D)=\theta_2\}|} \tag{11}$$

Where $\theta_1 = \text{max-MAD}(C)$ and $\theta_2 = \text{max-MAD}(C_O^{\mathcal{P}})$ for the given test column C, respectively.

Smoothing. Note that $\text{max-MAD}(D) \in \mathbb{R}^+$, and if we plot the probability density of max-MAD in $\mathbf{T}$, using max-MAD as the x-axis and $freq(x) = |\{D|D \in \mathbf{T}, \text{max-MAD}(D) = x\}|$ as the y-axis, we see a highly irregular and non-smooth distribution with significant ups and downs. This is because max-MAD takes a large range of possible values, so that for a specific $x = \text{max-MAD}(C)$ score we may not see tables with that exact x even in a large $\mathbf{T}$. However if we look at the small neighborhood $x \pm \delta$, we will find plenty of tables. This motivates the need to "smooth out" the distribution for a reliable estimate.

There are a couple of different ways to perform smoothing. One approach is to use the classical Kernel Density Estimation (KDE) [72], where kernels functions are used to smooth individual observations. We tested this method and find it to be ineffective, because smoothing parameters need to be empirically tuned (e.g., using cross-validation [43]) for each distribution in $S_D^F(\mathbf{T})$, which often leads to inaccuracy.

We instead propose an alternative smoothing by modifying how data are "described" in Equation (11). Recall that in Equation (11), $P_m(D|S_D^F(\mathbf{T}))$ is interpreted as *"tables in the subset $S_D^F(\mathbf{T})$ that are 'like' D based on metric-function m"*. Instead of interpreting the "likeness" as an exact point-based estimate of *"before perturbation the max-MAD of a table is exactly θ_1, after perturbation the max-MAD is exactly θ_2"*, which suffer from non-smoothness, we instead use *range-based predicates*. We rewrite Equation (11) as $\frac{P_m(D|S_D^F(\mathbf{T}))}{P_m(D_O^{\mathcal{P}}|S_D^F(\mathbf{T}))} =$

$$\frac{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq \theta_1, \text{max-MAD}(D_O^{\mathcal{P}}) \leq \theta_2\}|}{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq \theta_2\}|} \tag{12}$$

Note that in Equation (12), we reformulate $P_m(D|S_D^F(\mathbf{T}))$ as $\frac{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq \theta_1, \text{max-MAD}(D_O^{\mathcal{P}}) \leq \theta_2\}|}{|S_D^F(\mathbf{T})|}$, which reads as *"before perturbation the max-MAD is greater than or equal to θ_1, after perturbation the max-MAD is less than or equal to θ_2"*. Note that this still captures the intuition of surprisingness – an max-MAD score higher than θ_1 before perturbation that drops below θ_2 after perturbation, is more surprising than the exact $\theta_1 \rightarrow \theta_2$.

Theorem 1. [Monotonicity.] Let $r(D)$ be the smoothed ratio defined in Equation (12), the following *monotonicity*

holds for $r(D)$. For any column pair C and C', with θ_1 = max-MAD(C), θ_2 = max-MAD($C_O^{\mathcal{P}}$), θ_1' = max-MAD(C'), θ_2' = max-MAD($C_O'^{\mathcal{P}}$):

$$\theta_1 \geq \theta_1', \theta_2 \leq \theta_2', \Rightarrow r(C) \leq r(C') \tag{13}$$

We note that *monotonicity* is a desirable property, for intuitively it guarantees that a more surprising discovery from perturbation is guaranteed to produce a smaller (more surprising) LR result using the smoothed $r(D)$. Note that this property generalizes to all other error types. A proof of this result can be found in Appendix E.

We use the following example to illustrate smoothed scores.

Example 5. [Numeric outliers.] Continue with Example 4, note that both the max-MAD(C^+) and max-MAD(C^-) have the same score of 8.1, which make them indistinguishable for MAD-based methods such as [48].

In comparison, UniDetect reasons about the likelihood of error using the ratio in Equation (12). Specifically, for C^-, before perturbation max-MAD(C^-) is 8.1 (θ_1^- = 8.1), after perturbation (removing value "43") max-MAD($C_O^{-\mathcal{P}}$) becomes 7.4 (θ_2^- = 7.4). For C^+, before perturbation max-MAD(C^+) is 8.1 (θ_1^+ = 8.1), after perturbation (removing value "8.716") max-MAD($C_O^{+\mathcal{P}}$) becomes 3.5 (θ_2^+ = 3.5).

Putting θ in Equation (12), the ratio for C^+ is:

$$\frac{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq 8.1, \text{max-MAD}(D_O^{\mathcal{P}}) \leq 3.5\}|}{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq 3.5\}|} \tag{14}$$

Note that it is uncommon to see data with max-MAD larger than 8.1 before perturbation, and after perturbation drops to below 3.5, this leads to a small LR ratio.

In comparison, the ratio for C^- is:

$$\frac{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq 8.1, \text{max-MAD}(D_O^{\mathcal{P}}) \leq 7.4\}|}{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq 7.4\}|} \tag{15}$$

Observe that consistent with Theorem 1, the ratio in Equation (14) is guaranteed to be smaller than that in Equation (15), because the denominator in Equation (14) is larger than that of Equation (15), while its nominator is (substantially) smaller. Overall we would find C^+ to be significantly more suspicious than C^- using these scores, and correctly predict "8.716" in C^+ (Figure 4(e)) to be an outlier, while "43" in C^- (Figure 2(e)) is not. □

3.2 Spelling Errors.

We explained intuitions of detecting spelling errors in Example 1. In this section we define it more formally.

We instantiate UniDetect using *minimum pair-wise edit-distance (MPD)* as the metric function m (inspired by Fuzzy-Clustering in OpenRefine [8] and Paxata [9]), defined as:

$$MPD(C) = \min_{u \in C, v \in C, u \neq v} Edit(u, v)$$

The intuition here is that a small MPD indicates likely misspellings. For example, the tables in Figure 2(g), 2(h), 4(g) and 4(h) all have value pairs with *MPD* of 1.

Given that we want to identify misspellings, a natural perturbation $\mathcal{P}$ is to drop a value from the pair with the smallest distance. For featurization F, we use featurization similar to Figure 5, which includes (1) data types, defined as: {string, integer vs. floating-point numbers vs. mixed-alphanumeric}, (2) number of rows, defined as: {(0-20] , (20-50], (50-100], (100-500], (500-1000], and (1000-∞)}, and (3) the average length of the tokens that differ between the MPD pair, again bucketized into ranges ({(0-5] , (5-10], (10-15], (15-20], and (20-∞)}).

While the first two featurizations are straightforward, the last one is specific to misspellings, and is based on the observation that if edit between a pair of values happens on long tokens, it is more likely to be an misspelling (e.g., "Doeling" and "Dowling"), whereas for shorter tokens (e.g., "XXI" and "XXII") it is more likely to be false-positives.

Given these we compute the ratio as:

$$\frac{P_m(D|S_D^F(\mathbf{T}))}{P_m(D_O^{\mathcal{P}}|S_D^F(\mathbf{T}))} = \frac{|\{D|D \in S_D^F(\mathbf{T}), \text{MPD}(D) \leq \theta_1, \text{MPD}(D_O^{\mathcal{P}}) \geq \theta_2\}|}{|\{D|D \in S_D^F(\mathbf{T}), \text{MPD}(D) \leq \theta_2\}|}$$

Where θ_1 = MPD(C) and θ_2 = MPD($C_O^{\mathcal{P}}$) are computed from the given test column C, respectively.

It is worth noting that this computation not only captures our intuition that a column C is suspicious if it initially has a small $MPD(C)$, which when perturbed produces a substantially large $MPD(C_O^{\mathcal{P}})$; it also *quantitatively* measures surprisingness, which makes it directly comparable between an *MPD* increase from 1 to 10, v.s. another *MPD* increase from 3 to 15. UniDetect quantifies all such scenarios in a data-driven manner to accurately predict errors.

3.3 Uniqueness violations.

We informally explained uniqueness in Example 2. More formally, we instantiate it using UniDetect as follows. We use the *uniqueness-ratio (UR)* function as the metric function m. For a column C, $UR(C)$ is defined as $\frac{\text{num-distinct-values}(C)}{\text{num-total-values(C)}}$. The intuition is that a column with a UR close to 1 likely has violations to uniqueness constraints.

For perturbation $\mathcal{P}$ it is natural to drop duplicate values in C. For featurization F, we use: (1) Data types, defined as: {string, integer vs. floating-point numbers vs. mixed-alphanumeric}. (2) Number of rows, defined as: {(0-20] , (20-50], (50-100], (100-500], (500-1000], and (1000-∞)}. (3) The leftness of column C in a table [26, 28], defined as the column position of C counting from the left. (4) The average prevalence of tokens in C, or on average the number of times the tokens in C occur in other tables, defined as:

$$Prev(C) = \text{avg}_{v \in C} \ \text{avg}_{t \in \text{tokenize}(v)} |\{T|T \in \mathbf{T}, v \in T, t \in \text{tokenize}(v)\}|$$

We also bucketize this as {(0-50] , (50-100], (100-1000], (1000-10000], (10000-100000], and (100000-∞)}.

The ratio can be instantiated as:

$$\frac{P_m(D|S_D^F(\mathbf{T}))}{P_m(D_O^{\mathcal{P}}|S_D^F(\mathbf{T}))} = \frac{|\{D|D \in S_D^F(\mathbf{T}), \text{UR}(D) \le \theta_1, \text{UR}(D_O^{\mathcal{P}}) \ge \theta_2\}|}{|\{D|D \in S_D^F(\mathbf{T}), \text{UR}(D) \le \theta_2\}|}$$

Where $\theta_1 = \text{UR}(C)$ and $\theta_2 = \text{UR}(C_O^{\mathcal{P}})$ are computed from the given test column C, respectively.

As we discussed in Example 2, features such as mixed-alphanumeric data-type and low $Prev(C)$ intuitively capture "ID"-like columns (unique identifiers, code, etc.), which predicts violations more accurately.

3.4 FD violations.

FD violations are conceptually similar to Uniqueness, but are defined over two groups of columns. We use the *FD-compliance-ratio (FR)* function as the metric function m. Given a table T, let C_l, C_r be two groups of columns in T, that are the lhs and rhs of an FD, respectively. Let u, v be two rows in D, and $u(C)$ and $v(C)$ be the values of u and v in columns C. The FD-compliance-ratio of FD candidate $(C_l \rightarrow C_r)$ over table D, denoted by $FR_D(C_l, C_r)$, is defined as follows.

$$FR_D(C_l, C_r) = \frac{|\{(u(C_l), u(C_r))|\nexists u, v \in D, u(C_l)=v(C_l), u(C_r)\neq v(C_r)\}|}{|\{(u(C_l), u(C_r))|u \in D)\}|}$$

For instance, for the table in Figure 4(c), the FR("ID", "Awardee") = $\frac{4}{6}$. Like the UR metric function for Uniqueness, an FR closer to 1 indicates likely FD violations. A natural perturbation $\mathcal{P}$ is to drop rows in suspected violations, $\{u|u, v \in T, u(C_l) = v(C_l), u(C_r) \neq v(C_r)\}$. And we use the same featurization F as in Section 3.3.

The ratio can be computed as $\frac{P_m(D|S_D^F(\mathbf{T}))}{P_m(D_O^{\mathcal{P}}|S_D^F(\mathbf{T}))} =$

$$\frac{|\{(C_l, C_r)|D \in S_D^F(\mathbf{T}), C_l, C_r \in D, \text{FR}_D(C_l, C_r) \le \theta_1, \text{FR}_{D_O^{\mathcal{P}}}(C_l, C_r) \ge \theta_2\}|}{|\{(C_l, C_r)|D \in S_D^F(\mathbf{T}), C_l, C_r \in D, \text{FR}_D(C_l, C_r) \le \theta_2\}|}$$

Where $\theta_1 = \text{FR}_T(C_l, C_r)$ and $\theta_2 = \text{FR}_{T_O^{\mathcal{P}}}(C_l, C_r)$ are computed for the given T and $(C_l, C_r) \in T$.

3.5 Compatibility Error in Auto-Detect [50]

A recent work AutoDetect [50] focuses on detecting pattern incompatibility, which is an orthogonal class of data errors. For example, it detects incompatible values such as "2001-Jan-01" vs. "2001-01-01" in same columns, also leveraging statistical analysis on large table corpus.

We show that the basic mechanism of PMI (point-wise mutual information) in AutoDetect is actually consistent with the likelihood-ratio (LR) test in UniDetect, which further illustrates its generality. We give a detailed derivation in Appendix C in the interest of space.

4 EXPERIMENTS

4.1 Datasets

In our experiments, we use two web table corpora extracted from the index of a commercial search engine, and a corpus of enterprise spreadsheet tables extracted from the intranet of a large enterprise.

	total #tables	avg-#columns per-table	avg-#rows per-table
WEB	135M	4.6	20.7
WIKI	3.6M	5.7	18
Enterprise	489K	4.7	2932

Table 2: Summary statistics of table corpora.

• **WEB**. WEB contains a set 135M relational tables extracted from the web. Non-relational and low-quality tables have been filtered by ML-classifiers in a production pipeline [28].

• **WIKI**. WIKI a subset of WEB from the wikipedia.org domain with over 3M tables. As one would expect, WIKI is of high quality since these pages are collaboratively edited by millions of editors.

• **Enterprise**. Enterprise is a collection of 489K spreadsheet tables, extracted from Excel (.xlsx) files, crawled from the intranet of a large enterprise. These tables are substantially larger than web tables, and are often populated directly from enterprise databases (as evidenced by the presence of database connection-strings [2]).

Summary statistics of the corpora can be found in Table 2.

UniDetect uses a large table corpus for statistical reasoning, which is referred to as **T** that is analogous to "training data". In this work we use WEB as the training corpus **T**. Given the large number of tables in WEB, it is likely to cover diverse tables and generate reliable statistics.

In order to test the quality of error detection for different types of errors, we sampled 10% of WIKI, 1% of WEB, and all of Enterprise as our test benchmarks, henceforth referred to as WIKI$^{\mathsf{T}}$, WEB$^{\mathsf{T}}$ and Enterprise$^{\mathsf{T}}$. We execute UniDetect models learned from the WEB *completely unchanged* on these corpora to produce ranked lists of predictions, and compare with those from existing algorithms.

4.2 Methods Compared

We implemented UniDetect and a total of 15 existing methods to generate predictions on WIKI$^{\mathsf{T}}$, WEB$^{\mathsf{T}}$ and Enterprise$^{\mathsf{T}}$, using a production Map-Reduce-like environment.

• **Speller** [1, 6]. Spellers from commercial search engines are trained using large amounts of usage data, and perform spelling corrections at very high accuracy (e.g., precision well over 0.9 [40, 63]). Applying Spellers to detect misspellings in tables is a natural baseline. We programmatically invoke Speller from a commercial search engine to produce a ranked list of predicted misspellings, ordered by confidence scores.

• **Speller (address-only)**. Since not all table data are suitable for Spellers to produce corrections, in this variant we invoke Speller only on the restricted domain of address data, for which Speller is believed to be suitable (to filter down to address data, we only look at columns whose headers are "address", "city", or "location", and manually remove non-address columns during final human evaluation).

• **Fuzzy-Cluster** [8, 9]. Existing systems such as Paxata [9] and OpenRefine [8] use fuzzy-clustering to group together

values in the same column that are within a small distance (e.g., Edit-distance=1), since these are likely misspellings. We simulate this Fuzzy-Cluster feature by producing a ranked list of value pairs within sames columns, ordered first by edit-distance, and then by the length of tokens where the values differ (as values that differ in longer tokens are more likely typos, e.g., "mississippi" vs. "missisippi", compared to shorter ones like "mark" and "mary").

• **Word2Vec** [67] and **GloVe** [73]. Word embedding such as Word2Vec and GloVe produce vector representations of words trained over large text corpus, and were suggested as alternatives to compare with in the review process. We use the Glove model trained over 840B tokens [5], and the Word2Vec model trained over 100B tokens [16]. Words that are out-of-vocabulary (OOV) are predicted as misspelled.

• **Distance-based outlier detection (DBOD)** [57]. DBOD is an outlier detection method proposed for databases, which scores a value v high if a large fraction of values in the same column lie far away from v. For a given column C, we sort values ascendingly to get $\{v_1, v_2, \ldots, v_n\}$, and score the most outlying values (v_1 and v_n) based on their distances to the closest neighbors, normalized by min/max of C. Namely, DBOD(v_1)= $\frac{v_2 - v_1}{v_n - v_1}$, and DBOD($v_n$)= $\frac{v_n - v_{n-1}}{v_n - v_1}$. All predication are again sorted by DBOD scores to produce a ranked list.

• **Local outlier factor (LOF)** [24]. LOF detects outliers using a notion of *local density* [24], and scores value v based on its density. We rank outliers using LOF scores.

• **Max-MAD** [48]. Hellerstein proposes to use MAD scores from robust statistics to detect numeric outliers [48]. This is one of the state-of-the-art approaches for outliers. We compute the MAD score (Section 3.1) of each value in a column, and rank predictions based on MAD scores.

• **Max-SD** [20]. This approach is similar to Max-MAD, but uses the more standard SD score (Section 3.1) instead of MAD score. We again rank by SD scores (larger scores indicate higher likelihood of error).

• **Unique-row-ratio** [37]. The Unique-row-ratio detects approximate uniqueness constraints, using the ratio of distinct values in a column to the total number of rows. Columns with scores close to 1.0 are more likely to be errors.

• **Unique-value-ratio** [48]. Unique-value-ratio is proposed as an improvement to Unique-row-ratio, and is robust to "frequency outliers" (values with high frequencies) [48]. It is defined as the ratio of unique values (values with frequency one) to the total number of distinct values in a column. We rank predictions the same way as Unique-row-ratio.

• **Unique-projection-ratio** [53]. This approach detects approximate FD ($X \rightarrow Y$) in T, using Unique-projection-ratio, defined as $\frac{|\pi_X(T)|}{|\pi_{XY}(T)|}$. We enumerate column pairs and rank predictions based on this score.

• **Conforming-row-ratio** [56]. This is a variant of the Unique-projection-ratio for approximate FDs, which uses Conforming-row-ratio defined as $\frac{|\{u | u \in T, \nexists v \in T : u[X] = v[X], u[Y] \neq v[Y]\}|}{|T|}$, or the ratio of rows conforming to FD to the total number of rows.

• **Conforming-pair-ratio** [56]. In Conforming-pair-ratio, approximate FDs are detected based on the ratio of row-pairs conforming to FD, defined as $\frac{|\{(u,v) | u, v \in T, u[X] = v[X], u[Y] \neq v[Y]\}|}{|T|^2}$.

• **UniDetect.** This is the method proposed in this work.

4.3 Evaluation of Prediction Quality

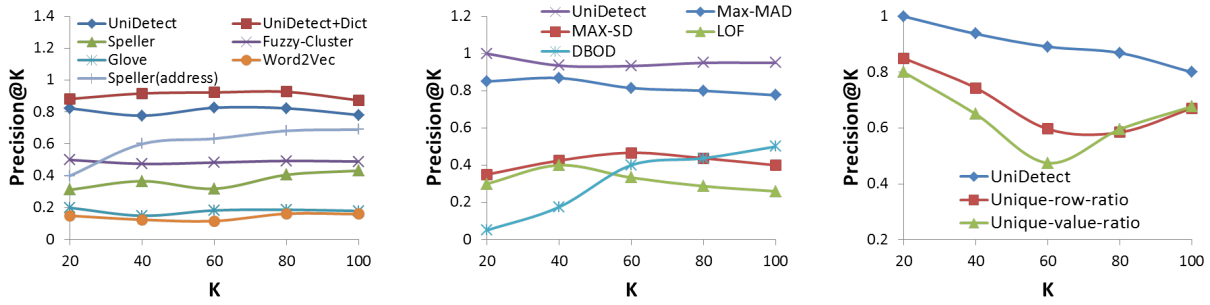
We manually judge top-100 predicted errors of each method described above, on WIKI^T, WEB^T and Enterprise^T (requiring over 5000 labels in total). Each prediction is labeled as true/false/not-sure. Our quality metric is *Precision@K* [79], defined as $\frac{\#\text{-true-errors@K}}{K}$.

Figure 8(a), Figure 9(a) and Figure 10(a) show the quality comparisons of predicted spelling errors on WEB^T, WIKI^T and Enterprise^T, respectively. We can see that UniDetect has the best precision: over 0.8 for all three corpora. Examples of detected misspellings can be seen in Figure 4(g) and 4(h). Fuzzy-Cluster performs reasonably well with precision at around 0.5. Surprisingly, the Speller from the search engine performs the worst. An inspection of the false-positives (shown in Figure 3) suggests that a mismatch between train/test – the training is based on search engine query logs, which are very different from the idiosyncratic data we encounter in tables (imagine a table with employee aliases like "JenniferA" and "SmithB" that would all trigger false-positives for Speller). When we manually filter down table data to only address data, the precision of Speller(address) improves to 0.4-0.7, which still lags behind UniDetect. Lastly, while commercial Spellers already leverage word-embedding, our results show that it is not a good fit to use Word2Vec and GloVe directly for spell checking.

We would like to highlight that in this case UniDetect detects spelling errors entirely based on a data-driven distribution analysis of WEB, without any lexical analysis, or looking at any English dictionary. This is the reason why a model trained on WEB can be applied unchanged, yet still generalize to Enterprise^T.

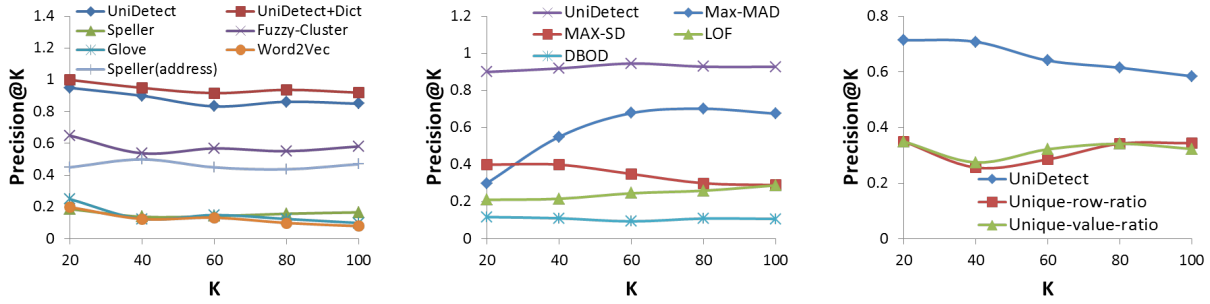
We found that false-positives of UniDetect in this case include pairs such as "Macroeconomics" and "Microeconomics" (listed in the same column). From a distribution's perspective this pair is suspiciously close, but a simple dictionary would refute the hypothesis that the pair has misspellings, as both of them are valid entries in dictionaries. We use this simple method to combine UniDetect with Wiktionary [4]. The result is labeled as "UniDetect+Dict" in Figure 8(a) and Figure 9(a), which consistently achieves precision over 0.9.

Given that WEB^T and WIKI^T are only 1% and 10% of their respective full corpus, we can extrapolate that if we were to label all tables, we can find *tens of thousands* of spelling errors



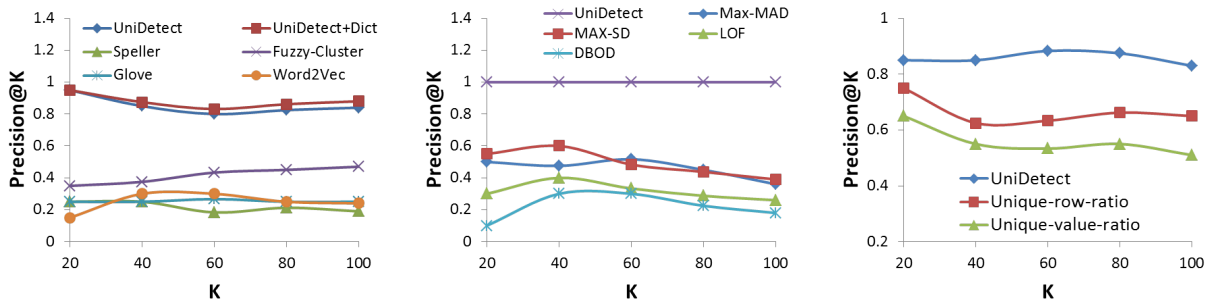
(a) Spelling (b) Numeric-outlier (c) Uniqueness

Figure 8: Quality of predicted errors on WEB^T, evaluated using Precision@K.



(a) Spelling (b) Numeric-outlier (c) Uniqueness

Figure 9: Quality of predicted errors on WIKI^T, evaluated using Precision@K.



(a) Spelling (b) Numeric-outlier (c) Uniqueness

Figure 10: Quality of predicted errors on Enterprise^T, evaluated using Precision@K.

on Wikipedia (non-trivial because Wikipedia is already of high quality), and *hundreds of thousands* more for the general Web, all at a high precision.

Figure 8(b), Figure 9(b), and Figure 10(b) show a similar comparison for numeric-outliers. UniDetect achieves precision at 0.92, 0.95 and 1, for WEB^T, WIKI^T and Enterprise^T, respectively, outperforming alternative methods specifically designed for numeric-outliers (LOF, DBOD, etc.). Among all other methods, we can see that MAX-MAD improves substantially over MAX-SD, reaffirming the benefit of leveraging robust statistics as reported in [48].

For Uniqueness-violations, a similar trend can be observed in Figure 8(c), Figure 9(c) and Figure 10(c). Note that compared to baselines, UniDetect uses the same Unique-value-ratio (UR) as its metric function, but is able to significantly outperform a simple application of the metric.

Additional experiments on FD and its variant synthesized-FD are discussed in Appendix D in the interest of space.

5 CONCLUSION AND FUTURE WORK

We propose UniDetect, a unified approach to error detection that leverages a large corpus of tables. We demonstrate that this framework can be instantiated to handle four seemingly disparate yet common classes of errors.

Interesting directions of future work include extending UniDetect to test its effectiveness for more types of errors, as well as exploring the possibility of learning configurations for more accurate detection.

A RELATED WORKS

Error detection vs. Error repair. We in this work focus on *automatic error-detection*, which is orthogonal to and one step before *error-repair* that has a long and fruitful line of research. Influential methods in error-repair include minimality-based [22, 36], HoloClean [77], as well as many other novel methods (e.g. [18, 34, 35, 82], etc.).

In the following we will review existing error-detection methods by types of errors. The authors in [17, 48] give excellent surveys of recent error-detection methods.

Numeric outliers. Hellerstein [48] proposes to use the metric of MAD (median absolute deviation) from robust statistics to detect outliers in numeric data. There is a large literature on non-parametric outlier detection in databases [19, 29, 42, 47–49, 55], which differ in aspects such as metrics used and application scenarios.

Spelling mistakes. Existing commercial systems such as Paxata [9] and OpenRefine [8] employ fuzzy-group-by to cluster together values in the same column that are syntactically close. Users are then expected to inspect the clusters to determine true misspellings. Spellers from commercial search engines [1, 6] are also relevant efforts.

Constraint violations. Existing methods detect likely violations of uniqueness [37, 48], FD constraints [21, 41, 54, 56], and other types of logic-based constraints [30, 33], mainly leverage the idea that constraints that almost hold are likely violations. While this is clearly useful, we show in UniDetect how such intuitions can be further improved in a data-driven manner.

Pattern-based error detection. Existing systems such as Trifacta [15], Power BI [10] and Talend [13] all have pre-defined regex-like patterns to recognize common data types (e.g., IP addresses, emails, etc.). When most values in an input column conform to known patterns while a small fraction of values do not, the non-conforming ones can be flagged as errors (Appendix B gives more details).

Potter's Wheel [76] infers patterns from a given column based on minimum description length (MDL).

Auto-Detect [50] uses large corpus and PMI to learn pattern incompatibility, which is consistent with and can be derived from the LR scores (Appendix C).

Other error-detection methods. There are a number violations that are not studied in this work, including more general forms of constraints (CFD [23], DC [32], etc.). As we discussed some of these are hard to automate as they tend to be highly specific to data sets that are hard to learn from **T**.

B DETECT ERROR IN EXISTING SYSTEMS

A few existing systems provide simple error-detection functionalities. We discuss a few representatives here.

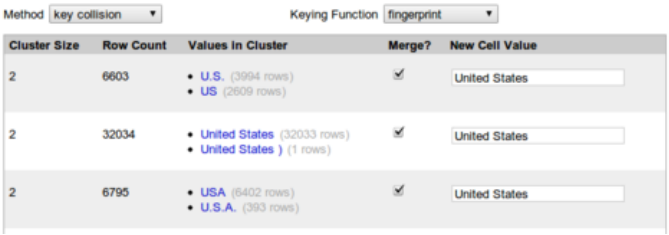


Figure 11: OpenRefine cluster similar values in the same column for possible spelling variations.

Microsoft Excel [7]. Excel pre-defines a set of 9 simple error checking rules (shown in Figure 1), which include well-known ones such as "Number stored as text", and "Formulas inconsistent with other formulas in the region". These are manually curated, high-precision rules, that covers a very limited number of scenarios, which are nevertheless already useful in a software setting (high-precision, low-recall).

Trifacta [14]. Trifacta recognizes around 10 built-in "data types" (IP-address, phone-numbers, email, etc.) based on predefined patterns [15]. Values in a column not conforming to patterns associated with a data-type are flagged as errors. In addition, Trifacta offers a rich set of visual-histograms (e.g., distribution of string lengths) for values in a column, which help users identify potential quality issues. Similar functionalities are also available in systems like Paxata [9] and Talend [13].

OpenRefine/GoogleRefine [8]. OpenRefine does not detect errors directly, but like Paxata [9] it provides a text clustering feature that groups together similar values in a column, so that users can see whether similar values may be misspelled variations of canonical values. Figure 11 shows results from column clustering, where similar values like "U.S." and "US", "USA" and "U.S.A." are grouped together, so that users can decide whether to collapse these clusters of values into a canonical representation.

C RELATIONSHIPS TO COMPATIBILITY ERRORS IN AUTO-DETECT [50]

A recent work AutoDetect [50] focuses on an orthogonal class of data errors, which is pattern incompatibility. For example, it detects values such as "2001-Jan-01" and "2001-01-01" as incompatible, using a PMI-based (point-wise mutual information) analysis of large table corpus.

We show that this basic mechanism of PMI is actually consistent with the likelihood-ratio (LR) test in the proposed UniDetect.

Specifically, the LR test we propose is $LR = \frac{P(H_0|\text{evidence})}{P(H_1|\text{evidence})}$, which can be rewritten as below (derived in Section 2.2):

$$LR = \frac{P(H_0|D,\mathbf{T})}{P(H_1|D,\mathbf{T})} = \frac{P(D|H_0,\mathbf{T})}{P(D|H_1,\mathbf{T})}\frac{P(H_0|\mathbf{T})}{P(H_1|\mathbf{T})} \propto \frac{P(D|H_0,\mathbf{T})}{P(D|H_1,\mathbf{T})} \tag{16}$$

Given two patterns P_1 and P_2 considered in AutoDetect (say, "\d\d\d-\l\l\l-\d\d" for "2001-Jan-01" and "\d\d\d-\d\d-\d\d" for "2001-01-01", where \d stands for digits and \l stands for letters, respectively), we can perform a similar LR ratio statistical test. Given a table column D that includes P_1 and P_2 in the same column, the null hypothesis H_0 is that P_1 and P_2 are not negatively correlated, or assumed to be randomly co-occurring. In this case, the probability of $P(D|H_0, \mathbf{T})$, or the chance of seeing D in $\mathbf{T}$, can be estimated as $\frac{n_1}{N}\frac{n_2}{N}$, where n_1, n_2, are the number of columns in the corpus $\mathbf{T}$ with P_1 and P_2, respectively, and N is the total number of columns in the corpus.

The alternative hypothesis H_1 is that P_1 and P_2 are negatively correlated, or incompatible. In this case, the probability of $P(D|H_1, \mathbf{T})$ can be estimated based on the observed co-occurrence of P_1 and P_2, or $\frac{n_{12}}{N}$, where n_{12} is the number of columns in the corpus $\mathbf{T}$ that have both P_1 and P_2.

Combining the above with Equation (16) above, we get

$$LR \propto \frac{P(D|H_0, \mathbf{T})}{P(D|H_1, \mathbf{T})} = \frac{\frac{n_1}{N}\frac{n_2}{N}}{\frac{n_{12}}{N}}$$

Note that the right-hand-side of the equation above is exactly the definition of PMI – the metric used in [50].

There are a substantial amount of optimizations in Auto-Detect that trades off accuracy for storage that are specific to pattern co-occurrences. However, it is interesting that the metric used in Auto-Detect, targeting an entirely orthogonal type of errors is actually consistent with our LR-tests.

D ADDITIONAL EXPERIMENTS

We also compare quality results for detecting FD errors in Figure 12. Specifically, we compare two types of FDs, one is the classical FD as discussed in Section 3.4, where FD ($X \rightarrow Y$) is said to hold on T at the instance-level, if $\frac{|\pi_X(T)|}{|\pi_{XY}(T)|} = 1$.

In addition, we also consider a variant that uses techniques from the programming synthesis literature (e.g., [45, 62, 81]), such that for a candidate $X \rightarrow Y$, not only do we require the functional relationship holds, as in a classical FD sense, but also we require an explicit programmatic-relationship to be learnt for a majority of rows from X to Y, before considering a relationship exists between the two columns.

Examples of explicit synthesized programmatic relationship are, for instance, a column with "full-name" (e.g., "Doe, John"), followed by two more columns of "first-name" ("John") and "last-name" ("Doe"). In such cases a programmatic relationship can be learnt between the columns (e.g., concatenating "last-name", a comma, a space, and "first-name" produces the "full-name" column; while splitting "full-name" using a comma and taking the first component produces the "last-name" column, etc.).

We term such refined form of FD as FD-synthesis (since program-synthesis is used). Note that explicit programmatic relationship produced between X and Y makes sure that a relationship really exists between the columns.The exact error-detection reasoning for FD-synthesis in UniDetect is identical to FD (described in Section 3.4).

Figure 12(a) and 12(b) show the comparison of detecting FD errors on WEB^T and WIKI^T, respectively. As can be seen from the results, UniDetect still outperforms FD-detection algorithms [53, 56], though the precision is not very high. This underlines the difficulty of detecting FD errors relative to other types of errors – FD contains two groups of columns, where it is substantially more likely to generate candidates that appear to have an FD, but in reality have no real relationship (e.g., because the two columns X and Y take values from a large range, making collision/FD-violations unlikely).

In comparison, Figure 12(c) and 12(d) show quality results of detecting errors that violate FD-synthesis (FD with learnt programmatic relationship via program synthesis). Compared to FD results in Figure 12(a) and 12(b), the quality of FD-synthesis is substantially better. Figure 13, and 14 show example Wikipedia tables that are detected to be in violation of FD-synthesis relationships. Note that the explicit programmatic relationships not only ensures high quality error-predictions, but also enables exact repair (through generative program synthesis).

E PROOF OF THEOREM 1

We show that the monotonicity property defined by the inequality in Equation (13) holds. Observe that given a column pair C and C', $r(C)$ and $r(C')$ are defined as follows.

$$r(C) = \frac{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq \theta_1, \text{max-MAD}(D_O^{\mathcal{P}}) \leq \theta_2\}|}{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq \theta_2\}|} \tag{17}$$

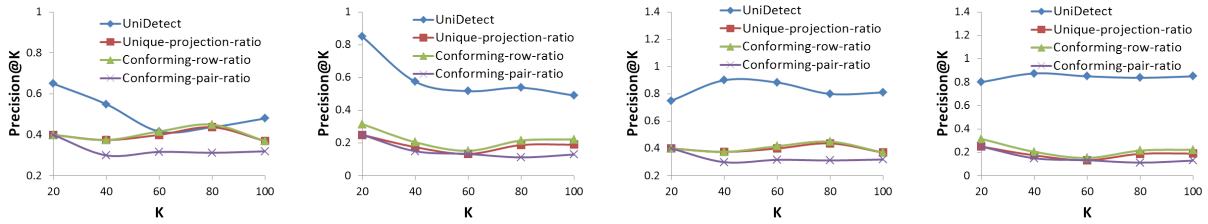
$$r(C') = \frac{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq \theta_1', \text{max-MAD}(D_O^{\mathcal{P}}) \leq \theta_2'\}|}{|\{D|D \in S_D^F(\mathbf{T}), \text{max-MAD}(D) \geq \theta_2'\}|} \tag{18}$$

Where $\theta_1 = \text{max-MAD}(C)$, $\theta_2 = \text{max-MAD}(C_O^{\mathcal{P}})$, $\theta_1' = \text{max-MAD}(C')$, and $\theta_2' = \text{max-MAD}(C_O'^{\mathcal{P}})$.

Given it is known that $\theta_1 \geq \theta_1'$ and $\theta_2 \leq \theta_2'$, it can be shown that the numerator of Equation (17) is smaller than that of Equation (18), because $\theta_1 \geq \theta_1'$ and $\theta_2 \leq \theta_2'$, which ensures that the numerator in Equation (17) is precisely a subset of that in Equation (18).

Conversely, it can be shown that the denominator of Equation (17) is larger than that of Equation (18), because $\theta_2 \leq \theta_2'$, which ensures that the denominator in Equation (17) is a superset of that in Equation (18).

Combining the relationships in numerator and denominator, we can conclude that $\Rightarrow r(C) \leq r(C')$.



(a) FD on WEB^T (b) FD on WIKI^T (c) Synthesis-FD on WEB^T (d) Synthesis-FD on WIKI^T

Figure 12: Quality of predicted errors, evaluated using Precision@K.

Highway shield	Name[1]
736	Malaysia Federal Route 736
737	Malaysia Federal Route 737
738	Malaysia Federal Route 738
739	Malaysia Federal Route 739
740	Malaysia Federal Route 740
738	Malaysia Federal Route 748

Figure 13: Real error from Wikipedia detected by FD-Synthesis. Value "738" should be "748", based on programmatic relationships with"Malaysia Federal Route 748".

Country	Contestant	National Title
Denmark	Michael Sinan[6]	Mr Gay Denmark
Finland	Janne Tiilikainen[7]	Mr Gay Finland
France	Armando Santos[8]	Mr Gay France
Hong Kong	Benjie Vasquez Caraig	Mr Gay Honkong
India	Nolan Lewis[9]	Mr Gay India
Mexico	Jaime David Montes Bernal	Mr Gay Mexico
Myanmar	Sai Pye Myo Kyaw	Mr Gay Myanmar

Figure 14: Real error from Wikipedia detected by FD-Synthesis. Value "Mr Gay Honkong" should be "Mr Gay Hong Kong", based on programmatic relationships with "Hong Kong".

REFERENCES

[1] Bing spell check. https://azure.microsoft.com/en-us/services/cognitive-services/spell-check/.
[2] Database connection strings in excel. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/connection-string-syntax.
[3] Excel error checking rules. https://excelribbon.tips.net/T006221_Changing_Error_Checking_Rules.html.
[4] Excel error checking rules. https://www.wiktionary.org/.
[5] Glove 840B tokens model. https://nlp.stanford.edu/projects/glove/.
[6] Google spell check. https://code.google.com/archive/p/google-api-spelling-java/.
[7] Microsoft excel error checking rules. https://excelribbon.tips.net/T006221_Changing_Error_Checking_Rules.html.
[8] OpenRefine (formerly Google Refine). http://openrefine.org/.
[9] Paxata data preparation. https://www.paxata.com/.
[10] Power bi. https://docs.microsoft.com/en-us/power-bi/desktop-data-types.
[11] Self-service data preparation, worldwide, 2016. https://www.gartner.com/doc/3204817/forecast-snapshot-selfservice-data-preparation.
[12] Spreadsheet mistakes - news stories, compiled by european spreadsheet risk interest group EuSpRiG. http://www.eusprig.org/horror-stories.htm.
[13] Talend data services platform studio user guide: Semantic discovery. https://help.talend.com/reader/nAXiZW0j0H~2~YApZIsRFw/_u0D0oqWxesgBDSihDgbYA.
[14] Trifacta. https://www.trifacta.com/.
[15] Trifacta built-in data types. https://docs.trifacta.com/display/PE/Supported+Data+Types.
[16] Word2Vec 100B Google News model. https://github.com/mmihaltz/word2vec-GoogleNews-vectors.
[17] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *VLDB*, 9(12), 2016.
[18] F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *Proceedings of the 12th International Conference on Database Theory*. ACM, 2009.
[19] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large databases. In *KDD*, 1996.
[20] I. Ben-Gal. Outlier detection. In *Data mining and knowledge discovery handbook*. Springer, 2005.
[21] L. Berti-Equille, H. Harmouch, F. Naumann, N. Novelli, and S. Thirumuruganathan. Discovery of genuine functional dependencies from relational data with missing values. *VLDB*, 2018.
[22] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.
[23] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*. IEEE, 2007.
[24] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*. ACM, 2000.
[25] E. Brewer. Cap twelve years later: how the. *Computer*, (2), 2012.
[26] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *VLDB*, 1(1), 2008.
[27] G. Casella and R. Berger. R. 2001, statistical inference. *Duxbury Press*.
[28] K. Chakrabarti, S. Chaudhuri, Z. Chen, K. Ganjam, Y. He, and W. Redmond. Data services leveraging bing's data assets. *IEEE Data Eng. Bull.*, 39(3), 2016.
[29] V. Chandola, A. Banerjee, and V. Kumar. Outlier detection: A survey. *ACM Computing Surveys*, 2007.
[30] F. Chiang and R. J. Miller. Discovering data quality rules. *VLDB*, 1(1), 2008.
[31] X. Chu, Y. He, K. Chakrabarti, and K. Ganjam. Tegra: Table extraction by global record alignment. In *SIGMOD*, 2015.
[32] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *VLDB*, 6(13), 2013.
[33] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *VLDB*, 6(13), 2013.
[34] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*. IEEE, 2013.
[35] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases

and crowdsourcing. In *SIGMOD*, 2015.
[36] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.
[37] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, 2002.
[38] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *VLDB*, 2(1), 2009.
[39] D. Freeman. How to make spreadsheets error-proof. *Journal of Accountancy*, 181(5), 1996.
[40] Y. Ganjisaffar, A. Zilio, S. Javanmardi, I. Cetindil, M. Sikka, S. Katumalla, N. Khatib, C. Li, and C. Lopes. qspell: Spelling correction of web search queries using ranking models and iterative correction. In *Spelling Alteration for Web Search Workshop*, 2011.
[41] L. Golab, H. Karloff, F. Korn, and D. Srivastava. Data auditor: Exploring data quality and semantics using pattern tableaux. *VLDB*, 3(1-2), 2010.
[42] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *TKDE*, 26(9), 2014.
[43] P. Hall, J. Marron, and B. U. Park. Smoothed cross-validation. *Probability theory and related fields*, 92(1), 1992.
[44] W. R. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *ACM SIGPLAN Notices*, volume 46. ACM, 2011.
[45] Y. He, X. Chu, K. Ganjam, Y. Zheng, V. Narasayya, and S. Chaudhuri. Transform-data-by-example (TDE): an extensible search engine for data transformations. *VLDB*, 11(10), 2018.
[46] Y. He, K. Ganjam, and X. Chu. SEMA-JOIN: joining semantically-related tables using big table corpora. *VLDB*, 8(12), 2015.
[47] Z. He, S. Deng, and X. Xu. An optimization model for outlier detection in categorical data. *Advances in Intelligent Computing*, 2005.
[48] J. M. Hellerstein. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*, 2008.
[49] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 2004.
[50] Z. Huang and Y. He. Auto-Detect: Data-Driven Error Detection in Tables. In *SIGMOD*, 2018.
[51] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2), 1999.
[52] B.-G. I. *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers*. Kluwer Academic Publishers, 2005.
[53] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, 2004.
[54] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, 2004.
[55] E. Keogh, S. Lonardi, and C. A. Ratanamahatana. Towards parameter-free data mining. In *SIGKDD*, 2004.
[56] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1), 1995.
[57] E. M. Knox and R. T. Ng. Algorithms for mining distance based outliers in large datasets. In *VLDB*, 1998.
[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.
[59] V. Le and S. Gulwani. Flashextract: a framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49. ACM, 2014.
[60] P. M. Lee. *Bayesian statistics*. Arnold Publication, 1997.
[61] E. L. Lehmann and J. P. Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.
[62] H. Lieberman. *Your wish is my command: Programming by example*. Morgan Kaufmann, 2001.
[63] G. Luec. A data-driven approach for correcting search quaries. In *Spelling Alteration for Web Search Workshop*, 2011.
[64] Y. E. Mark Ziemann and A. El-Osta. Gene name errors are widespread in the scientific literature. *Genome Biology*, 2016.
[65] R. A. Maronna, R. D. Martin, V. J. Yohai, and M. Salibián-Barrera. *Robust Statistics: Theory and Methods (with R)*. Wiley, 2018.
[66] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*. IEEE, 2007.
[67] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 2013.
[68] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman. *Applied linear statistical models*, volume 4. Irwin Chicago, 1996.
[69] J. Neyman and E. S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Phil. Trans. R. Soc. Lond. A.*, 1933.
[70] R. R. Panko. What we know about spreadsheet errors. *Journal of Organizational and End User Computing (JOEUC)*, 1998.
[71] T. Papenbrock, T. Bergmann, M. Finke, J. Zwiener, and F. Naumann. Data profiling with metanome. *VLDB*, 8(12), 2015.
[72] E. Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3), 1962.
[73] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.
[74] S. G. Powell, K. R. Baker, and B. Lawson. Errors in operational spreadsheets: A review of the state of the art. In *System Sciences, 2009. HICSS'09*, 2009.
[75] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.
[76] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, 2001.
[77] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *VLDB*, 10(11), 2017.
[78] S. Schelter, D. Lange, P. Schmidt, M. Celikel, and F. Biessmann. Automating largescale data quality verification. In *VLDB*, 2018.
[79] H. Schütze, C. D. Manning, and P. Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press, 2008.
[80] R. Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *VLDB*, 9(10), 2016.
[81] R. Singh and S. Gulwani. Transforming spreadsheet data types using examples. In *Acm Sigplan Notices*, 2016.
[82] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD*, 2013.
[83] C. Yan and Y. He. Auto-Type: Synthesizing type-detection logic for rich semantic data types using open-source code. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018.
[84] C. Zhao and Y. He. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *WWW*, 2019.
[85] Z. Zhao, L. De Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska. Controlling false discoveries during interactive data exploration. In *SIGMOD*, 2017.
[86] E. Zhu, Y. He, and S. Chaudhuri. Auto-join: Joining tables by leveraging transformations. *VLDB*, 10(10), 2017.