DetCat: Detecting Categorical Outliers in Relational Datasets

Arthur Zylinski
a.zylinski@students.uu.nl
Utrecht University
Utrecht, The Netherlands

Abdulhakim A. Qahtan*
a.a.a.qahtan@uu.nl
Utrecht University
Utrecht, The Netherlands

ABSTRACT

Poor data quality, commonly caused by data glitches such as outliers, significantly affects any data analytics task, leading to inaccurate decisions and poor predictions of the machine learning models. While detecting outliers in numerical data has been extensively studied, few attempts were made to solve the problem of detecting categorical outliers. In this paper, we introduce DetCat for detecting categorical outliers in relational datasets by utilizing the syntactic structure of the values. For a given attribute, DetCat identifies a set of patterns that represents the majority of the values and declares the data values that cannot be generated by those patterns as outliers. Moreover, DetCat considers the trade-off between the expressiveness and the compactness of the generated patterns and introduces a new similarity measure between the patterns in order to reduce the false predictions.

CCS CONCEPTS

• **Information systems → Data cleaning**.

KEYWORDS

Categorical values, outliers, syntactic structure, string similarity measures

ACM Reference Format:
Arthur Zylinski and Abdulhakim A. Qahtan. 2024. DetCat: Detecting Categorical Outliers in Relational Datasets. In *Proceedings of The annual International Conference on Extending Database Technology (EDBT) (EDBT '24)*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Poor data quality is a challenging problem that researchers and data scientists have to deal with during their daily work. Estimates show that around 50-80% of a data scientist's time is spent on "janitor work", such as finding datasets, data profiling, cleaning, and wrangling [3]. These activities are mandatory for preparing the data for training a machine learning model, or performing any data analytics task. Data anomalies represent one of the common data quality issues. Even though detecting anomalies in attributes with numerical values has been studied extensively, detecting anomalies in categorical attributes is less mature due to the difficulty in defining the similarity between the values. Therefore, developing a tool that can identify outliers (anomalies) in attributes with categorical attributes is of great importance.

A common solution to maintain the data quality is defining a set of business rules manually using the knowledge of domain experts. Data quality can then be assessed according to the conformance of the data with these rules. However, such a solution is extremely expensive and not scalable. Because of that, a set of tools have been developed to help data scientists in detecting erroneous values from both academia and industry.

For relational data, values in the categorical attributes usually follow specific syntactic structures. Anomalies in these cases can be identified as the values that do not conform with the syntactic structure of the majority of the values. For example, a *data* attribute could take the form of *dd-mm-yyyy*, *dd-mm-yy* or *dd/mm/yyyy*, ...etc. In such case, values that have the year only *yyyy* can be reported as outlier. This characteristic about relational data was utilized by FAHES [7] for detecting the disguised missing values and by SURAGH [1] for detecting ill formed records. However, FAHES focuses on the disguised missing values (DMVs) only where it assumed that a value should be used repeatedly to be considered as a DMV. Suragh, on the other hand, reports the whole record as erroneous without identifying the actual cell that caused the error.

Another tool called dBoost [5] identifies a set of rules based on the data patterns and reports the values that do not conform with the rules as outliers. A major problem with dBoost is the large range of parameters that requires skilled users to use it. RAHA [4] combines dBoost with a set of other data cleaning tools to help the users in using those tools without the need for setting a large number of parameters. RAHA asks the user to provide a set of positive and negative examples of the erroneous cells. It then decides on which tool to be used for cleaning the data and what parameters should be passed to the the selected tool based on the data profile and the provided examples. However, labeling enough set of examples could be challenging in cases when the domain knowledge about the data is limited.

In this paper, we propose DetCat[1] as a tool that utilizes the syntactic structure of the values in the categorical attributes for detecting syntactic anomalies. DetCat detects the anomalies by identifying a set of dominant patterns (Reg-Ex like patterns) that generate the majority of the values in the attribute and reporting the values that cannot be generated by the dominant patterns as anomalies (outliers). Unlike FAHES and SURAGH, DetCat focuses on the expressiveness of the identified patterns such that the kept portion of the original values in the patterns is maximal. For example, if a "year" attribute contains values that are from the 20-th and 21-st centuries, then it would be more informative to have the patterns {'19ĐĐ, 20ĐĐ}, where 'Đ' represents a digit, than having a single pattern 'ĐĐĐĐ'.

[1]DetCat is short for Detective Cat.

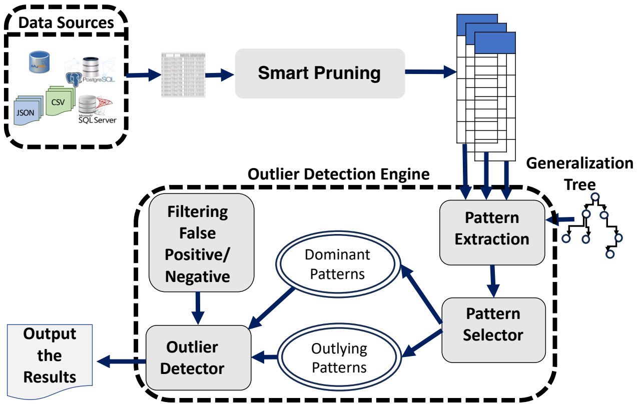


Figure 1: The DetCat Architecture.

Moreover, the tool does not consider the frequency of the values as a major factor in reporting a value as an outlier or normal like FAHES. Furthermore, to reduce the false predictions of the reported outliers, we introduce a modified version of the Levenshtein distance to compare the reported outlying patterns with the dominant patterns. An outlying pattern can be considered as a false positive if the average distance to the other dominant patters is small such that it is comparable to the pairwise distance between the dominant patterns. The proposed distance measure works well for comparing the patterns and reducing the number of values that are falsely reported. The demo will show the effectiveness of DetCat in detecting the outliers in the categorical attributes.

2 THE SYSTEM ARCHITECTURE

The DetCat architecture is shown in Figure 1. The tool consists of multiple components including a data profiler, and outlier detection engine. The detection engine contains a pattern generation module that generates syntactic patterns of the data values using the Generalization Tree (GT) in Figure 2, a pattern selector module that discriminates the dominant patterns from the outlying patterns, and a false positive/negative filtering module.

2.1 Data Profiling and Numerical Attributes Pruning

Using this module, a summary statistics about the data are collected and the type of each attribute is identified. We identify four types of the data *{integer, float, text, code}*. Inspired by [6], we remove the numerical attributes as their syntactic structure is not helpful in detecting the outliers. The removed attributes are the pure numerical attributes that represents quantitative values. By *quantitative*, we refer to the values that are commonly used in statistics, which typically has meaning as a measurement (e.g., a person's height) or a count (e.g., a person's stock shares). In contrast to quantitative values, *qualitative* values are non-statistical as they cannot be aggregated, for example, it is meaningless to compute the average of two zip codes. The qualitative attributes are identified as 'code' by DetCat and are considered during the detection process.

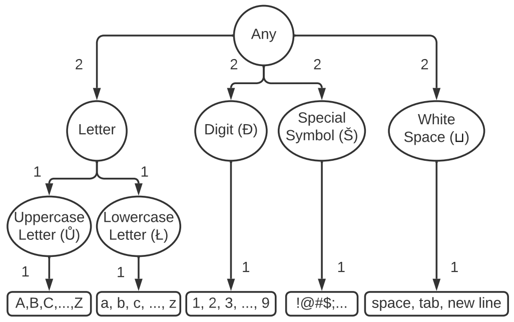


Figure 2: The generalization tree (GT) for generating the Reg-Ex like patterns.

Table 1: Examples of patterns that can be generated from simple values.

Literal String	Possible Patterns
leaf	leaf, leaŁ, leŁf, leŁŁ, lŁaf, lŁaŁ, lŁŁf, lŁŁŁ, Łeaf, ŁeaŁ, ŁeŁf, ŁeŁŁ, ŁŁaf, ŁŁaŁ, ŁŁŁf, ŁŁŁŁ
US$	US$, USŠ, UŮ$, UŮŠ, ŮS$, ŮSŠ, ŮŮ$, ŮŮŠ

2.2 Pattern Generation

For finding the dominating patterns in the attributes, Reg-Ex like expressions are generated by replacing zero or more characters in each value with their character classes (the parent nodes of the characters in the GT (Figure 2)). The characters in the values are represented in the leaves of the GT. In the GT, the input characters from the keyboard are categorized into five main classes that are represented with commonly unused symbols {Ů, Ł, Đ, Š, ⊔} instead of the conventional symbols \u, \l, \d for readability and simplicity. In addition, we define a virtual class that represents the English alphabetic 'Letter'. Even though the 'Letter' is not used in the generated patterns, it is used to increase the distance between the alphabet symbols and the other symbols. Here, we use the alphabet to refer to the set of lower/upper-case English letters, even though, in general, the alphabet refers to the set of characters that are used in the values.

While the patterns can contain symbols from the internal nodes in the tree, the actual characters are used to increase the expressiveness of the generated patterns. In this work, we aim to keep as many characters as possible from the original values. Examples of the patterns that can be generated from the different values are presented in Table 1. Since the number of possible patterns that can be generated from a value with length n is 2^n, we use multi-level index for fast access to the generated patterns. First, we index the patterns based on their lengths. After that, we index them based on their values. Moreover, for each pattern, we store the number of values that can be generated by that pattern to identify the dominant patterns that generate the majority of the values in the attribute. Furthermore, the number of generated patterns is controlled by merging patterns into more generic patterns that contains more class labels instead of the actual characters (less expressive patterns).

For reducing the running time, attributes with long strings and high diversity in the classes of their characters, only two patterns are generated. The first pattern contains the characters from the leaves of the GT. In the second pattern, each character from a given value is replaced by the class label of the parent of the leaf that contains the character. This heuristic reduces the expressiveness of the generated patterns but improves the efficiency of the tool significantly. Moreover, for large tables, we sample a representative set of values and generate an initial set of patterns from those values. We add more patterns when necessary after comparing the rest of the values against the initial set of patterns.

2.3 Dominant Pattern Identification

After generating the set of patterns that can generate all the values in the attribute, we identify the set of patterns that can generate the majority of the values. For selecting the set of dominant patterns, we define a scoring function based on the number of values that can be generated by that pattern and the number of characters in the pattern that belong to the leaf nodes in the GT. This scoring function considers the trade-off between the expressiveness and compactness of the patterns. The score of a given pattern p is define as: $s(p) = \frac{|v_p|}{|c_p|+1}$, where v_p is the set of values that can be generated by the pattern p and c_p is the number of character class labels in the pattern p. For example, if we have an attribute "year" with 96 values that starts with '19' and 108 that starts with '1', the score of $p_1 = 19ÐÐ$ and $p_2 = 1ÐÐÐ$ are 32 and 27, respectively. In this case, the pattern p_1 will be chosen over p_2. It is worth noting that the generated patterns are forced to be disjoint. That is, for the final set of patterns $\mathfrak{P} = \{p_1, p_2, \ldots, p_k\}$, the set of values that can be generated by a pattern p_i cannot overlap with the set of values that are generated by pattern p_j, where $1 \leq i, j \leq k$ and $i \neq j$.

Once the final set of patterns that represents all values in an attribute is obtained, we declare the set of patterns with high score values as dominating patterns. Any value that is generated by one of the dominating patterns is considered as a normal value. The rest of the values are declared as outliers. However, determining the score value that can be used to declare a pattern as dominant or not is a challenging task. According to the principal component analysis (PCA), eigenvectors that preserves 70% – 90% of the data variation are considered good enough for providing a good approximation of the data [2]. If we map the generated patterns to the eigenvectors and the pattern coverage to the eigenvalues, then it would be recommended to declare the patterns that cover more than 70% of the values as outliers. However, according to the normality assumption of the data in statistics and using the 3σ rule, 99.7% of the data are considered to be normal. Moreover, previous studies [8] estimated the ratio of erroneous values to be around '5%' in many datasets. Based on this, we consider the patterns that can generate more than 95% of the values in the attribute as dominating patterns (we use 99% in 3). The rest of the patterns are flagged as potential outliers and passed with the dominating patterns to a module that filters the false positives/negatives.

2.4 Eliminating False Predictions

After determining the set of dominating patterns, we use another module to minimize the number of values that are reported incorrectly (false positives/negatives). To do so, we compare the patterns that are close to the cutoff threshold with the patterns that are

Table 2: Examples of the distance between different values and patterns.

String 1, String 2	Cost	Notes
Netherlands, 5etherlands	7	Substitution of 'N' with '5'.
Neherlands, Netherlands	3	Insertion of 't'.
ÐÐÐÐ, 12345	6 or 5	1 for each of the 4 substitution, and 2 for the insertion.
Alex, Adam	6	Substitution of 3 characters within the same class, each with a cost of 2.
Alex, 3lex	7	Substitution of 'A' with '3'.
Alex, A3lex	8	Insertion of '3' after an 'A'.

clear dominant/outlying patterns. Patterns with small average distance to the dominant patterns are considered as normal patterns and vice versa. However, string similarity functions such as Levenshtein distance or Jaro similarity are not suitable to compare the patterns. For example, when using Levenshtein distance (Minimum Edit Distance (med)) to compare "Alex" with "3lex" and "Adam", the $med(Alex, 3lex) = 2$ whereas $med(Alex, Adam) = 6$. For an attribute with the names of people, one can argue that "3lex" should be reported as an outlier and thus should have larger distance to the rest of the values. For this reason, we define a customised version of Levenshtein distance that utilizes the weights on the edge of the GT in Figure 2.

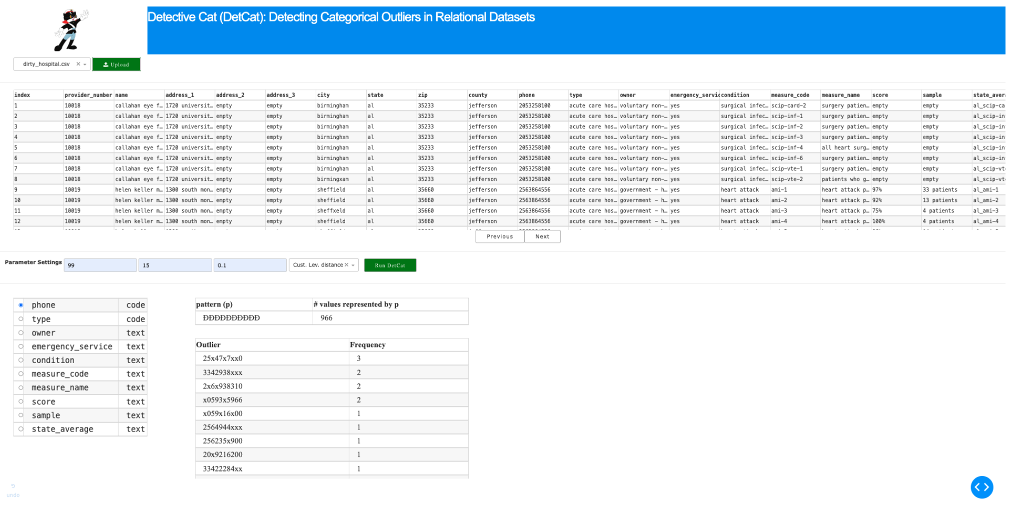
2.5 Customized Min-Edit Distance

The proposed distance measure is based on character classes in the GT. Each edge in the tree have a weight associated with it. These weights determine the cost of transforming a string x into another string y, string x into pattern y, or pattern x into pattern y. The substitution cost of a given character (or class label) in node n_i of the GT with another character (class label) n_j is the summation of the weights associated with the edges on the path from n_i to n_j in the GT as shown in Figure 2. For the insertions and deletions, we measure the cost based as the cost of substituting the character by the character before the inserted/deleted character in the updated string plus '1'. We use larger cost than the substitution cost to preserve length of the patterns, which is a key feature of the patterns in attributes with type 'code' (such as zip code).

According to this technique, substituting a leaf node character with its parent class label has a cost of '1' (e.g. substituting '3' with 'Ð' costs 1). Moreover, substituting a character in a given leaf node with another character in the same leaf node will cost 2 (e.g. 'a' with 'b' or '3' with '9'). However, substituting characters on the leaves of different subtrees around the root will have higher cost (e.g. substituting 'a' with '3' costs 7 while inserting/deleting '3' after 'a' costs 8). Table 2 shows a set of examples for computing the Min-Edit Distance between a set of patterns (or values).

3 DEMONSTRATION PLAN

During the demo, we will use publicly available datasets that are used as benchmarks for evaluating the data cleaning tools. We will use all the dataset that were used by RHAR [4]. However, the audience are also welcome to bring their own datasets and test our tool using those datasets.


Detective Cat (DetCat): Detecting Categorical Outliers in Relational Datasets
dirty_hospital.csv
Upload
Previous
Next
Run DetCat
undo

index	provider_number	name	address_1	address_2	address_3	city	state	zip	county
1	10018	callahan eye f…	1720 universit…	empty	empty	birmingham	al	35233	jefferson
2	10018	callahan eye f…	1720 universit…	empty	empty	birmingham	al	35233	jefferson
3	10018	callahan eye f…	1720 universit…	empty	empty	birmingham	al	35233	jefferson
4	10018	callahan eye f…	1720 universit…	empty	empty	birminghxm	al	35233	jefferson
5	10018	callahan eye f…	1720 universit…	empty	empty	birmingham	al	35233	jefferson
6	10018	callahan eye f…	1720 universit…	empty	empty	birmingham	al	35233	jefferson
7	10018	callahan eye f…	1720 universit…	empty	empty	birmingham	al	35233	jefferson
8	10018	callahan eye f…	1720 universit…	empty	empty	birmingxam	al	35233	jefferson
9	10019	helen keller m…	1300 south mon…	empty	empty	sheffield	al	35660	jefferson
10	10019	helen keller m…	1300 south mon…	empty	empty	sheffield	al	35660	jefferson
11	10019	helen keller m…	1300 south mon…	empty	empty	sheffxeld	al	35660	jefferson
12	10019	helen keller m…	1300 south mon…	empty	empty	sheffield	al	35660	jefferson

index	phone	type	owner	emergency_servic	condition	measure_code	measure_name	score	sample	state_aver
1	2053258100	acute care hos…	voluntary non-…	yes	surgical infec…	scip-card-2	surgery patien…	empty	empty	al_scip-ca
2	2053258100	acute care hos…	voluntary non-…	yes	surgical infec…	scip-inf-1	surgery patien…	empty	empty	al_scip-in
3	2053258100	acute care hos…	voluntary non-…	yes	surgical infec…	scip-inf-2	surgery patien…	empty	empty	al_scip-in
4	2053258100	acute care hos…	voluntary non-…	yes	surgical infec…	scip-inf-3	surgery patien…	empty	empty	al_scip-in
5	2053258100	acute care hos…	voluntary non-…	yes	surgical infec…	scip-inf-4	all heart surg…	empty	empty	al_scip-in
6	2053258100	acute care hos…	voluntary non-…	yes	surgical infec…	scip-inf-6	surgery patien…	empty	empty	al_scip-in
7	2053258100	acute care hos…	voluntary non-…	yes	surgical infec…	scip-vte-1	surgery patien…	empty	empty	al_scip-vt
8	2053258100	acute care hos…	voluntary non-…	yes	surgical infec…	scip-vte-2	patients who g…	empty	empty	al_scip-vt
9	2563864556	acute care hos…	government - h…	yes	heart attack	ami-1	heart attack p…	97%	33 patients	al_ami-1
10	2563864556	acute care hos…	government - h…	yes	heart attack	ami-2	heart attack p…	92%	13 patients	al_ami-2
11	2563864556	acute care hos…	government - h…	yes	heart attack	ami-3	heart attack p…	75%	4 patients	al_ami-3
12	2563864556	acute care hos…	government - h…	yes	heart attack	ami-4	heart attack p…	100%	4 patients	al_ami-4

Parameter Settings 99 15 0.1 Cust. Lev. distance

●	phone	code
○	type	code
○	owner	text
○	emergency_service	text
○	condition	text
○	measure_code	text
○	measure_name	text
○	score	text
○	sample	text
○	state_average	text

pattern (p)	# values represented by p
ĐĐĐĐĐĐĐĐĐĐ	966

Outlier	Frequency
25x47x7xx0	3
3342938xxx	2
2x6x938310	2
x0593x5966	2
x059x16x00	1
2564944xxx	1
256235x900	1
20x9216200	1
33422284xx	1

Figure 3: The generalization tree for generating the Reg-Ex like patterns.

Table 3: Examples of the discovered erroneous patterns in the hospital dataset.

attribute	Dominant Patterns	Erroneous Patterns
provider_number	100ĐĐ	x00ĐĐ, 100xĐ, 100Đx
city	birmingham, gadsden, dothan	faxette, annxston, hamilxon
zip	3ĐĐĐĐ, 99508	3x640, x590x, 99559
sample	Đ patients, ĐĐ patients, ĐĐĐ patients, empty	x patients, 15 patiexts, 126 patxents

3.1 Patterns Discovery and Outlier Detection

Figure 3 shows the interface of the DetCat where a snippet of the hospital[2] dataset is shown for the user to give an understanding about the dataset. Attributes that are identified by the profiler as 'text' or 'code' are shown in the table on the bottom-left corner with the type of each attribute. The selection buttons next to the names of the attributes allow the user to display the dominant patterns and the outliers in the selected attribute. The figure shows that for attribute 'score', there are a few dominant patterns {'100%', 'empty', 'ĐĐ%', 0%'}. The identified outliers are also displayed with their frequencies. The pattern '4%' clearly represents a normal value but it has been identified as an outlier. However, this pattern has been flagged as a false positive pattern by the false positive/negative filtering module. Other patterns and outliers that were discovered by DetCat from the dirty version of the hospital dataset are shown in Table 3. It is interesting that DetCat reported the zip code {99559} as outlier while this code is a correct value. However, this code belong to a different state (Alaska) than the state (Alabama) of the other values in the dataset.

3.2 Effects of the Configuration Parameters

The conference attendees can also check the effects of changing the configuration parameters on the performance of our tool. Detcat has four main parameters: i) minimum coverage of the dominant patterns (ξ): we add more patterns to the set of dominant patterns until the ratio of the generated values using the dominant patterns to the total number of values is greater than ξ; the rest of the patters are flagged as potential erroneous patterns. ii) The max allowed length for values (L): no exhaustive search for patterns in values with length greater than L; instead, we use only two patterns (the pattern of the original characters and the fully generalized pattern). iii) The minimum coverage per dominant pattern (γ): for a pattern p_i to be flagged as dominant pattern, the ratio of the generated values by p_i to the total number of values in the attribute must be greater than γ; Parameters ξ and γ are considered based on the fact that outliers are rare events and should be minority in any given dataset. iv) The similarity measure: we use the customized Levenshtein distance to measure the distance between patterns. However, attendees can use the original Levenshtein distance to filter the false positives/negatives. This can show the power of our new distance measure in filtering the false predictions. The demo will provide a clear understanding of outlier detection in attributes with categorical values.

[2]https://github.com/BigDaMa/raha/tree/master/datasets/hospital

REFERENCES

[1] Mazhar Hameed, Gerardo Vitagliano, Lan Jiang, and Felix Naumann. 2022. SURAGH: Syntactic Pattern Matching to Identify Ill-Formed Records.. In *(EDBT'22)*. 143–154.
[2] I. T. Jolliffe. 2002. *Principal Component Analysis.* Springer-Verlag, New York.
[3] Steve Lohr. 2014. For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights. *New York Times* (Aug 2014).
[4] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *SIGMOD '19*. 865–882.
[5] Clement Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam Madden. 2016. Outlier detection in heterogeneous datasets using automatic tuple expansion. (2016).
[6] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern Functional Dependencies for Data Cleaning. *Proc. VLDB Endow.* 13, 5 (2020), 684–697.
[7] Abdulhakim A. Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. 2018. FAHES: A Robust Disguised Missing Values Detector. In *SIGKDD'18*. 2100–2109.
[8] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201.