Detective Cat: Detecting Categorical Outliers in Relational Datasets

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

Abdulhakim 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 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 Detective Cat (DetCat) for detecting categorical outliers in relational datasets by utilizing the syntactic structure of the values. DetCat identifies a set of patterns in a given column 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 compactness of the generated patterns and introduces a new similarity measure that helps in reducing the number of false positives/negatives.

CCS CONCEPTS

• **Information systems → Data cleaning**.

KEYWORDS

Categorical values, outliers, syntactic structure, string similarity measures

ACM Reference Format:
Arthur Zylinski and Abdulhakim Qahtan. 2024. Detective Cat: 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, 5 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, performing data profiling, cleaning, and wrangling activities before the data can be utilized for a machine learning, data mining or analysis task [3]. 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 hiring domain experts who can define a set of business rules manually. Data quality can then be estimated according to the conformance of the data with such rules. However, this solution is extremely expensive and not scalable. For this reason, a set of tools have been developed to help the data scientists in detecting erroneous values. 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.

In previous studies, the syntactic structure of the values in relational data was utilized for detecting the disguised missing values in FAHES [7] and detecting ill formed records in SURAGH [1]. 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, report the whole record as erroneous without identifying the actual cell that caused the error. Another system, RAHA [4], combines multiple data cleaning tools including dBoost [5], which identifies a set of rules based on the data patterns and report 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 solved such problem by asking the user to provide a set of positive and negative examples of the erroneous cells. It then decide 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, providing the labeled examples could be hard in cases when the domain knowledge about the data is limited.

In this paper, we propose Detective Cat as a tool that utilizes the syntactic structure of the values in the categorical attributes for detecting syntactic anomalies. Detective Cat 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, Detective Cat focuses on the expressiveness of the identified patterns such that the kept portion of the original values in the patterns is maximal. Moreover, the tool does not consider the frequency of the values as an important factor in reporting a value as an outlier or normal like FAHES. To reduce the false positive/negative of the reported

EDBT '24, March 25 - 28, 2024, Paestum, Italy

ACM ISBN XXX-X-XXXX-XXXX-X/XX/XX…$15.00
https://doi.org/XXXXXXX.XXXXXXX

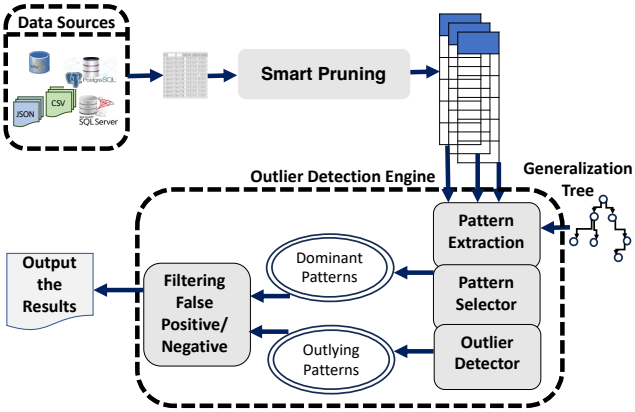

Figure 1: The Detective Cat Architecture.

outliers, we introduce a modified version of the Levenshtein distance to compute the distance between the different patterns and compare the reported outlying patterns with the dominant patters. An outlying pattern can be considered as a false positive if the average distance to the other dominant patters is small. 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 Detective Cat in detecting the outliers in the categorical attributes.

2 THE SYSTEM ARCHITECTURE

The Detective Cat 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 utilizes the Generalization Tree (GT) in Figure 2, a pattern selector module that distinguish between dominant and outlying patters, and a false positive/negative filtering module.

2.1 Data Profiling and Numerical Attribute Pruning

In this task, 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 (for example, a person's height) or a count (for example, a person's stock shares). In contrast to quantitative values, *qualitative* values that are non-statistical as they cannot be aggregated, for example, one cannot compute the average value of two zip codes. The qualitative attributes are identified as CODE and are considered during the detection process.

2.2 Pattern Generation

For finding the dominating patterns in the values, Reg-Ex like expression are generated by replacing zero or more characters in each value with their character classes in the GT that is shown in Figure 2 (the parent nodes). The characters in the actual values

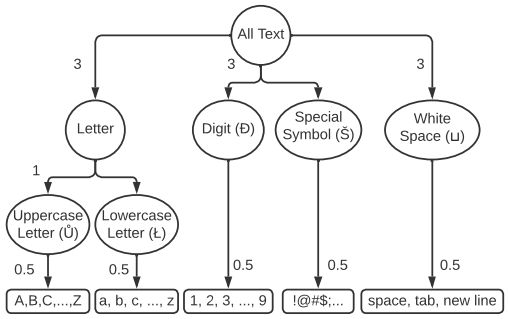

Figure 2: The generalization tree (GT) for generating the Reg-Ex like patterns.

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

are represented in the leaves of the GT while the patterns can contain symbols for internal nodes in the tree. In this work, we aim to keep as many characters as possible from the original values (increasing the expressiveness of the patters). We represent the classes with commonly unused symbols {Ů, Ł, Đ, Š, ␣} instead of the conventional symbols \u, \l, \d for simplicity.

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 to allow for a fast access to the extracted 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 generates the majority of the values in the attribute.

For long strings with high diversity in the classes of their characters, we use the classes directly instead of the actual characters. In this case, for each value, we generate only two patterns where the first pattern contains the characters from the leaves of the GT. In the second pattern, each character from the given value that belongs to a specific leaf in the tree is replaced by the class label of the parent of that leaf. This can reduce the expressiveness of the pattern but improves the efficiency of the tool significantly.

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 of the

patterns (containing as many characters from the original values) and the compactness of the patterns (containing more class labels from the internal nodes in the tree). 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, a "year" attribute contains 96 values that starts with '19' and 108 that starts with '1'. In this case, the score of p_1 = 19ĐĐ and p_2 = 1ĐĐĐ are 32 and 27, respectively. In this case, the pattern p_1 will be preferable over p_2.

In order to maintain a minimal set of patterns per attribute, 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 a column is obtained, we declare the set of patterns with high score values as dominating patterns. The values that are 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. To solve this problem, we map the outlier detection problem to the principal component analysis (PCA). The selection of the principal components/eigenvectors of a given dataset can be analogized to the selection of the dominant patterns. In the PCA, the principal components that accumulate 70% − 95% of total variation in the data are considered important [2]. Given that outliers represent rare events, we use a more conservative threshold. After sorting the patterns based on their score,we consider the patterns that can generate more than 95% of the values in the attribute as dominating patterns. The rest of the patterns are flagged as potential outliers and passed into a module that filters the false positives/negatives.

2.4 Eliminating False Positives/Negatives

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 clear dominant/outlying patterns. Patterns with small average distance to the dominant patterns are considered as normal patterns and vice versa. However, strings 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", then $med(Alex, 3lex) = 2$ whereas $med(Alex, Adam) = 6$. For an attribute with the names of people, one can argue that "3lex" should be reported as outlier and thus should have larger distance value to the rest of the values. For this reason, we define a customised version of Levenshtein distance that utilizes the weights in GT in Figure 2.

2.5 Customized Min-Edit Distance

The proposed similarity measure describes a hierarchical GT based on character classes. Traversing each level of the tree will have costs associated with it, and will determine the cost of transforming string x into string y, string x into pattern y, or pattern x into pattern y.

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

String 1, String 2	Cost	Notes
Netherlands, 5etherlands	8	Substitution of 'N' with '5'.
Neherlands, Netherlands	1.5	Insertion of 't'.
ĐĐĐĐ, 12345	3.5	0.5 for each of the 4 substitution 0.5, and 1.5 for the insertion.
Alex, Adam	1.5	Substitution of 3 characters within the same class, each with a cost of 0.5.
Alex, 3lex	8	Substitution of 'A' with '3'.
Alex, A3lex	8	Insertion of '3' after an 'A'.

Insertion, deletion, and substitution costs have been determined based on the cost associated with each edge in the GT (Figure 2).

The substitution cost of a given character (or class label) in node n_i in the GT with another character (class label) n_j is the summation of the costs associated with the edges on the path from n_i to n_j in the GT. Based on the GT, substituting a leaf node character with its parent class label has a cost of 0.5 (e.g. substituting '3' with 'Đ'). Moreover, substituting a character in a given leave node with another character in the same leaf node will cost 1 (e.g. 'a' with 'b' or '3' with '9').

We measure the cost of insertions and deletions according to the previous letter in the string that is being updated. If the inserted/deleted character is within the same subtree around the root 'All Text', we consider the cost as 1.5, while it is 8 if the inserted/deleted character is in a different subtree than the previous character in the string. We use larger cost than the substitution cost to preserve length, which is a key feature of the patterns in attributes with type 'CODE' (such as zip code). 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. However, the audience are also welcome to bring their own datasets and test our tool using those datasets. We will focus on evaluating the accuracy of the tool in discovering syntactic outliers and on explaining why the reported values are considered as outliers.

3.1 Patterns Discovery and Outlier Detection

Figure 3 shows the interface of the Detective Cat where a snippet of the data is shown for the user to give an understanding on the types of the values in the dataset. The output of the profiler are shown in the table on the bottom-left corner with the type of each attribute that was determined by the profiler. 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 'year', there are two dominant patterns '19ĐĐ' and '200Đ'. The identified outliers are also displayed with their frequencies. Other identified patterns and outliers are shown in

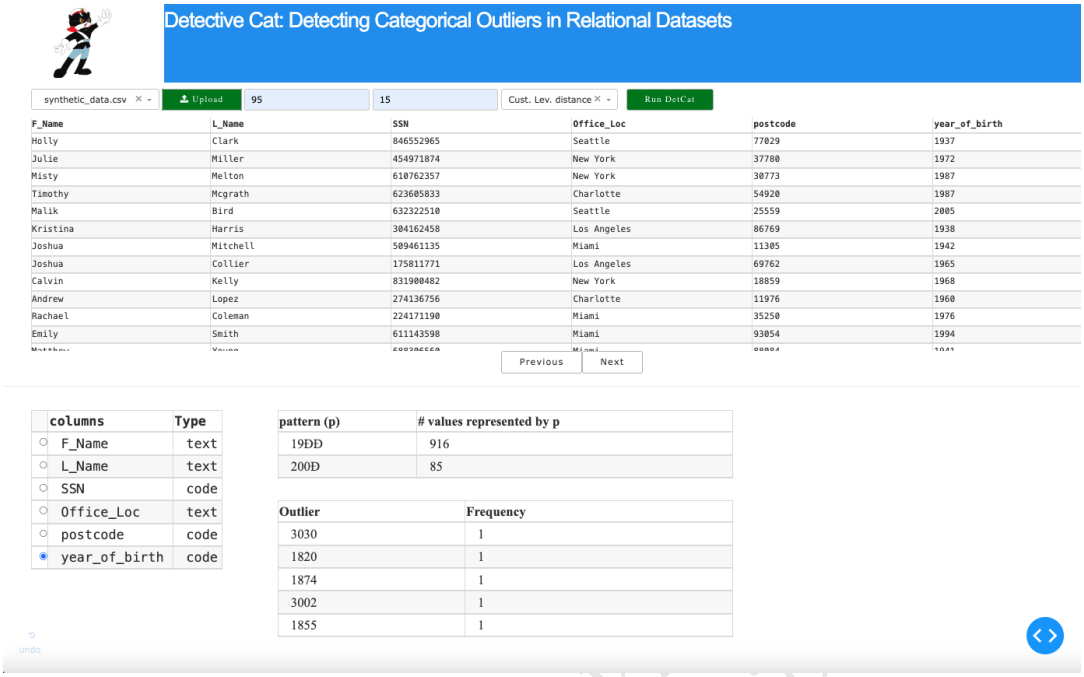

Figure 3: The generalization tree for generating the Reg-Ex like patterns.

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

attribute	Dominant Patterns	Erroneous Patterns
provider_number	100ĐĐ	1xxĐĐ, 100xĐ, 100Đx
city	birmingham, gadsden, dothan	faxette, annxston, hamilxon
zip	3ĐĐĐĐ	3x640, x590x, 99508, 99559
sample	Đ patients, ĐĐ patients, ĐĐĐ patients, empty	x patients, 15 patiexts, 126 patxents

Table 3 show a set of erroneous patterns that were discovered in the hospital dataset (the dirty version)[1].

It is interesting that Detective Cat reported the zip codes {99508, 99559} as outliers. While these codes are correct values, they belong to a different state (Alaska) than the state (Alabama) of the other values in the dataset.

3.2 Robustness to the Configuration Parameters

The conference attendees can also check the effects of changing the configuration parameter on the performance of our tool. Detective cat has three main parameters: i) minimum coverage of the dominant patterns (ξ): we keep adding 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.

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

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 similarity measure: we use the customized Levenshtein distance to measure the distance between patterns. However, attendees can use the original Levenshtein distance and Jaro similarity 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 the outlier detection in attributes with categorical values.

ACKNOWLEDGMENTS

This project was partially supported by the Focus Area Applied Data Science at Utrecht University.

REFERENCES

[1] Mazhar Hameed, Gerardo Vitagliano, Lan Jiang, and Felix Naumann. 2022. SURAGH: Syntactic Pattern Matching to Identify Ill-Formed Records.. In *Proceedings of the 25th International Conference on Extending Database Technology (EDBT)*. 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 *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 865–882.
[5] Clément Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam Madden. [n. d.]. Outlier Detection in Heterogeneous Datasets using Automatic Tuple Expansion. ([n. d.]).
[6] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern Functional Dependencies for Data Cleaning. *Proc. VLDB Endow.* 13, 5 (jan 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 2100–2109.