FIONA - A Categorical Outlier Detector Framework

Computer Science Masters Thesis

Tsiamis, A. (Thanos)

First supervisor
Dr. A.A.A. (Hakim) Qahtan

Second supervisor
Prof. Dr. I. (Yannis) Velegrakis

Utrecht University

Department of Information and Computing Sciences
Utrecht University
The Netherlands
January 2023

Abstract

FIONA (FInding Outliers iN Attributes) is a novel framework designed for detecting outliers in categorical data. Outliers, often indicative of errors or anomalous observations, can have a significant impact on data analysis and decision-making processes. In the case of categorical attributes, the task of detecting outliers necessitates the definition of a similarity metric between different values, a task more intricate than with numerical attributes. FIONA aims to address this challenge by focusing on the syntactic structures of attribute values, providing a powerful tool for identifying unusual patterns within datasets.

The framework operates in an unsupervised manner, eliminating the need for training examples. It leverages syntactic transformations, such as regular expressions and generalizations, to capture and analyze the structural characteristics of categorical values. By constructing a tree-like structure and applying a custom scoring function, FIONA systematically compares and evaluates the similarity of attribute values.

The evaluation of FIONA on various datasets, demonstrates its effectiveness in outlier detection. While some false positives are identified, further analysis reveals interesting insights and highlights the importance of considering semantic context alongside the syntactic structures. FIONA's scalability allows it to handle large datasets efficiently in contrast to conventional baseline methods, making it a valuable tool for outlier detection in various real-world applications.

Keywords— Syntactic Structure, Data Cleaning, Categorical Outliers, Framework

Acknowledgements

I would like to acknowledge my two supervisors, Hakim Qahtan and Yannis Velegrakis, for their guidance throughout my thesis.

I am also deeply indebted to my family for their endless encouragement and understanding. Mom, Dad and Lydia thank you.

I would also like to extend my heartfelt appreciation to Marina for her unwavering support throughout this journey. Your belief in my capabilities, endless encouragement, and willingness to lend a listening ear have been invaluable sources of motivation and strength.

Last but not least, I would like to express my sincere gratitude to Eva for helping me believe in myself.

Contents

Chapter 1

Introduction

The amount of data generated by humanity is notable. In 2021, 11.9 billion devices were connected to the Internet, accessing databases and generating data while forecasts show this number may reach almost 30 billion by 2030 [86]. In particular, on the 29th of June 2017 the European Organisation for Nuclear Research (CERN) announced that it exceeded the 200 Petabyte threshold of data captured in its servers [31], while 4 months later they announced that it collected a staggering 12.3 Petabytes of data for the month of October [48]. On a similar note, the amount of individuals accessing the Internet and subsequently databases has doubled since 2010, reaching 60% of the global population [24].

Given the enormous amount of transactions that occur within databases, it's not surprising that they may also contain incorrect values. However, identifying these values is a complex task due to the vast quantity of legitimate entries. While there are many different database models available today [78, 67, 56], relational databases are still the most prevalent, accounting for 71.9% of all databases worldwide [28]. These databases, which were first described by Codd in 1970 [23], are based on collections of tables with rows and columns, also known as relations.

Even within relational databases, which are highly structured, it's still highly unlikely that there won't be any incomplete, incorrect, corrupted records or individual values. For example, the Diabetes Dataset of the UCI ML Repository contains a value of zero for the attribute of blood pressure [26], while the Road Safety Data 2015-2016 from *data.gov.uk* repository contains both dates and unique IDs for an attribute named reference [6].

Outlier detection (also called anomaly detection) pertains to finding unexpected patterns in the data. Since this is a fairly broad concept, this thesis will focus on detecting outliers in categorical attributes that are syntactically different than the normal values, id est values in which their syntactic structure substantially deviates from the rest of the entries. In contrast to the numerical attributes where a lot of work has been done [18], the work on detecting outliers in categorical attributes is still limited.

1.1 Motivation

In today's data-driven world, categorical data is ubiquitous, and its importance cannot be overstated. However, it is highly improbable that all the data collected is clean and free from errors or anomalies. Outliers in categorical data can manifest in various ways, including

unexpected values or patterns, and detecting them is critical for maintaining the integrity of the data and ensuring the validity of any conclusions drawn from it.

The field of outlier detection has seen extensive research over the years, but most of the existing methods have been developed for continuous data. Therefore, there is a growing need for methods that can handle categorical data and detect outliers in this context. This thesis aims to fill this gap by proposing a novel approach for outlier detection in categorical data.

Thus, designing an algorithm to find them has practical value alongside its theoretical interest. Various attempts have been made towards the development of such tool (e.g. [13, 74, 35]) with varying results in accuracy. However, none seems to explore the potentially interesting combination of a classification/isolation tree with a distance based anomaly score.

With the aforementioned foundations in place, this thesis aims to address the following research inquiries: Can an algorithm be developed to identify values within a dataset $\mathcal{D}$ containing n entries and m attributes that exhibit substantial deviations from the majority of values? How well does this algorithm fair across various real-world datasets?

1.2 Thesis Layout

This thesis is organized into six chapters. Chapter 1 provides an introduction to the topic of outlier detection in categorical data, and highlights the importance of this area of research. Chapter 2 provides a comprehensive review of the literature and compares the strengths and limitations of the different approaches while chapter 3 presents a thorough discussion of the preliminary concepts and theoretical analysis. Chapter 4 presents the proposed method for outlier detection in attributes with categorical values. Chapter 5 focuses on the experimental evaluation of the proposed method and chapter 6 summarizes the findings of this project and discusses possible future work directions.

Chapter 2

Related Work

2.1 Background

Anomaly detection for categorical data refers to finding anomalies in data and in particular categorical ones. However, long before anomaly detection in categorical data was discerned as a potentially interesting field of its own, it was thought as an integral part of Data Mining. Frawley et al. [30] define Data Mining , or equivalently knowledge discovery in databases, as the process of extracting implicit and non trivial information from data. This significant and interesting information (according to a user's standard) is called knowledge, while the output of the aforementioned extraction is called discovery knowledge.

There are numerous data mining algorithms as the research community has made a systematic attempt to develop this field. Mining association rules in relational databases is thought to be one of the earliest approaches in tackling this problem. For reference, association rule mining refers to the extraction of interesting correlations of the type $A_1 \wedge A_2 \wedge \cdots \wedge A_j \Rightarrow B_1 \wedge B_2 \wedge \cdots \wedge B_m$ where $A_i, i \in 1, ..., j, B_k, k \in 1, ..., m$ are sets of items. A typical example is the supermarket basket which states that if a consumer buys bread and milk they may also buy cheese [20]. Agrawal, Imilienski and Swami can be considered the forefathers of association rule mining. In their paper [2], association rule mining is divided into two subproblems. That is, create all possible combinations of items in a database that occur over a predefined threshold and, for those, generate association rules. This paper has been an inspiration for many ideas that followed (e.g. [64, 71, 81, 84]). Among them is Apriori [3], a famous algorithm that heavily influenced the data mining community.

The aforementioned algorithms mainly pertain to transactional databases, which as their name suggests, contain transactions or, in other words, subsets of items. However, the evergrowing expansion of the industry towards relational databases required an appropriate support by various data mining tools and languages. Sarawagi et al. [79] were one of the first in integrating relational databases with association rule mining by implementing the Apriori algorithm in SQL-92[1]. Meanwhile, the DMQL language [37] consisted one of the maiden attempts in establishing some sort of systematic language when extracting information from relational databases.

What is more important though, is that this rapid expansion of the data mining field triggered a chain reaction to a plethora of frontiers [36], one of them being the systematic detection of erroneous values in databases. Formally coined "Anomaly detection", this process refers to

[1]SQL-92 was the third revision of the SQL Query Language

the problem of finding values or tuples in a database that do not align with the rest of the entries. Anomaly detection is closely related to noise removal, database cleaning and data deduplication [47].

2.2 Anomaly Detection (AD)

2.2.1 Outlier Definition

Studying erroneous values is not a newly discovered field but has a long history, studied extensively by various scientists [8]. Bernoulli [11] more than 200 years ago agreed with astronomers that were rejecting observations deviating from their expected truth, while Edgeworth [27] later on, defined discordant observations as "those which present the appearance of differing in respect of their law of frequency from other observation which they are combined".

For this thesis Hawkins' definition of an outlier is adopted [39], which characterises one as "*an observation that exhibits such significant deviation from the norm that it raises concerns that it may have been generated by a distinct process or mechanism*".

However, given that the focus of this thesis is on studying syntactical outliers, which are anomalous observations that deviate from the expected syntax or structure of the data, the definition of an outlier provided by Hawkins may be too broad. Therefore, this thesis utilises the terms outlier, anomaly, discordant observation, and erroneous values interchangeably. This approach allows for a more comprehensive analysis of the data and a more nuanced interpretation of the results, particularly in the context of syntactical outliers.

2.2.2 Applications of AD

Outlier detection systems are not contained in a specific domain but are rather spread to a variety of them. Some examples are in Network Intrusion, Financial Fraud and Health Sector.

More specifically, numerous methods have been developed in order to detect anomalies in the domain of networks [80]. There are various Anomaly-based Network Intrusion Detection Systems (ANIDS), such as the Statistical Packet Anomaly Detection Engine (SPADE) [85], Snort [17] and Prelude [97]. With respect to the domain of financial fraud, not only a highly methodical overview of the anomaly detection techniques in various financial sectors is provided [68] but also a framework, coined Anomaly Detection Tool (ADT) [49], with the goal of detecting anomalies in the cryptocurrency market. In the medical care, outlier detection mostly deals with finding erroneous values in patient records. Exemplary of this are discrepancies detection in the time series of electrocardiograms excerpts [59] and general outlier detection in health care records in a big data context [22, 95].

2.2.3 Challenges of AD

Finding outliers in its most abstract form may seem like a straightforward task. However, based on previous work [83] four main problems can be drawn:

- Defining what the normal behaviour is, can be considered an extremely complex task. It requires strong domain knowledge and a large amount of data to classify and distinct erroneous values.
- An outlier in one dataset may be a normal value in another. Thus, persistence of information across various datasets is not feasible.
- Labeled data is very scarce and difficult to obtain.
- In several domains, data is fast-paced which constantly alters the definition of normality.

Challenges in Categorical Data

In addition to the aforementioned problems, Anomaly Detection for categorical data encounters more issues [89]:

- Unlike numerical data where various metrics exist to calculate the distance between two data points (e.g. Minkowski distance, Manhattan distance, etc.), such a comparison is not straightforward for categorical data.
- Computational complexity is a big obstacle, especially for large datasets.

2.2.4 Types of Anomalies

In general, anomalies are divided into three main categories: (i) Point Anomalies, (ii) Contextual Anomalies and (iii) Collective Anomalies [19].

(i) Point anomalies refer to individual points deviating significantly from the rest of the values. They consist of the simplest form of outliers. As an example, let a database contain business expenses for a particular company. If an entry is significantly higher that the average range of expenses then it could signify that this expense is an outlier.

(ii) Contextual anomalies pertain to points which can be considered an anomaly when taking into account a particular context, or equivalently, a particular subset of the data. For instance, a snowfall of 10cm may not be considered a lot for a whole year in a northern European country, but it is especially peculiar if it happens only in May.

(iii) Collective anomalies indicate a particular group of similar values that deviate significantly from the rest of the values when grouped together. Collective outlier detection has been implemented for various kinds of data such as graph data [69], sequential data [87] and spatial data [29]. It should be noted that the point anomalies and collective anomalies can be also thought as contextual anomalies, given an appropriate context.

2.2.5 Outlier Detection Techniques

Outlier Detection Techniques can be broadly classified into two coarse categories: (i) Supervised / Semi-Supervised and (ii) Unsupervised.

Supervised AD

Supervised Anomaly Detection techniques require every entry in a database to be labelled whether it consists of an anomaly or not. Supervised anomaly detection algorithms closely relate to classification techniques, a method which has been extensively studied by the academic community [1]. It can be quite challenging to label completely and accurately an entire dataset. Not only are there potentially millions of records which makes a manual process prohibitive but also, since by definition outliers are rare instances, there is a great imbalance among the two classes (i.e. outliers and non-outliers).

Semi-supervised AD

A more feasible alternative to a complete labelling of all the data points would be a partial label of some normal, non-outlying data points. This labelling indeed paves the way for semi-supervised anomaly detection techniques where given n labeled data points and m unlabeled ones, the goal is to train and develop a model to accurately discern the normal ones from the outliers. Semi-Supervised can be considered a special case of the supervised AD techniques. As is the case with supervised, semi-supervised methods have gained considerable traction in the academic community. More specifically, important work is done on the subject from

Learning from Positive and Unlabeled Examples (LPUE) in the context of PAC Learning as long as sufficient information about the underlying distribution of the data is given [25]. For context, LPUE is based on the famous framework of Valiant [94]. Building upon LPUE, a method to estimate accurate results based on positive (normal) and unlabeled examples is developed [98]. However, utilising exclusively the normal data is not the only approach to semi-supervised methods. Exemplary of this, is the creation of Support Vector Data Descriptor (SVDD) based on both unlabeled and labeled examples. For reference, SVDD [93] is a model that uses the concept of Support Vector Classifiers in order to bound and contain the data of a dataset along an area. Finding outliers in that sense, is more straightforward, since they are nothing more than the data points outside those bounds.

Unsupervised AD

Unsupervised outlier detection techniques require no labelling on the data. Therefore, the applications that utilise these methods are far more numerous than those mentioned previously [16].

However, this unavailability of the data consists a double-edged sword. On one hand, far more and bigger data sets can be utilised since no manual work is needed to label them accordingly. On the other hand, performance metrics (such as false positive rates or Receiver Operating Curve) still require a ground truth set or a domain expert to verify the results.

Hence, choosing the right dataset is crucial in unsupervised learning as the performance of the algorithm heavily depends on the quality and characteristics of the data. The dataset should contain features that are relevant to the problem being solved while also capturing the meaningful underlying structure of the data. It is apparent, that finding such dataset is an extremely complex task. This is one of the reasons why the term "*semantically meaningful outlier datasets*" is introduced for such collections of data [16].

2.3 Anomaly Detection Methods in Categorical Data

The following section provides an overview of the different approaches for outlier detection techniques in categorical data. The focus towards categorical data is justified since sufficient attention has been given to numerical data by the academic community [18].

2.3.1 Transformation to Numerical Data

Perhaps the simplest way to take advantage of the several numerical outlier detection tools would be to transform categorical data to nominal or decimal values. There are various ways of achieving this. For example, Labib and Vemuri [55] represent their data by converting all of the characters of a word to their sum of ASCII values, while Campos et al. [16] use the inverse document frequency (IDF) to reflect the importance of a single word amongst a collection of words. Indicator variables are also utilised to solve this problem based on the well-known mathematical concept of indicator functions [82]. In this approach, a data point is considered an outlier if its outlying score surpasses a user-defined threshold. The score is based on Canberra distance [57] and can be expressed by the following formula:

$$C_\alpha(x, y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}$$

The concept of indicator variables as well as the other aforementioned approaches, albeit noteworthy attempts at solving the general problem, encounter two main issues:

1. The transformation from categorical to numerical values is not trivial nor straightforward. There are multiple ways to do it and there is a great risk of information loss, depending on the approach used. For example, with the previously mentioned ASCII value transformation, the words "it" and "on" are mapped to the same value of 221 even though they are different syntactically and semantically.
2. The efficiency of the transformation to nominal data is decreased as the number of indicator variables increases. An example to illustrate this is the unique tax identification numbers of the citizens of the Netherlands, where each unique value would be mapped to a unique value, which would not only be computationally expensive but also not offer any significant additional value.

2.3.2 Frequent Itemset Detection approaches

A viable alternative to the aforementioned issues would be to use frequent itemset techniques. Indeed, frequent itemset mining approaches are a fair attempt at outlier detection. They involve identifying itemsets, or collections of items, that occur frequently and using them to determine outliers. Frequent Pattern Outlier Factor (FPOF) [41] is a cornerstone of the frequent itemset approaches. The FPOF algorithm first finds all the frequent itemsets using the Apriori Algorithm and based on that computes a score for each data point. On a similar note, an outlieness score can be computed based on the inverse of the infrequent itemsets [70].

However, most frequent itemset algorithms suffer from a phenomenon called pattern explosion, where the number of frequent itemsets increases exponentially. Condensed representation approaches can overcome this problem as they aim to find patters that offer the most succinct representations of the whole set of patterns. A consequence of that are deduction rules that, as their name suggests, deduct upper and lower bounds on the superset of a candidate set [15]. An itemset that can be derived from these bounds is called derivable and thus, poses no interest, while the rest are called non-derivable itemsets. Based on that, FNDI-OD is developed with running times much faster than other FI equivalents [53].

2.3.3 Frequency-based approaches

Frequency-based approaches in outlier detection involve analysing the frequency distribution of data points to identify values that occur less frequently than others. These infrequent data points are considered to be potential outliers, as they do not conform to the majority of the data. A typical algorithm of this approach is Attribute Value Frequency (AVF), which assigns a score for each data point based on the occurrences of a value on the k_{th} attribute $f_k(x)$ [52, 51]. A low AVF score would mean that the object in question tends to be more of an outlier compared to an object with a higher AVF score. AVF's simplicity makes it extremely easy to parallelise, significantly reducing the computational times. Thus, taking advantage of the MapReduce technology the MR-AVF is proposed, which is the parallel version of AVF [54]. AVF though, requires the whole dataset to be available which can not always be the case. Taking this into account, the One-Pass AVF is introduced which in short is AVF in a streaming context, id est when the whole data is not available at once but comes gradually [91]. One-Pass AVF uses the cumulative probability of an attribute instead of the relative frequency and can handle large amounts of data with AUC scores fairly close to AVF as shown by experimental evaluations. Inspired by those, Squares of the Complement of Frequency (SCF) is proposed [90]. SCF in comparison to AVF uses the sum of squares of the complement of the marginal frequency in order to place emphasis between the values occurring frequently and those that are sparse. By construction, SCF is fast, scalable and does not suffer the large computational overheads frequent itemsets algorithms do.

2.3.4 Density-based approaches

A fundamental aspect of most techniques for categorical outlier detection would be that similar observations should be grouped together in the space they are mapped to. As a result, specific areas in that space would have a greater concentration of elements and thus, a bigger density, while outliers should lie in areas with lower density score. Density is closely related to clustering and distance-based methods described in sections 2.3.5 and 2.3.6 respectively.

In the subfield of density-based outlier detection, several methods have been proposed by different groups of researchers. One such method is called Local Outlier Factor (LOF) which assigns an outlieness score for each object in the dataset [14]. LOF mostly deals with data in numerical space and shows promising results in many different datasets. However, albeit its sounding success, it may rule outliers close to some non-outliers that have low density. To address this issue another method called Connectivity Outlier Factor (COF) is proposed, which discerns a data point that has low density from one that is isolated [92]. Isolativity pertains to the fact that an object is related to other objects and implies low density while the opposite does not necessarily hold. Based on those algorithms, k-LOF is proposed which is the categorical space equivalent technique [96]. The focal point of k-LOF is that the density is measured by computing the radius of each data points through graph walks. In other words, the amount of walks that go through other data points and the amount of walks that come inwards. The larger the number of walks for a given point, the more certain the system is that this point is a local outlier.

2.3.5 Clustering-based approaches

Clustering-based approaches refer to techniques that use clustering algorithms to identify anomalous data points within a dataset. Clustering algorithms group similar points together based on their similarity in a feature space. Outliers are those data points that do not fit within any cluster or belong to a small, isolated cluster.

Once again, numerical-based methods constitute the inspiration for further improvements down the line of this particular approach. The well known k-means algorithm [61] is utilised to solve the problem of categorical outliers [75]. This approach transforms categorical attributes to binary (0 or 1) based on their coexistence in an observation and then it applies the k-means algorithm in a numerical context. Arguably though, this implementation imposes a massive computational overhead for datasets with just as few as a couple of dozen attributes. Based on the k-means algorithm and its improvements [4], the k-prototypes algorithm is proposed which deals with categorical as well as numerical data [45]. K-prototypes selects k initial entries from the dataset as representatives (i.e. prototypes) for each of the k-clusters and, based on those, assigns the rest of the objects of the dataset. When all objects have been assigned to a cluster, a new prototype is set and the process starts anew. A simplified version of k-prototypes is k-modes which only deals with categorical data [44]. K-modes follows the same process as in k-means but, as the name suggests, uses the mode of a categorical attribute based on the relative frequency of the category.

On a similar note, the Cluster Based Local Outlier Factor (CBLOF) is proposed that clusters the data using k-means [40]. CBLOF makes a distinction between large and small clusters and, based on that, computes a similarity score for any record. Similarly, RObust Clustering using K-links (ROCK) is developed, which is an agglomerative hierarchical clustering algorithm [34]. In short, ROCK fetches a random sample in the database, creates clusters based on those and iteratively merges those clusters based on a goodness measure. The goodness measure depends on the number of common neighbours between two clusters. Building upon the previous two algorithms, the Ranking-based Outlier Analysis and Detection (ROAD) is suggested, which is a two-phase clustering algorithm [88]. In the first step, ROAD computes the densities of the objects by using AVF (section 2.3.3). In the second step, ROAD splits

the dataset with the help of k-mode algorithm into clusters. Clusters that contain more than a pre-defined α% of observations are deemed as big clusters.

2.3.6 Distance-based approaches

Distance-based outlier detection is a statistical technique used to identify unusual or anomalous data points in a dataset. This method involves calculating the distance of each data point from the other points in the dataset and comparing it to a threshold value. Points that have a distance greater than the threshold are considered outliers.

The concept of distance-based methods on categorical data is heavily influenced by its numerical counterpart. In particular, Knorr et al. [50] pave the way by introducing DB(p,D)-outliers. An object O is a DB(p,D)-outlier if at least a fraction p of the objects in T lies greater than distance D from O. Based on that approach, various other scientists are proposing innovative alternatives. For example, one approach orders the points in such a way, that the top n-points are the n-outliers in a dataset [76]. An extension of this is proposed which utilises only a subset of the dataset needed to predict the outliers [5].

However, the very nature of categorical data renders most metrics such as Manhattan distance or Minkowski distance largely impractical, since there is no straightforward way to map categorical data to higher dimensional space. An attempt to tackle this problem is done by the introduction of the Common Neighbor Based distance (CNB) [58]. The main idea of the CNB algorithm is that entries of a dataset that are similar should have many attributes with the same value (i.e. *neighbors*). Based on the concept of neighbourhood, this new metric is proposed that measures the distance between two entries. Therefore, the outliers are the top p points whose average distance to their k nearest neighbors is greatest.

Leveraging this last notion Orca is proposed, a fairly simple and near-linear time algorithm [7]. In short, Orca uses randomization techniques and pruning strategies to detect outliers. More specifically, Orca shuffles randomly the dataset and if the k-th neighbor of a data point has a distance smaller than a cutoff value c then it is pruned from the process. The authors show that this randomization enables the algorithm to process a large number of non-outliers which are then pruned, thus speeding up the computations significantly. However, later work states that these aforementioned claims of the computational times are highly improbable for most datasets [32]. More specifically, the expected number of distance computations for a data point x that is not considered an outlier is 1 / $\Pi(x)$, where $\Pi(x)$ is a variable denoting the probability that a randomly selected data point lies within the cutoff threshold. The claim of near-linear time complexity holds only if $\Pi(x)$ is a constant, which is highly improbable apart from a few, carefully selected datasets.

Thus as a viable alternative, Recursive Binning and Projection (RBRP) algorithm is proposed [32]. RBRP comprises of two steps: First, it partitions the data points into clusters. Afterwards, for each data point, the points inside the cluster it belongs to are searched first and then the closest ones and so forth. Thus, a list of top k outliers is created based on this notion of distance. Empirical evaluations show that it fairs better than Orca, but in order for it to work, it either requires the whole dataset to be loaded in memory or an extreme amount of I/O. These issues are effectively addressed by iOrca, which builds on top of Orca with the addition of an effective indexing technique [12]. iOrca selects a random data point, which based on probability theory should be an inlier and reorders all the data points based on its distance with it. This reordering makes, in general, iOrca faster than Orca, although this is not a complete guarantee. Since the randomly selected data point is chosen by chance, in the worst case iOrca fairs worse that Orca.

2.3.7 Pattern Extraction and Pattern Matching approaches

In contrast to numerical or nominal data, a major advantage of well structured and normal categorical data is that they follow a specific and expected pattern. For example, the name of a country in most cases starts with an uppercase and it does not contain digits. A deviation of a value from this predetermined pattern suggests a strong inclination of the value being an outlier.

Extracting such patterns in the first place is no easy task. Typically, string patterns are succinctly described by regular expressions (regex) which can be learned by Determinstic Finite Automata (DFAs). For reference, a DFA can be described as a "finite-state machine that accepts or rejects a given string of symbols by running through a state sequence uniquely determined by the string" [42]. However, it is established that this approach is more or less infeasible since the problem is NP-hard [33]. This problem though is surpassed by XSYSTEM, which is "nothing more than a method to learn and represent syntactic patterns of datasets as data structures, called XTRUCTURES" [46]. By construction, XSYSTEM is as expressive as a Deterministic Acyclic Finite State Automaton which is asymptotically less complex than a DFA. Besides a syntax-based outlier detection, XSYSTEM offers automatic label assignment as well as a brief summary of the attribute columns.

Regular expressions and DFAs, though, are not the only way to describe a string sequence. For example, by taking advantage knowledge in stochastic processes and Markov chains, a crucial property can be noticed that holds across several domains and fields [77]. This property, named short memory, states that for a given sequence $s = s_1 s_2 \cdots s_l \ \exists \ \ L < l$ such that $P(s_l|s_{l-k} \cdots s_{l-1}) \approx P(s_l|s_{l-L} \cdots s_{l-1})$, $\forall k > L$. In other words, previous states before a certain point do not have a significant effect on the current state. The length of the cutoff point varies per domain. Based on the short memory property, a Probabilistic Suffix Automata (PSA) model can be created, in order to model different transitions between states with the help of a tree structure. A slight variant of PSA is the Probabilistic Suffix Tree (PST), which computes efficiently the sequence s as the joint probability $P(s_1 s_2 \cdots s_l) = P(s_1)P(s_2|s_1) \cdots P(s_l|s_1 s_2 \cdots s_{l-1})$. Based on this equation, a similarity function SIM_O can be defined in order to measure how much a sequence s belongs to a tree T instead of being a random sequence [9, 10]. This knowledge can be then applied to strings in order to detect outliers in sequential databases [87]. More specifically, a faster and more efficient PST can be created which in general puts the outliers near the root of the tree. Values are then ranked based on a similarity measure SIM_N which is the normalised version of SIM_O. The n lowest SIM_N values are considered outliers.

However, even if the problem of pattern extraction is somehow surpassed or neglected, datasets without errors or missing values are a minority and not the norm. Extracting patterns from incomplete datasets leads to inaccurate results. Therefore, an appropriate approach has to be drawn that deals with these inconsistencies. A hasty solution would be to simply consider the erroneous values as an empty string and extract patterns from the rest of the entries. For that reason, Oracle database systems have full support for NULL values, while the Python programming language treats None (i.e. nulls) as first-class citizens. However, the problem is not simply null values but rather non-null values that are semantically equivalent to null values. For instance, if a user needs to specify an unknown date in a system, they may input "NA," "01/01/1970," or even "0". Such values are known as Disguised Missing Values (DMV) and can be particularly challenging to detect, depending on how they were generated. Pearson [72] first introduces the problem of DMV and highlights its impact on various datasets. Three models are proposed in order to handle efficiently DMVs: (i) MCAR, (ii) MAR and (iii) NMAR [60].

1. MCAR (Missing Completely At Random) assumes that the probability of a value missing from a specific attribute is independent of the presence (or absence) of any other

attribute.

2. MAR (Missing At Random) model lifts the independence constraint and assumes that a particular value that is missing depends on the presence of another value.
3. The NMAR (Not Missing At Random) model assumes that the missingness of data depends on the missing values themselves. Detecting NMAR constitutes an extremely hard task.

A heuristic approach for finding DMVs in relational databases is based on Embedded Unbiased Samples (EUSs) [43]. Specifically, if a value u is a DMV in an attribute A_i of a table T, then it should also contain a large subset $\tilde{T}$ (i.e. an EUS) that does not contain attribute A_i but has a probability distribution similar to T. Partially based on this, a missing values detector, coined FAHES, is developed [74]. FAHES makes a distinction of DMVs between syntactic or numeric outliers and inliers. For reference, a syntactic outlier is a value with a syntactic structure that varies significantly from the rest of the patterns. In FAHES, two main techniques are used to find syntactic outliers: (i) syntactic pattern discovery and (ii) repeated pattern discovery. Syntactic pattern discovery finds the optimal pattern and reports values that do not fit this pattern, while repeated pattern discovery finds out values that contain repetitions such as *123123123* for a numeric attribute or "*xyzxyz*" for a text attribute. Numeric outliers are found by taking advantage of well known methods in literature [38], whereas inlier DMVs are found only under the MCAR or MAR hypothesis, as the NMAR outliers are extremely difficult to detect. The main idea behind the inlier DMV search is that DMVs are values that occur frequently. Therefore, if frequent values are replaced sequentially by null values and the two datasets follow the MAR or MCAR models, then the (replaced) values can be considered as DMV candidates.

On the topic of syntactic pattern discovery, another approach is SURAGH, a method to discern records based on their syntactic structure [35]. SURAGH analyses the dominant syntactic row patterns for each schema and reports as outliers the entries that do not conform to this pattern. Bouganim et al. [13] extract DMVs through text embeddings and classification. Candidate DMVs are first preprocessed (e.g. removal of stopwords etc.) and then the TF-IDF scheme is applied, which gives a higher score to more frequent words, discerning the DMVs from the rest of the values. Another framework for efficient outlier detection is DBoost, which in general, comprises of two steps: (i) tuple expansion and (ii) analysis of the expanded tuples [73]. Tuple expansion refers to the process in which semantically rich candidate features are added to each tuple of the dataset. The analysis is done by training three different kind models: histograms, Gaussian and multivariate Gaussian mixtures. Gaussian and multivariate Gaussian mixtures assume that the underlying distribution of data points follows a normal distribution, while the histogram model makes no such assumption. Using these models, outliers are those entries that deviate significantly from them.

In addition to those, Raha is developed, which is a system responsible for detecting errors in datasets [63]. Raha provides outstanding results by combining various outlier detection strategies. However, this approach cannot offer any guarantees on its performance. Systems such as *Rotom* and *Baran* outperform Raha by using deep-learning techniques [66, 62].

2.4 Limitations of current work

As with any approach, each of the outlier detection methods has its own limitations. These limitations may apply to specific datasets, higher computational costs, or explainability of results. A brief overview of the limitations for each method will be summarised in the following section.

As mentioned in 2.3.1 approaches that transform the data to ordinal or nominal data, suffer from two main things: information loss and efficiency loss. Tables 2.1 and 2.2 illustrate this

Value	ASCII Transformation
live	432
long	432
wild	432
take	421
Text	421

Table 2.1: Information Loss

Value	Nominal Data Transformation
Apple	1
Banana	2
Coconut	3
Apricot	4
Kiwi	5
Watermelon	6

Table 2.2: Efficiency Loss

fact with the help of a toy dataset. In table 2.1, significant information loss is observed since groups of words that are neither semantically nor syntactically related are mapped to the same value. In table 2.2, no additional value is added since the 6 unique values are mapped to 6 unique nominal values.

As for frequent itemset approaches, most methods suffer from pattern explosion which occurs when the number of frequent patterns or association rules generated by an algorithm becomes too large to be practically useful or interpretable. While some approaches, such as non-derivable itemsets, attempt to address this issue, there are still computational challenges when the minimum support threshold σ is set too low (e.g. around 5%).

Frequency-based approaches, such as traditional outlier detection methods based on frequencies or counts, typically do not consider the syntactic structure of outliers and primarily focus on the frequency or occurrence of values. These methods identify outliers based on the notion that rare values are more likely to be outliers. However, this approach may overlook the importance of syntactic structure in distinguishing outliers. For instance, consider a dataset as shown in table 2.3 . Notice that the values “CY” and “ERROR” both appear with the same frequency. In a frequency-based approach, “CY” and “ERROR” would be considered equally rare and potentially labeled as outliers. However, if “CY” follows the same syntactic structure as the other patterns in the dataset, it may not be an outlier based on its syntactic properties. In contrast, “ERROR” could be an outlier as it deviates from the expected syntactic structure.

Country Codes
GR
GR
GR
GR
CY
ERROR
EG
EG

Table 2.3: CY and ERROR each appear once

As for the density-based approaches, LOF suffers from increased computational complexity, which is a major issue when dealing with real-world datasets that contain millions of rows. Meanwhile, other methods, such as k-LOF, still require from the user to choose an appropriate value of k.

Clustering-based approaches are no exemption to some problems. K-means related algorithms

still require a valid k to be chosen by an end-user, which can be extremely difficult in some situations. The selection of an inappropriate value for k can lead to suboptimal clustering results, affecting the accuracy of outlier detection. If the chosen k is too low, the clusters formed may not adequately represent the underlying structure of the data, resulting in some outliers being misclassified as regular data points. On the other hand, if the chosen k is too high, it may lead to overfitting and the creation of small, fragmented clusters that do not capture meaningful patterns.

As for distance-based approaches, the quality of outlier detection heavily relies on the ability of the chosen distance measure to accurately capture the dissimilarities or similarities between data points. If the distance measure fails to adequately separate normal data points from outliers, the performance of the outlier detection algorithm may be compromised.

Furthermore, one of the main difficulties in pattern extraction and pattern matching is the computational complexity of finding a regular expression that accurately captures the underlying patterns in the attribute. The task of determining a concise regular expression that represents the entire attribute is known to be NP-hard, which means that it becomes increasingly difficult to solve as the size and complexity of the dataset grow. This computational complexity poses challenges in terms of efficiency and scalability when applying these approaches to large-scale datasets. Finally, the definition of what constitutes an erroneous value or an outlier is often subjective and context-dependent. While pattern extraction and matching approaches can identify values that deviate from established patterns, they do not provide a formal definition of what is considered erroneous or anomalous. This lack of a precise mathematical formulation for erroneous values makes it challenging to determine the threshold or criteria for identifying outliers solely based on patterns. The interpretation of patterns and their relevance to identifying outliers often requires domain expertise or manual inspection by data analysts.

Chapter 3

Preliminaries and Theoretical Analysis

3.1 Preliminaries

Let D be a database containing n tuples $x_1, \ldots, x_n$. Each tuple x_i contains m values $\{u_1, \ldots, u_m\}$, where u_j denotes the value of attribute j. A value u_i is sequence of characters $u_i = (c_1, \ldots, c_k)$ where each c_q denotes the character of the value u_i at position q.

Each c_q belongs to an alphabet class, or equivalently, $c_q \in \mathcal{A}_\mathcal{C}$, where $\mathcal{A}_\mathcal{C}$ indicates the alphabet $\mathcal{A}$ of a class $\mathcal{C}$. There are 5 possible distinct classes: Uppercase (U), Lowercase (l), Digits (d), Whitespace (w) and Special Characters (s). The various classes are non overlapping; $\mathcal{A}_\mathcal{X} \cap \mathcal{A}_\mathcal{Y} = \emptyset$. However, an important note is that a value u_i can belong to more than one class.

In addition, let the alphabet of U class and l class be the uppercase and the lowercase characters of the English language respectively. As for the d class, let the alphabet be the numbers {0,. . . , 9} and for the s class let the alphabet be the special non-alphabetical Unicode characters. The alphabet of w class has a cardinality of 1 and contains only the empty string ϵ. As a result, the power set $\mathcal{P}$ encapsulates all the possible combinations for a random value u_k. The arrangement of classes is illustrated in Figure 3.1 through a generalization tree.

Generalised String

Given a value $u_i = (c_1, \ldots, c_k)$, a generalised string is the transformation of the value u_i to v_i, where $v_i = (a_1, \ldots, a_k)$ and $a_h,\ h = 1, \ldots, k$ denotes the character class c_h belongs to.

Table 3.1 provides an example of class belonginess as well as their generalised pattern for some given values.

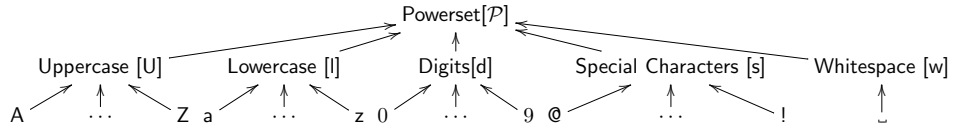


Figure 3.1: A Generalization Tree

Value	Classes it belongs to	Generalised String
Las Vegas	U,l,w	UllwUllll
Hell0!	U,l,d,s	Ulllds
5473	d	dddd
555 012	d,w	dddwddd

Table 3.1: Example values and the class they belong in

Semi-Generalised String

Following the generalised string, a semi-generalised string s_k is a partial transformation of the value u_i to a generalised v_i. Equivalently,

$$s_k = \begin{cases} c_i & i = 0, \ldots, k \\ a_i & i = k+1, \ldots, l \end{cases} \tag{3.1}$$

Evidently, a semi-generalised string is dependent on the parameter k. Table 3.2 shows a visual example of this transformation.

Value	Semi-generalised String	k
Las Vegas	LaswUllll	2
Hell0!	Hell0s	5
5473	5ddd	0
555 012	555 012	6

Table 3.2: Example values and their semi-generalised transformation alongside their k values

Specificity Level

Given a semi-generalized string s_k, the specificity level refers to the zeroth indexing pointer that indicates the character at which the word starts to become more generalized. The notation extends to negative numbers as well (e.g. -2, -1, ...), indicating that the entire word is generalized. In table 3.2 the k-value refers to the specificity level.

Regexed String

Given a value u_i, a regexed string r_i is the replacement of the repeated characters of u_i with a plus symbol (+) except for the first occurrence of each character. Table 3.3 provides an example of regex transformation for some exemplary values.

Value	Regexed String
Las Vegas	Ul+wUl+
Hell0!	Ul+ds
5473	d+
555 012	d+wd+

Table 3.3: Example values and their regex transformation

Representative

A representative *rep* of a group of strings $(s_1, \ldots, s_n)$ is a string s of which prefix is the same for all strings s_i. An example is presented in Table 3.4.

Values	Representative
Neon New York New	Ne
Apple Tomato Coconut	-

Table 3.4: Example values and their representative

Syntactic Structure

One of the distinguishing features of FIONA is its interpretation of syntactic structure, which differs somewhat from the concept defined by Chomsky [21]. In FIONA, syntactic structure refers to the specific arrangement of character classes within a value. This structure is determined by the application of regular expressions (regex) or generalizations of the word through transformations.

For instance, consider the proper noun "New York." In FIONA, its syntactic structure can be described as follows: an uppercase character followed by some lowercase characters, a whitespace character, another uppercase character, and finally, some lowercase characters. This arrangement of character classes represents the syntactic pattern of the value.

Semantics

In contrast to syntactic structure, which emphasizes the arrangement of character classes within a value, semantics pertains to the meaning conveyed by the value itself. While syntactic structure primarily considers the structural patterns of a word, semantics focuses on the interpretation and significance of that word.

For example, consider the words "New York" and "Asd Fghj." From a syntactic perspective, both words have the same structure of an uppercase followed by some lowercases and a whitespace character. However, their semantics differ significantly. "New York" carries the meaning of the city of New York, while "Asd Fghj" is a nonsensical combination of characters that does not convey any specific meaning.

It is important to note that FIONA primarily focuses on capturing and detecting outliers based on syntactic patterns. While it excels at identifying anomalies that deviate from expected syntactic structures, FIONA's capability to capture semantic outliers is limited. If an

outlier possesses the same syntactic structure as the normal patterns in the dataset, FIONA may not be able to differentiate it solely based on semantics.

Custom Rooted Tree

Furthermore, let a set of n values be $\{u_1, \ldots, u_n\}$. $\mathcal{T}$ is a rooted tree constructed in the following way:

- The zeroth level (sp = -2) contains a node with the values of the entire set and splits the values (in the next level) based on their class membership
- The first level (sp = -1) splits the values (from the same class) based on their length.
- The second level (sp = 0) and the following ones split the values based on increasing character match.

By construction, the leaf nodes of $\mathcal{T}$ do not contain overlaps and can thus recreate the initial value set.

3.2 Theoretical Analysis

3.2.1 Tree Analysis

Let a tree $\mathcal{T}$ constructed as outlined in Section 3.1. Let a dataset contain n entries with m attributes. Let the value within each attribute be distinct from the others.

Lemma 1 *The maximum number of nodes on the first level of the tree $\mathcal{T}$ is $2^{|\mathcal{P}(\mathcal{C})|} - 1$.*

Since the number of different classes $\mathcal{C}$ is a finite set, then the number of all of the subsets of it is $2^{|\mathcal{P}(\mathcal{C})|}$. In addition, by construction, each node carries exactly one class. Therefore, the maximum number of nodes is $2^{|\mathcal{P}(\mathcal{C})|}$.

However, the powerset $\mathcal{P}$ contains also the empty set $\emptyset$, which in this case is the same as the whitespace class $\mathcal{C}_w$. Therefore, the maximum number of nodes on the first level of the tree $\mathcal{T}$ is $2^{|\mathcal{P}(\mathcal{C})|} - 1$.

Corollary 1.1 *Since there are 5 distinct classes, the maximum number of nodes on the first level of tree is $2^5 - 1 = 32 - 1 = 31$.*

Lemma 2 *Let h be the number of subsets of classes present in a tree and let $l_{\mathcal{C}}$ be the number of different lengths of words belonging in subset $l_{\mathcal{C}}$. Then the number of nodes on the second level is $\sum_{i=1}^{h} l_{\mathcal{C}}$.*

By construction of the tree $\mathcal{T}$, there are h nodes on the first level and for each node n_i, $i = 1, \ldots, h$ there are l_i nodes which correspond to the words of varying length belonging in the subset i. Thus the number of nodes is $\sum_{i=1}^{h} l_{\mathcal{C}}$.

Corollary 2.1 *The number of nodes on the second level is n if and only if the n (distinct) values are of different length per subset class.*

By construction, and and the pidgeonhole principle, the result is proven.

Corollary 2.2 *The maximum number of nodes on the second level is n.*

By construction, no node can carry no value from the dataset. Thus the number of nodes on the second level is $\leq n$. Thus, $\max |nodes| = n$.

Extending this notion:

Lemma 3 *Every level is bounded by n.*

Detrimental for levels 1 and 2 since $2^{|\mathcal{P}(\mathcal{C})|} - 1 \leq n$ and $\sum_{i=1}^{h} l_{\mathcal{C}} \leq n$. For levels ≥ 3, by construction no value can belong to more than one node and by pigeonhole principle the number of nodes are at most n.

Theorem 4 *The number of nodes in the whole tree is bounded by* $l_{max} \cdot (n+1) + 2^{|\mathcal{P}(\mathcal{C})|} - 1$, *where* l_{max} *is the longest word in the database.*

Proof: There are $l_{max} + 2$ levels in the tree, each bounded by either n or $2^{|\mathcal{P}(\mathcal{C})|} - 1$.

Thus a bound on the total number of nodes on the tree is $l_{max} \cdot n + n + 2^{|\mathcal{P}(\mathcal{C})|} - 1$.

Chapter 4

The Categorical Outlier Detector

In the following section a detailed description of the system FInding Outliers iN Attributes (FIONA) is presented[1]. It can be summarised by the following three components: (i) Input, (ii) Analysis and (iii) Output.

In short, it finds contextual syntactical outliers by creating a specific tree structure and assigning a custom scoring function on its leaves. The scoring is a similarity function that does an all pair comparison among all the leaves. It is reiterated once more that FIONA primarily focuses on syntactical outliers, meaning it may not capture semantic outliers if their syntactical patterns closely resemble those of normal values.

Figure 4.1 provides a high level overview of the system.

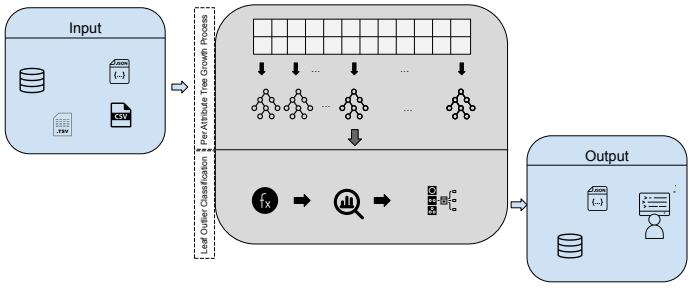


Figure 4.1: FIONA Architecture

4.1 Architecture

Before delving deeper into the high level analysis of the system, a short overview of the underlying infrastructure is given. The infrastructure consists of two main components: (i) the back-end and (ii) the front-end. The back-end is developed in Python and implements

[1]The full implementation can be found in Github: https://github.com/ThanosTsiamis/MasterThesis

most of the logic in categorical outlier detection. Meanwhile the front-end is developed in TypeScript and is responsible for the presentation and user interface of the application. To ensure that the application can run smoothly across different operating systems, both the front-end and the back-end are containerised in a Docker environment. In general, since it can be especially hard to memorise specific commands for different jobs, designing a user-friendly interface is of uttermost importance. Therefore, the whole system can be considered as a black-box for the end-user and will be accessed only though a graphical user interface.

4.2 Methodology

As highlighted earlier, the core structure of the framework revolves around three pivotal components: Input, Analysis, and Output. These components collectively delineate the key stages through which the framework functions, with each encompassing specific functionalities. This segmentation facilitates a systematic and organized approach to outlier detection. In the following sections, the details of each component will be explored, revealing their distinct roles and contributions to the overarching aim of precise and effective categorical outlier detection.

4.2.1 Input

FIONA supports as an input a plethora of file formats and encodings. Each dataset is loaded into main memory and each attribute is fed sequentially into the next step. For the end user this is done through the graphical user interface where they can upload their dataset and inspect it before it gets sent to the back-end. Note that the process of sending attributes to the back-end can be also done in parallel, if enough resources are available, since the search for outliers is confined each time in a single attribute space.

Figure 4.2 shows a screenshot of the application.

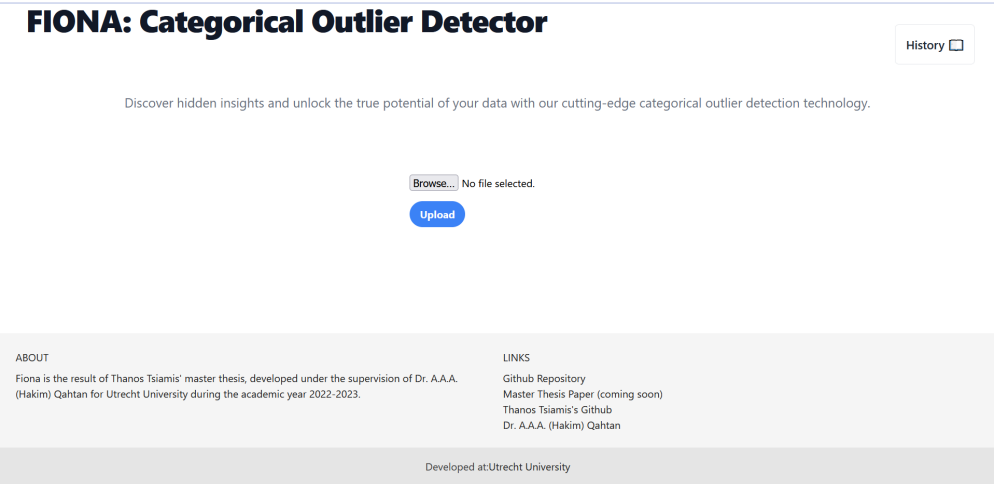


Figure 4.2: FIONA's Welcome Screen

4.2.2 Analysis

The analysis phase of the system can be roughly divided into three main components: (i) Tree grow algorithm, (ii) Matrix scoring function and (iii) Outlier extraction. The general scheme of the algorithm is given in Algorithm 1.

Algorithm 1: Find outliers in a single attribute

Data: An attribute of the dataset
Result: A list of values that were reported as outliers
attribute ← *read_column* ; `/* Loads column into main memory */`
tree ← *tree_grow*(*attribute*);
leaves ← *tree.get_leaves*;
scoring_matrix ← *scoring_function*(*leaves*);
outliers ← *extract_outliers_from_matrix*;

Initially, the algorithm loads the attribute of the dataset into main memory and constructs a tree based on it. The leaves of the tree represent the values of the column, using the concepts of generalised patterns and specificity level as described in section 3. The tree growth algorithm is explained in more detail in subsection 4.2.2. At line 4, a scoring function is used to assign scores between each pair of leaves. The scoring function is extensively described in section 4.2.2.

Consequently, a $k \times k$ matrix is generated, where k represents the number of leaves created by the tree. Using this scoring matrix, leaves that exhibit a different syntactic structure compared to the rest are identified as potential outliers, while the remaining leaves are regarded as patterns.

However, the algorithm described in 1 applies only for a single attribute. Therefore, the process is repeated for every attribute.

As mentioned in 4.2.1 this process can be also done in parallel since the values of a particular column don't interact whatsoever with the values of another column.

Tree Grow Algorithm

The algorithm in its implementation is mostly a conventional tree growing algorithm with some custom made rules that pertain to its structure. A general overview is given in algorithm 2.

The algorithm originally considers the entire attribute as the root of the tree and uses a stack as a data structure in order to keep track of the node at hand. The current node (i.e. head of the stack) is then based on its specificity level. For specificity level of -2 the node's data is split based on their class belonginess. For specificity level of -1 the splits are based on the length of each value. Figure 4.3 offers a visual example.

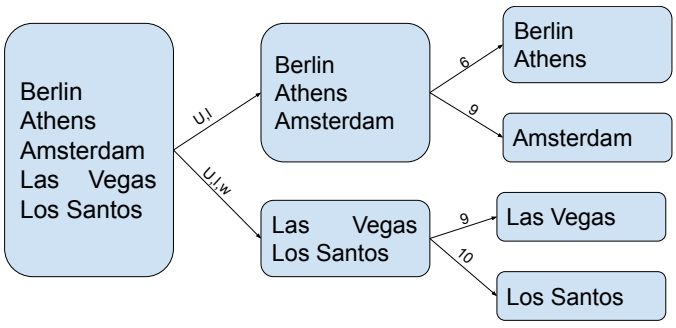


Figure 4.3: Splits done for specificity levels -2 and -1

In this visual example the words "Berlin", "Athens" and "Amsterdam" are grouped together on the first level (-1 specificity level) since they all belong in the class {U,l} while the words

Algorithm 2: Tree Growing Algorithm

Data: A single attribute of the dataset
Result: A custom tree
root ← *attribute*;
node_stack ← [*root*];
while *node_stack is not empty* **do**
    current_node ← *node_stack.pop*();
    check_if_any_pruning_rules_apply();
    specificity_level ← *current_node.get_specificity_level*();
    if *specificity_level == -2* **then**
        children ← *make_split_based_on_classes*;
    else if *specificity_level == -1* **then**
        children ← *make_split_based_on_the_length_of_strings* ;
    else
        children ← *make_split_based_on_the_representative* ;
    end
    node_stack.push(*children*);
    specificity_level ← *specificity_level* + 1;
end
return root;

"Las Vegas" and "Los Santos" both belong in class {U,l,w}. As for the second level (specificity level of 0) "Las Vegas" has a length of 9, hence creating a leaf of its own.

For specificity level greater or equal to zero, the splits are based on the concept of representatives. Once again, figure 4.4 shows a visual example.

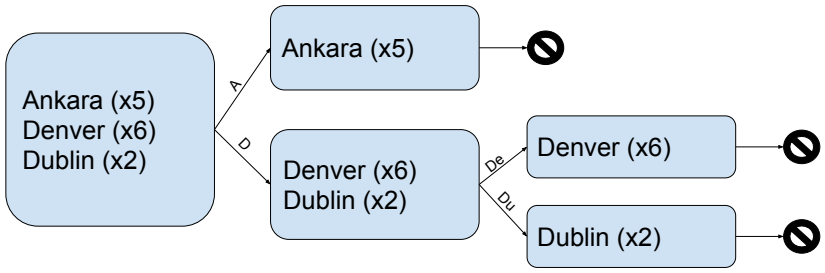


Figure 4.4: Splits done for specificity levels of 0 and 1

In this example, all of the data in the current node belongs to the class {U,l} with a length of 6. The parentheses denote the frequencies of the values. Let the *ndistinct* parameter be 2, id est stop when less than two distinct values are in a node. At the first split (specificity level of 0) the 3 distinct values are split into two separate groups: those that have a representative of "A" and those that have a representative of "D". Note how the specificity level matches the representative in the sense that the first 0+1 characters are the exact values of the latter. Therefore, "Dublin" and "Denver" belong to the same node. In the subsequent level (specificity level of 1), the split for these two cities separates them into different nodes, one with a representative of "De" and the other with a representative of "Du". The "A" node is not divided any further due to the *ndistinct* values restriction. The same holds for the "Du"

and “De” nodes. Tree growing restrictions are described in more detail in section 4.2.2.

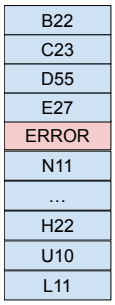


Figure 4.5: Case (i): Syntactic Outliers differ from the rest of the patterns

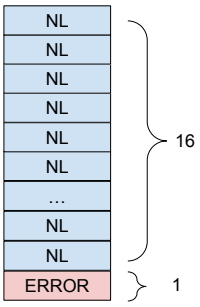


Figure 4.6: Case (ii): Outliers have less frequency than the normal values

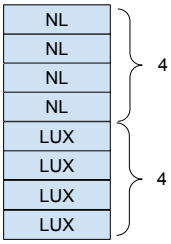


Figure 4.7: Case (iii): Strictly counting on frequencies creates some problems

Matrix Scoring Function

In general, the goal of this process it to extract outliers given a specific context. The specific implementation of FIONA is set aside for now, while an important assumption is being made that, in the most generic case, an outlier is characterized by three important quantities when compared to other elements in the database:

(i) In its most general sense, an outlying element is distinguished from a typical value by its syntactic structure. Essentially, the disparity between an outlying string and a non-outlying string is typically greater than that between two non-outlying strings. It’s important to note that this assertion isn’t always universally true since there may be situations where it doesn’t apply, but anecdotally it is generally applicable in most cases.

(ii) Outlying values occur less frequently than normal values. For instance, let m_1 and m_2 be the frequency of some normal values in a dataset and m_3 be the frequency of some outlying value. Generally, it is expected that $m_1 \times m_2$ will be relatively greater than $m_1 \times m_3$ or $m_2 \times m_3$, implying that a larger product is less likely to be an outlier.

(iii) However, there are some exceptional situations that undermine the previous notion. For instance, consider two cases: (m_{i_1},m_{j_1})=(16,1) and (m_{i_2},m_{j_2})=(4,4). In both cases, the product is identical, but it’s clear that m_{j_1} is more of an outlier than m_{j_2}. To address this, a subtraction is introduced, specifically, $|m_{i_1}\text{-}m_{i_2}|$. To maintain consistency with (ii), where a smaller number indicates an outlier, this difference is included in the denominator, i.e., $\frac{1}{|m_{i_1}-m_{i_2}|+1}$.

The previous 3 points can be now summarised in a single formula:

$$f(i,j) = \frac{1}{d^2} \times (m_i \times m_j) \times (\frac{1}{|m_{i_1} - m_{i_2}| + 1}) \tag{4.1}$$

where, m_k denotes the frequency of the value k and d the squared distance between i and j in the tree structure. Note that the distance counts edges and not nodes. Therefore, a distance from leaf i to leaf i is a self referential node of 1 edge.

The scoring formula used in FIONA generates a score for every pair of leaves in the dataset. As a result, an $n \times n$ matrix is created, where n represents the number of leaves. It is worth noting that the scoring function f exhibits symmetry, meaning that $f(i,j) = f(j,i)$ for any pair of leaves i and j. This symmetry property extends to the scoring matrix itself, resulting in a symmetric matrix. This reduces the computational burden by half for filling the scoring matrix.

Returning to FIONA, the leaves of the tree contain all the information needed to compute the formula 4.1. The data in the leaves already include the frequencies of the values, thereby avoiding the need for an expensive lookup in the original dataset.

Let sp_sum be the sum of specificity levels of the two strings of representatives. For example, let str_1="2ddd" and str_2="555dd", then their specificity sum is 0+2=2. Once again, a small reminder that for better coding practices the specificity sum follows a zero-based numbering.

As for the distance, one could apply a conventional algorithm since a tree structure is a graph, and the leaves are the nodes of the graph. However, implementing such a graph-finding algorithm is expensive, especially since it is performed for all pairs of leaves. Consequently, attention is once again drawn to the concept of representatives, as they provide a faster alternative. The distance function is as follows:

$$d(s_1, s_2) = \begin{cases} sp\_sum - 2 \times char\_pos & s_1, s_2 \text{ in same class and } |s_1| = |s_2| \\ sp\_sum + 2 & s_1, s_2 \text{ in same class but } |s_1| \neq |s_2| \\ sp\_sum + 4 + p & \text{otherwise} \end{cases} \tag{4.2}$$

where, p and $char\_pos$ are penalty functions and character difference functions respectively defined as follows:

$$p(s_1, s_2) = \lfloor \frac{max\_d}{sp_1 + 1} \rfloor + \lfloor \frac{max\_d}{sp_2 + 1} \rfloor \tag{4.3}$$

The max_d denotes the max depth of the tree and the $\lfloor\rfloor$ indicates the floor function.

A penalty is included in the distance function to ensure that potential outliers, which would normally be close to the root of the tree (and therefore close to everything on average), are pushed towards the bottom of the tree. Selecting a value greater than the maximum depth of the tree does not provide a significant advantage to the penalty, as no two nodes can be more than $2 \times max\_d$ distance apart. A leaf that is originally closer to the root would correspond to a lower specificity level, resulting in a higher penalty. For example, let s_1 be "66ddd" and s_2 be "AAAll" the leaf representatives of a tree with a depth (max_depth) of 6. Then the penalty function equals to 3+2=5.

As for the $char\_pos$ it is the minimum index in which the characters of two strings s_1 and s_2 differ.

$$char\_pos(s_1, s_2) = \min_{i=1,\ldots,|s_1|} \{u_i \neq v_i | u_i \in s_1, v_i \in s_2\} \tag{4.4}$$

For example, let str_1 be "Kllllld" and str_2 be "Kairld". Then the *char_pos* is 1 since the strings differ at index 1.

Taking all these into account algorithm 3 summarises the matrix scoring function.

Algorithm 3: Matrix Scoring Algorithm

Data: The k leaves of the tree
Result: A k×k matrix with the score of every pair of leaves
matrix ← create a 3 dimensional empty matrix of size $4 \times |leaves| \times |leaves|$;
for *each* $leaf_i \in leaves$ **do**
  for *each* $leaf_j \in leaves$ **do**
    matrix[0][i][j] ← $leaf_i.mass \times leaf_j.mass$;
    matrix[1][i][j] ← $d(leaf_i, leaf_j)^2$;
    matrix[2][i][j] ← $|leaf_i.mass - leaf_j.mass| + 1$
  end
end
matrix[3]←matrix[0] $\odot(\frac{1}{\text{matrix[1]}})\odot\ (\frac{1}{\text{matrix[2]}})$;
return matrix[3]

The $\odot$ symbol denotes the Hadamard product. Notice how the double for loop is an all-pairs comparison among the leaves which requires $\mathcal{O}(n^2)$ time.

Outlier Extraction

An output of a $k \times k$ matrix with scores is returned by the previous step, where each cell c_i indicates how much leaf_i is an outlier when compared to leaf_j and vice versa. Therefore, a random column l forms a vector of scores $(s_{l_1}, \ldots, s_{l_i})$ which shows the outlieness of $leaf_i$ in comparison to all other leaves.

However, this matrix contains redundant information since the goal is to obtain a single measure of outlierness for each leaf, rather than the pairwise scores. As a result the median is extracted from every vector. Using the median as a measure of outlierness is preferred over the average because it avoids being influenced by extreme values. Thus, each leaf is assigned a median score on the real number line, ranging from 0 to infinity. Leaves with median scores positioned more towards the left side of the real line are compared against leaves located on the right side. If the contents of a leaf on the left side do not appear among the leaves on the right side, it can be considered an outlier. The determination of what is considered "left" and "right" is explained in subsequent steps.

This aformentioned outlier extraction process is nicely summarized by Algorithm 4.

In the algorithm, the k vectors are represented by their median, and their generalised syntactic structures are compared iteratively by examining the leaves placed on the median line on the left compared to the rest. The comparison can be done using two different methods: (i) generalised word comparison and (ii) regular expression (regex) comparison.

(i) In the generalised word comparison, the first word in the leaf's data (and by extension its representative) is transformed into its generalised structure. For example, if a node has a representative string of "A" and contains the words "Ankara" and "Athens," the generalised word of this leaf would be "Ullllll." This generalised word is then compared against other generalised strings, such as "Ull" or "Ullllllll."

(ii) In the regex comparison, the word is transformed into a regex equivalent generalised expression using the same process. For example, in the previous example, the string "Ullllll"

Algorithm 4: Outlier Extraction Algorithm

```
Data: A k×k scoring matrix
Result: Reported Outliers
leaf_score_list ← scoring_matrix.get_vectors.get_median;
outlier_list ← [];
for every step in the lower half of steps do
    for leaves in that step do
        possible_outliers ← leaf_score_list.bottom_percentile;
        possible_normal_values ← leaf_score_list.upper_percentile;
        for outlier in possible_outliers do
            if outlier.syntactical_pattern is in
              possible_normal_values.syntactical_patterns() then
                disregard outlier
        end
    end
end
return outlier_list
```

would be transformed into "Ul+" where the plus sign denotes one or more repetitions. This regex expression is then compared to other regexed leaves, such as "d+" or "UlU".

There are cases where one transformation method is more suitable than the other. For example, consider an attribute that contains alpha-3 codes (ISO 3166-1) for countries, except for a single value that contains the country's full name (e.g., "AUSTRALIA" instead of "AUS"). Assume for simplicity that the frequencies of the values are the same. Based on the previous tree and matrix implementation this particular value would have a lower matrix score, while the rest would be somewhat grouped together. As a result when comparing this leaf to its generalised structure it would be noted as an outlier since UUUUUUUUU is not in UUU, but this doesn't apply for the regex case as U+ (AUSTRALIA) is present in U+ (alpha 3-codes). Figure 4.8 shows this example in a visual form.

Another example is an attribute containing names of leading roles in movies, where the strings have varying lengths and syntactic structures. For example, a cell contains the strings "Sandra Bullock, George Clooney" while another contains the string "Brendan Fraser, Sadie Sink". The generalised version of the first would be UllllllwUlllllllswUllllllwUlllllll. In this case, the generalised word comparison may result in many false positives, as the varying structures make it difficult to find matches between the left and the right. The regex transformation can be a valid choice in this case, as most names would follow a pattern like "Ul+sUl+". This allows for a more flexible matching criterion and reduces the number of false positives. True syntactical outliers, such as a value like "ERROR 404," would still be reported as they would have significantly different regex transformations (i.e. *U+wd+*) compared to other values.

Instead of burdening the user with the choice between generalised word comparison and regex comparison, the framework automatically determines the appropriate method based on the average ratio of matches and mismatches between possible outliers and normal values for both generalised and regexed strings. If the average ratio falls between 42% and 98%, the regex syntactical structure is preferred; otherwise, the generalised pattern is used. This approach covers both scenarios mentioned above and provides a balanced and automated decision-making process. The specific lower and upper bounds of 42% and 98% for the average ratio were chosen anecdotally. However, these values have been found to provide excellent representations across many datasets.

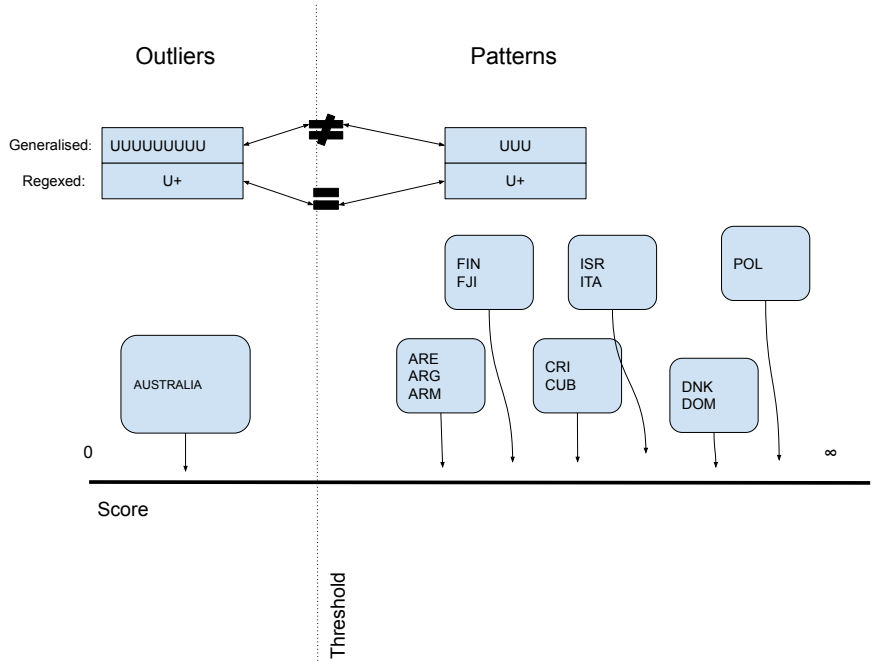


Figure 4.8: Comparison of potential outliers(left) and patterns(right) for a specific threshold value

Finally, it is a valid question to wonder why only these two transformations are considered and not a third transformation that for example may combine elements of both. The chosen transformations offer distinct approaches to generalizing the strings. The first transformation aims to generalize the string while still retaining its original length. This allows for a more specific representation of the string while capturing patterns and similarities among similar values. On the other hand, the second transformation discards the length of the word entirely and uses a more generic representation. This approach focuses more on capturing the overall syntactic structure of the string rather than its specific details.

While there are indeed an infinite number of possible transformations, any other transformation would fall somewhere in between these two approaches in terms of specificity. The choice of these two transformations strikes a balance between preserving specificity and capturing general patterns in the data.

Pattern Extraction

Having done the outlier extraction the pattern extraction, follows a similar thinking. The leaves that are in the upper side of the percentile are semi-generalised as described in preliminaries section 3.1. This semi generalisation is the golden spot that can summarise many patterns without being too specific. It is important to note that this semi-generalization is only applied to the patterns and not to the outliers. If the semi-generalization were applied to the outliers as well, it could lead to a significant number of non-outliers escaping the pattern matching comparison. Algorithm 5 provides a nice overview of the details.

Algorithm 5: Pattern Extraction Algorithm

Data: A k×k scoring matrix
Result: Dominating Patterns
leaf_score_list ← *scoring_matrix.get_vectors.get_median*;
pattern_list ← [];
for *leaves with score in the upper half of steps* **do**
| *pattern_list.add*(*leaf_semi_generalised_presentation*)
end
return *pattern_list*

Discerning Patterns from Outliers

Determining the halfway split for selecting outliers and patterns in the algorithms raises an important question about how to define this split point. There are different approaches that can be considered, each with its own implications.

The most simple option would be to use the 50% percentile as the split point. After all, taking the 50% percentile would mean that 50% of the leaves fall below that, and can be considered as potential outliers. Note that they are not yet true outliers as algorithm 4 shows. For that to happen they need to have a syntactic pattern that is not found in the rest 50% (patterns). However, even in that case, where a check is still applied, the search space for potential outliers extends way beyond to the right.

An alternative approach would involve an uneven split towards the lower end of the line, such as using the 1%, 2%, or 5% percentile as the split point. However, this method has an issue as well. Selecting a specific percentile value becomes arbitrary and lacks general applicability to random datasets, as it heavily depends on the prevalence of outliers within the dataset. For instance, in a dataset with a high percentage of erroneous values, setting a percentile of 5% would mean that either all of the syntactical forms of the outliers are found in the patterns or a large number of outliers are falsly considered as patterns. Thus, it becomes evident that setting a low and arbitrary threshold is not a viable solution.

How about, a conservative but still lenient enough threshold, for example in the range of around 30%? That threshold also encounters some issues. For instance, in Figure 4.9, the leaves colored in yellow are identified by a domain expert as true outliers, while the remaining leaves are considered normal values. It is crucial to note that even a small cluster of normal values positioned below the threshold can result in a significant number of false positives. This occurs because the syntactic pattern of this cluster is exclusively found within the boundaries of the cluster itself, without any corresponding occurrences elsewhere to the right.

The previous examples do not even cover the case in which two or more leaves may have the same score. For instance, consider a dataset that creates 10 leaves as in figure 4.10. In cases where there are ties in the scores of the leaves, selecting a specific threshold value, such as 30%, may not hold any particular significance over other nearby values like 20% or 40%. This is because these thresholds would cover the same number of leaves with equal scores.

Indeed, setting a threshold value for outlier detection can be a challenging task, as there may always be datasets or scenarios where a fixed threshold may not effectively differentiate between outliers and patterns. Recognizing this limitation, it becomes clear that a shift in perspective is necessary to determine the boundary between outliers and patterns.

Instead of solely focusing on the percentile value itself, it is more meaningful to consider the number of leaves that fall within the bottom k-th percentile. These leaves potentially contribute to increasing the number of outliers identified by the framework.

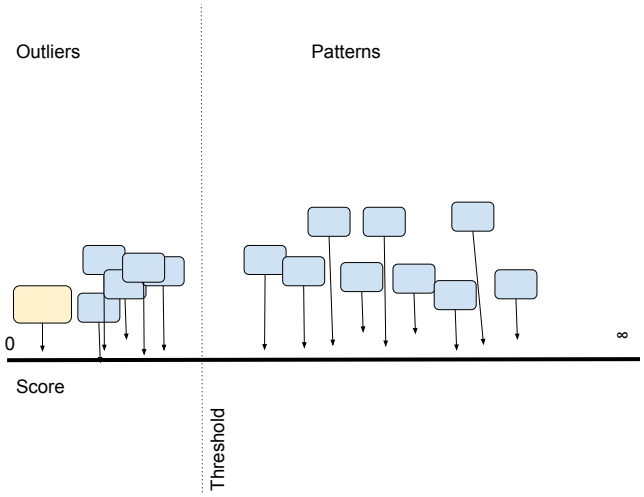


Figure 4.9: A small cluster of normal values may complicate outlier extraction for a fixed threshold

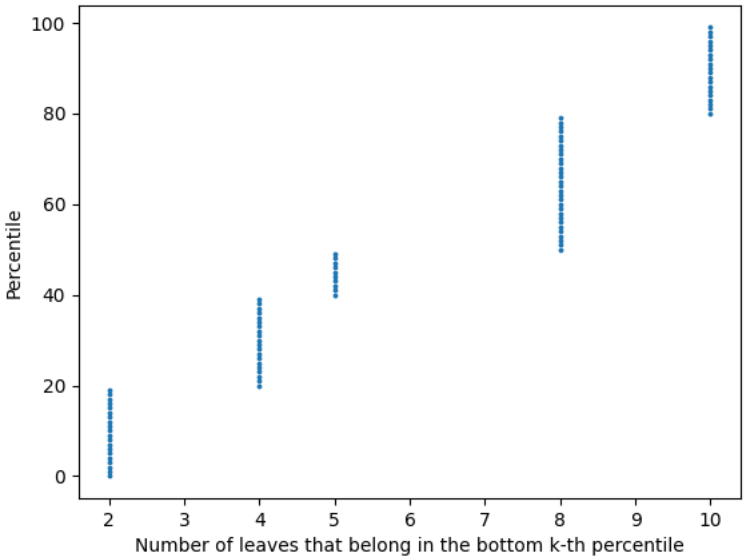


Figure 4.10: Leaves that have the same score create problems when setting a custom fixed threshold

Let the vertical points in the figure 4.10 be coined as a single "step". Each new step increases the number of leaves in the outlier section and thus potentially increasing the number of outliers. Therefore, a logical thing would be to consider half steps as potential outliers and half steps as potential patterns. Figure 4.11 provides a visual representation of this concept, illustrating how the steps can be classified as either outliers or patterns based on their positioning and count.

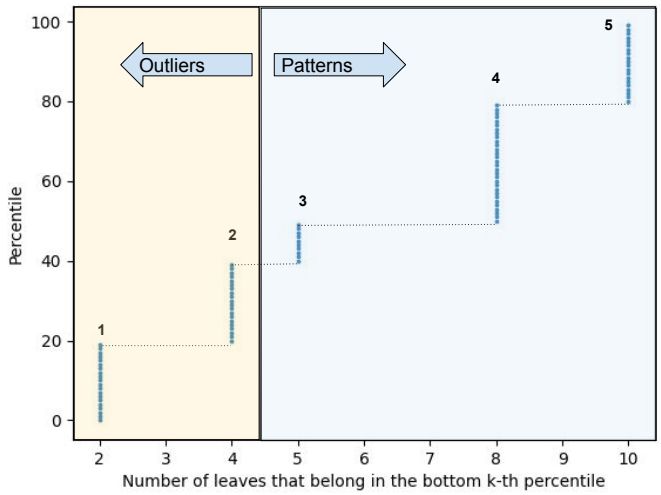


Figure 4.11: Half of the steps are outliers and half of the steps are patterns

In some cases, it may be argued that this approach doesn't solve the issue at hand. For instance, in Figure 4.9, the outlier portion would absorb the cluster of normal values as a whole because it contains fewer steps compared to the median line. While this observation may be valid, it's important to understand how the algorithm operates.

In Algorithm 4, the first loop starts with the lowest step, which in this case is the single yellow leaf. It compares this leaf against all the other patterns and deems it as an outlier. In the subsequent loops, the algorithm considers the entire cluster as an outlier as well. This is why there might be instances where a significant number of false positives are detected, as mentioned in Section 5.

However, it's crucial to note that the algorithm is designed to prioritize outliers in the top positions. By starting with the lowest step and progressively evaluating higher steps, the algorithm aims to capture the most significant outliers. While there may be cases where the cluster of normal values is also considered an outlier, the algorithm's prioritization mechanism ensures that outliers are still given prominence.

Pruning Rules

As indicated in algorithm 2, pruning rules are implemented to effectively control the growth of the tree and avoid unnecessary splits.

There are three pruning parameters in place: (i) Long string pruning/merging, (ii) numeric attribute pruning, and (iii) minimum amount of distinct values pruning.

(i) Long string pruning/merging: In the context of conducting outlier search within relational databases, it is crucial to take into account the specific structure of the database. This consideration becomes particularly relevant when dealing with attributes that store information distributed across multiple fields, rather than being consolidated in a single attribute.

For example, in an employee records database, employee information such as full name and home address may be distributed across several attributes, such as "first_name," "last_name," "street_address," "city," and so on.

If a single attribute contains extremely long values, it can pose challenges both for the data scientist and for the underlying outlier detection framework. From the data scientist's perspective, dealing with long attribute values can be laborious and time-consuming, as it requires extensive data manipulation and extraction to identify outliers accurately. Moreover, from the framework's perspective, accommodating long attribute values can lead to scalability issues, as the tree structure may become too wide or too deep to handle efficiently.

To address this, a threshold is established to halt the tree construction process when such lengthy values or columns are encountered. In this framework, the threshold is set at 36, which corresponds to the length of a Universal Unique Identifier (UUID) consisting of 32 hex digits and 4 dashes. The rationale behind this threshold is that values longer than 36 characters should ideally be replaced with their UUID equivalents by a database engineer, as they consume unnecessary memory without serving a meaningful purpose as a whole. Furthermore, the complete value should be associated with an index using this UUID. However, it is important to note that since this limit is not supported by a formal proof, it can be adjusted by modifying a variable within the framework.

Consequently, when encountering trees with lengthy values, the algorithm distinguishes between two types of nodes based on the length of the values. The first type consists of nodes with values having a length less than 36. In these nodes, the data within each node has its own specific length, and the algorithm proceeds with the previously described algorithms for tree growth and analysis. The second type represents nodes with values of length 36 or greater. Since these values exceed the defined threshold, these nodes are treated as one leaf in the tree which is not further expanded. This process is referred to as *merging* in this thesis.

(ii) Numeric attribute pruning:

If an attribute consists of numeric values that are 95% unique, it is not considered for computation in the first place. The classification of attributes into numeric and non-numeric is based on the string's membership in the class {d} (e.g., value 332) or the class {d, s} (e.g., value 12.88). It is important to note that this classification based on string membership may not cover every numeric attribute or account for all edge cases that involve peculiar values with outliers. However, it serves as a good starting point that effectively handles the majority of cases.

The rationale behind this pruning parameter is that although FIONA performs adequately in detecting numeric outliers (see section 5.1.4), its primary purpose is not specifically focused on numerical outlier detection. There are already various other tools available that are specifically designed for that purpose.

The choice of 95% uniqueness as the threshold is to account for cases where attributes may have a small percentage of random or erroneous values, such as "ERROR" or "NULL," which would render the attribute unclean. Additionally, this threshold helps differentiate between purely numerical attributes and categorical attributes that have been translated into nominal values (e.g., "Yes" mapped to 1).

(iii) Minimum amount of distinct values pruning:

This pruning parameter is also known as *nDistinctMin*. It applies to a specificity level greater than 0 (i.e., 1 and onwards) and stops the growth of a node if the number of distinct values within the node falls below this threshold. The growth of the rest of the tree continues as expected.

Choosing the appropriate nDistinctMin value could be left to the user's discretion. However, since the graphical user interface does not provide information about the underlying tree structure, it may be out of place to require the user to make this decision. Instead, an alternative approach is proposed based on the maximum computational capabilities of the machine running the framework. While the graphical user interface still provides an option to input the nDistinctMin value, it is not mandatory or required for the user to fill in this information.

In this approach, the nDistinctMin value is determined based on the machine's computational capacity. For example, consider the worst-case scenario where a nDistinctMin value of 2 would result in a tree with as many leaves as there are values in the attribute. While each value is considered unique and a potential outlier, it imposes a significant memory overhead. For instance, in the current implementation using numpy, as shown in Algorithm 3, 24 thousand leaves equate to a 24000 $\times$ 24000 matrix, that consumes approximately 15 GB of RAM. Although some machines may be able to accommodate this matrix, a more viable and configuration-free option is to increase the nDistinctMin value and build a new tree that produces a scoring matrix that fits into main memory. By increasing the nDistinctMin value, the resulting tree will have fewer leaves with more distinct values inside each leaf. The choice to increase the nDistinctMin value is anecdotally guided by the Fibonacci sequence, which grows at an acceptable rate for the given problem. The general process is shown by algorithm 6 which is a slight variation of algorithm 1.

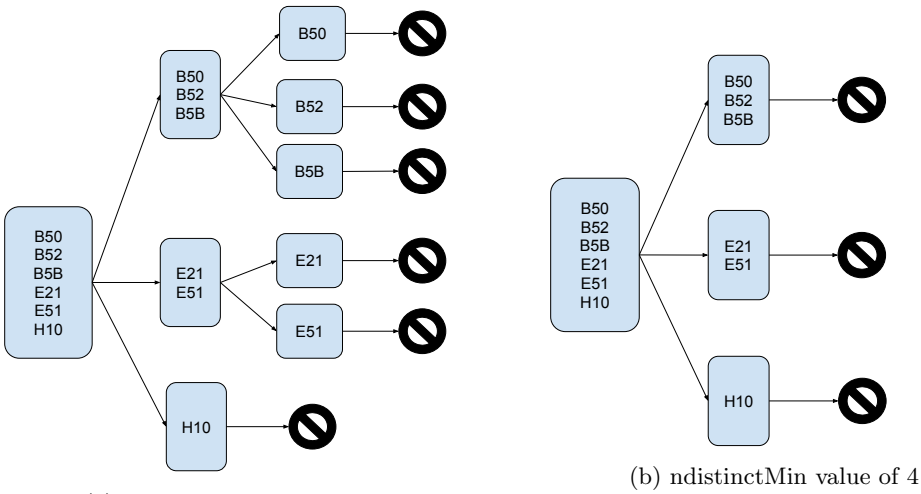
Algorithm 6: Search for the appropriate ndistinctMin value

Data: An attribute of the dataset
Result: A tree whose scoring matrix can fit into main memory
$attribute \leftarrow read\_column$; `/* Loads column into main memory */`
$nDistinctMin \leftarrow next(fibonacci)$;
$tries \leftarrow 0$;
$can\_Fit \leftarrow False$;
repeat
  $tree \leftarrow tree\_grow(attribute, nDistinctMin)$;
  $can\_Fit \leftarrow check\_memory\_fit(leaves)$;
until $tries < 40$ *or* $can\_Fit == True$;

In this algorithm the tree is grown repeatedly until the leaves create a matrix that can fit into the machine's main memory.

Larger nDistinctMin values mean that the tree stops with potentially more distinct values inside each leaf. Figure 4.12 illustrates examples with nDistinctMin values of 2 and 4. Notice how in Figure 4.12a, there are 6 leaves, while in Figure 4.12b, where the nDistinctMin value is 4, there are 4 leaves. This reduction in computational cost comes with the trade-off that the value "B5B" is grouped together with "B50" and "B52".

Note that in both cases, figure 4.12 does not depict the root and the first level of the tree due to space constraints but focuses on a specific node on the third level.



(a) ndistinctMin value of 2

(b) ndistinctMin value of 4

Figure 4.12: Different outcomes for different ndistinctMin values

4.2.3 Output

The output section of the system is responsible for presenting the results of the categorical outlier detection algorithm to the user in a clear and easy-to-understand format. The input to the front end is the list of outliers generated by the algorithm, which is then processed and presented to the user in a user-friendly interface. The system can also store the results in a subfolder for persistency, allowing the user to review the results at a later time or to compare them with the results of future runs of the algorithm.

The output provided to the user in FIONA includes both the patterns and the outliers.

Outliers

The outliers table consists of three columns: (i) System's Decision Making Confidence, (ii) Generic Strings, and (iii) Occurrences.

(i) System's Decision Making Confidence: This column serves as a visual indicator to the end user, representing the level of confidence the framework has in classifying a particular string as an outlier. The confidence level is calculated based on the threshold level at which the outlier is detected. The decision making confidence is presented as a percentage, where a higher value indicates a higher degree of confidence in the outlier classification.

(ii) Generic Strings: The Generic Strings column contains the generalized or regexed representations of the actual values in the dataset. The purpose of this column is to provide a summary view to the scientist, enabling them to quickly assess if a value is classified as an outlier without manually examining each individual value. By using generalized or regexed strings, the column presents a more condensed and standardized representation of the values.

(iii) Occurrences: The Occurrences column presents the actual values in the dataset. The values are displayed in a JSON format, where each key represents a specific value, and the corresponding value associated with the key indicates the frequency or occurrence of that value in the dataset. This information helps the user understand the distribution and prevalence of the outliers within the dataset.

Patterns

After the outliers table, FIONA presents the reported patterns. The patterns table consists of four columns: (i) Generic Patterns, (ii) Minimum Ensured Coverage for Generic Patterns, (iii) Specific Patterns, and (iv) Minimum Ensured Coverage for Specific Patterns.

(i) Generic Patterns: The Generic Patterns column contains either the generalized or regexed transformation of the contents of the leaves that belong to the patterns. These patterns represent the common syntactic structures found within the dataset.

(ii) Minimum Ensured Coverage for Generic Patterns: This column represents the minimum ensured coverage for each generic pattern. It is calculated as the sum of the percentages that belong to that specific generic pattern in relation to the entire dataset. The term "ensured" implies that this coverage is guaranteed, as there may be additional occurrences of the pattern in the outliers section that were not classified as outliers due to their syntactic similarity to the patterns.

(iii) Specific Patterns: The Specific Patterns column presents the individual representations of the leaves without any transformation on their syntactic structures. These patterns provide a more specific and detailed view of the syntactic structures identified in the dataset.

(iv) Minimum Ensured Coverage for Specific Patterns: Similar to the coverage for generic patterns, this column displays the minimum ensured coverage for each specific pattern. It represents the sum of percentages for that specific leaf pattern in relation to the entire dataset. It provides insights into the prevalence of specific syntactic patterns in the dataset.

Chapter 5

Evaluation

The subsequent section delves into the comprehensive evaluation of the framework. The goal of this segment is to demonstrate the effectiveness and efficiency of the proposed algorithm on various datasets. The datasets used in this evaluation have been carefully selected to cover a wide range of scenarios and to demonstrate the capabilities of FIONA on finding outliers. The results obtained will be analyzed, and insights into the strengths and limitations of FIONA will be provided.

5.1 Experimental Setup

The experimental setup serves as a comprehensive platform to evaluate the performance and capabilities of the FIONA framework. This evaluation encompasses a diverse range of datasets, including those with both ground truth and without. In order to benchmark FIONA's performance, several baseline methods have been chosen for comparison. These methods, namely dBoost, FAHES, Raha, and SURAGH, are aligned with FIONA's focus on categorical outlier detection. As an essential performance measure, precision is adopted to assess the accuracy of outlier detection. However, it's imperative to note that while precision is crucial, it should not be the sole criterion for gauging the framework's effectiveness. Overall, the following experimental setup aims to provide a rigorous assessment of FIONA's performance and robustness, shedding light on its strengths and areas for improvement.

5.1.1 Datasets

To evaluate the effectiveness of FIONA, two types of datasets are utilized: those that have a corresponding cleaned version and those that do not. The cleaned datasets are identical to the ones employed in the Raha framework, which are purportedly free from erroneous values. However, upon evaluation, it is revealed that even these cleaned datasets contain outliers. On the other hand, datasets without a cleaned counterpart are anticipated to contain inconsistent observations and are obtained from various open-data websites.

Datasets with ground truth

The Raha framework utilizes a total of 6 datasets, namely *beers*, *flights*, *hospital*, *movies*, *rayyan*, and *tax*. These datasets are extensively described in the accompanying paper of Raha. The paper provides detailed information about these datasets, including their characteristics, attributes, and any specific considerations relevant to outlier detection.

It is important to note that while other datasets may have been used in the Raha framework's paper, they are proprietary and were not available for utilization in the current framework. Therefore, the focus is primarily on the 6 datasets mentioned above, which are openly accessible and can be used for evaluation and comparison within the context of the framework.[1].

Datasets without ground truth

All datasets are fetched from open data websites for reproducibility purposes. Various datasets are chosen to ensure that a wide span of syntactical patterns are covered across several domains. A brief overview of the datasets used can be seen in table 5.1. Appendix section A.1 contains more information about the datasets.

Abbrv Name	Size (MBs)	# Rows	#Attributes	Industry
students	0.5	4,424	36	Education
calendar	112	1,048,575	7	Hospitality
listing	0.1	1,119	9	Alcohol
crime	180	739,687	28	Criminal Justice

Table 5.1: Summary of the datasets without ground truths used

5.1.2 Baseline Methods

As expounded upon in Section 2, significant attention has been given by the academic community to both numerical and categorical outlier detection. This has led to the development of a variety of tools, each possessing its own strengths and weaknesses. From this array of options, a careful selection had to be made. In cases where datasets have ground truth available, three other tools were chosen for comparison alongside FIONA. Those tools are dBoost [73], FAHES [74], Raha [63] and SURAGH [35]. These tools were selected due to their specialization in categorical outlier detection and their differing approaches in computation and outlier detection methodologies. Moreover, they represent the cutting-edge technologies among the multitude of available tools in this domain.

It's important to note that the baseline methods are primarily designed for datasets with existing ground truth, rather than those without. While these tools can still be applied to datasets lacking ground truth, the efficacy of doing so is limited. Without a domain expert on hand or a straightforward means of verifying the results, their applicability in such cases is somewhat constrained.

5.1.3 Performance Measure

Selecting an appropriate performance measure is a nuanced challenge, as it significantly impacts the evaluation of the framework outlined in the previous section. In the absence of ground truth for the majority of datasets, precision emerges as the most meaningful metric to assess the framework's performance. For reference, precision (also known as positive predictive value) is defined as the fraction of relevant instances among the total retrieved instances.

Even in cases where the datasets have been cleaned (datasets with ground truth), there may still be instances of typos or data anomalies that could be considered outliers depending on the circumstances. For instance, table 5.3 shows that the datasets can be very well considered uncleaned. Furthermore, achieving high precision on these specific datasets does not

[1]The Raha configuration system can be found on GitHub at https://github.com/BigDaMa/raha

necessarily extrapolate to high precision across all datasets in diverse contexts. Consequently, while precision stands as the singular metric employed across the datasets (as indicated in Table 5.2), it is approached with discernment, with more emphasis placed on discerning the framework's specific outcomes across the assortment of dataset types used.

5.1.4 Results and Analysis

Table 5.2 shows how FIONA and other tools fare for these datasets that have allegedly cleaned counterparts. Note that these metrics are taken from the original Raha paper. A detailed analysis of the results is provided further down. This section delves deeper into the interpretation and implications of the findings, offering further insights and understanding of the framework's performance.

Precision	**beers**	**flights**	**hospital**	**movies**	**rayyan**
dBoost	0.54	0.78	0.54	0.18	0.00
FAHES	0.73	0.4	0.111	0.00	0.00
Raha	0.99	0.82	0.94	0.85	0.81
SURAGH	-	0.9	-	-	-
FIONA	0.451	0.867	0.71	0.173	N/A (see table 5.11)

Table 5.2: Precision as reported by other tools

In Table 5.2, it is important to note that SURAGH reports indices rather than values. This distinction is crucial when considering the precision metric. In the precision calculation for the SURAGH framework only, an observation is considered a true positive if it contains at least one erroneous value.

The dash "-" indicates that the dataset could not be computed. The limitations are either due to space constraints, or its inability to process non-ascii characters. No metrics are reported for *tax* dataset since it is omitted by the authors of the Raha and was solely used for the scalability section of their work.

The experiments conducted have yielded valuable insights into the capabilities of the FIONA framework. While the precision values in Table 5.2 may not directly showcase its strengths, FIONA has proven to be effective in efficiently detecting outliers across datasets from various fields. However, it is important to acknowledge that FIONA also has its limitations, which will be discussed candidly in subsequent sections. Finally, there is ample scope for future work on the FIONA framework. As a recently developed framework, there are numerous areas where it can be enhanced and refined.

In general, FIONA demonstrates a strong performance in detecting outliers within specific contexts as indicated by table 5.2. However, the metric in this table while informative, has certain limitations as it provides a quantitative assessment based on predefined criteria. While in general metrics offer valuable insights into FIONA's overall performance, they may not capture the intricacies and nuances of outlier detection in every scenario.

For example, table 5.3 displays the outliers fetched by the tax dataset for the first name (coined "f_name") attribute.

Technically, since this is a cleaned dataset that has been verified by a domain expert, the precision of this algorithm for this particular attribute is zero. However, upon closer examination of the occurrences, interesting patterns emerge. While there are generally no strict rules for how names are formulated, except for adhering to naming laws in each country, names that are entirely in uppercase, such as "DAIDA", "ICT", "CPT", or "IEEE", appear fictitious.

System's Decision Making Confidence (%)	Generic Strings	Occurrences
99	UUUUU	{"DAIDA":16}
97.9414	Ullslll Ullllllls	{"Ken'ichi":28,"Sei'ichi":18,"Jun'ichi":14} {"Surveyors'":18}
96.9024	Ulllsll UUU	{"Shin'ya":24} {"ICT":28,"CPT":24}
96.8632	lll	{"eva":19,"van":25}
96.4711	UUUU	{"IEEE":20}
96.4515	Ulsll	{"Ra'ed":22,"Na'el":22}
96.177	Ullslll	{"Irs'hak":17,"Its'hak":24}
94.3343	UsUllll Ullslllll	{"M'Lissa":23} {"Shin'ichi":18,"Shun'ichi":23}
93.2757	Uslllll	{"M'hamed":21}
83.7481	UlllUllll	{"HongJiang":28}
78.063	UllllUll	{"AbdelAli":13,"ChingYen":27}

Table 5.3: Outliers for the cleaned counterpart of the tax dataset for the f_name attribute

Additionally, names that start with a lowercase letter, such as "eva" or names with the Dutch proposition "van" raise doubts about the dataset's actual cleanliness.

Regarding the other outliers, although they are considered false positives in the sense that a name search engine [2] retrieves results confirming their validity as names, the presence of a special character (apostrophe) in their names distinguishes them from the other names and is the reason they were identified as outliers.

Undoubtedly, FIONA excels in identifying outliers in values that are expected to have a specific fixed structure. Table 5.4 illustrates the output obtained from the dirty version of the flights dataset for the "scheduled departure time" (sched_dep_time) attribute.

In the cleaned version of the dataset, the "sched dep time" attribute typically contains values such as "4:29 p.m." or "10:32 a.m." where the time is represented in a 12-hour format followed by "ante meridian" (a.m.) or "post meridian" (p.m.). Values reported as outliers by FIONA clearly deviate from this expected structure. The table reveals a significant variety of different data entry mechanisms that have generated a wide range of syntactic structures.

FIONA's superiority over other systems, such as FAHES, is further exemplified when examining the results obtained from the flights dataset. Table 5.5 illustrates the outcomes produced by FAHES. It becomes apparent that FAHES not only contains errors, such as classifying legitimate values like "CO" or "AA-3-JFK-LAX" as outliers, but it also underreports outliers. For instance, FAHES fails to identify "12:00noon" as an outlier, whereas FIONA accurately detects it.

Due to the implementation of FIONA, it provides valuable insights into the structure of strings within a specific attribute, even without prior knowledge. Tables 5.6 and 5.7 present the results of outliers and patterns, respectively, for the cleaned version of the "county name" attribute ("CountyName") in the hospital dataset.

[2]The search engine used is https://www.behindthename.com

System's Decision Making Confidence (%)	Generic Strings	Occurences
97.8042	Ul+w+d+sd+ d+sd+	{"Thu 20:00":1} {"12:30":1}
81.7485	d+sd+wd+sd+wlsls	{"11/30 11:55 p.m.":1,"11/30 11:25 a.m.":1,"11/30 11:08 p.m.":1,"11/30 12:57 p.m.":1}
81.0036	d+sd+wdsd+wlsls	{"11/30 6:45 p.m.":1,"11/30 7:45 p.m.":1,"11/30 4:16 p.m.":1,"11/30 4:00 p.m.":1,"11/30 8:00 p.m.":1,"11/30 1:55 p.m.":1}
77.3573	d+sd+l+	{"12:00noon":2}
62.027		{"":784}
61.3017	d+sd+lUl+wd	{"10:00aDec 1":1,"10:15aDec 1":1,"10:40aDec 1":3,"10:45aDec 1":1,"10:25aDec 1":1,"10:39aDec 1":2,"12:30aDec 1":2,"11:15aDec 1":1,"11:55aDec 1":1,"11:50aDec 1":1,"11:25aDec 1":3,"11:35aDec 1":1,"11:45aDec 1":2,"11:40aDec 1":2,"11:05aDec 1":2}
58.3807	dsd+lUl+wd	{"8:55aDec 1":1,"8:35aDec 1":4,"8:05aDec 1":1,"8:15aDec 1":1,"8:19aDec 1":1,"8:40aDec 1":1,"8:41aDec 1":2,"7:15aDec 1":2,"7:10aDec 1":6,"7:00aDec 1":2,"7:05aDec 1":2,"7:25aDec 1":2,"7:55aDec 1":1,"7:53aDec 1":1,"7:35aDec 1":2,"7:30aDec 1":2,"6:00aDec 1":6,"6:59aDec 1":2,"6:45aDec 1":2,"6:40aDec 1":1,"6:14aDec 1":1,"9:45aDec 1":2,"9:05aDec 1":2,"9:15aDec 1":1}
56.93	Ul+wUl+	{"Not Available":13}
48.8924	dsd+wlslsws+d+sd+s	{"9:05 a.m. (-00:00)":4}

Table 5.4: Outliers reported for the sched_dep_time attribute of the flights dataset (dirty counterpart)

Table Name	Attribute Name	DMV	Frequency	Detecting Tool
flightsDirty	src	world-flight-tracker	55	SYN
flightsDirty	src	CO	19	SYN
flightsDirty	flight	AA-3-JFK-LAX	25	SYN
flightsDirty	sched_dep_time	Not Available	13	SYN
flightsDirty	sched_dep_time	9:05 a.m. (-00:00)	4	SYN
flightsDirty	act_dep_time	Not Available	18	SYN
flightsDirty	act_dep_time	Contact Airline	14	SYN
flightsDirty	act_dep_time	1:50 p.m. (+8:00)	4	SYN
flightsDirty	sched_arr_time	Not Available	13	SYN

Table 5.5: Results generated by FAHES for the same (dirty) flights dataset

System's Decision Making Confidence (%)	Generalised Strings	Occurrences
87.4925	l+wl+	{"de kalb":25}

Table 5.6: Outliers reported for the CountyName attribute of the hospital dataset (cleaned version)

By examining the provided tables, it becomes evident why "de kalb" is identified as an outlier in the specific attribute. Most of the other strings in the dataset consist of a single lowercase word, while "de kalb" is composed of two words. This deviation from the common syntactic structure of the attribute likely led FIONA to classify it as an outlier.

Interestingly, even though this is the cleaned dataset and such an outlier is reported as a mistake, further investigation reveals a nuanced understanding. After conducting manual searches, it is discovered that the "DeKalb Regional Medical Center," to which this particular row corresponds, does not belong to "De Kalb" county but to "Dekalb" county. The former proper noun refers to a general from the American Revolutionary War and is not associated with the county.

This example highlights the complexity of outlier detection and the importance of considering the semantic context alongside syntactic structures. While FIONA correctly identifies the deviation in the syntactic structure of "de kalb", it is essential to conduct additional research or domain-specific knowledge to determine whether it is a genuine outlier or a valid but less common entity.

While FIONA excels in outlier detection, it becomes even more valuable when compared to other systems that may fall short in providing detailed information about the specific attributes causing issues. For instance, when SURAGH is run on the same cleaned dataset (hospital dataset), it reports 531 ill-formed records without specifying the actual attribute that is problematic. This poses a significant challenge, especially for datasets with a large number of attributes, as scientists would need to manually examine each attribute to identify the outliers.

In contrast, FIONA offers a distinct advantage by allowing scientists to quickly pinpoint whether a value is an outlier or not. Using the previous cleaned dataset as an example, FIONA (potentially falsely) identifies the occurrence of "de kalb" as an outlier. This targeted information enables scientists to focus their attention on the specific attribute and make informed decisions. Meanwhile, SURAGH provides no guarantee that the reported indices correspond to the "de kalb" attribute, as it includes the next and previous 50 indices without explicitly associating them with a specific attribute.

Table 5.8 provides a small example that demonstrates how FIONA performs in outlier detection for a dataset without a corresponding cleaned counterpart, where common sense or domain knowledge is required to verify the outliers.

In this specific example, it confidently identifies the age values of -1, -2, and 0 as outliers with a 98% confidence level. It is worth noting that while other values in the dataset may also appear to be outliers, the lack of background knowledge or domain expertise prevents definitive classification, and they are considered as potential false positives. The patterns extracted for the column, as displayed in Table 5.9, further support the identification of these values as outliers. Note that this is not the complete patterns table due to space limitations. However, determining the precision of such a large table is infeasible.

An intriguing observation is that, even when compared to other outlier detection tools, FIONA stands out in its ability to process and detect outliers in data-intensive contexts using the same

Generic Patterns	Generic Patterns Coverage	Specific Patterns	Specific Patterns Coverage
l+	0.97500	etllll	0.05300
		bellll	0.00200
		bllllll	0.02000
		moblll	0.01300
		mallll	0.05000
		collll	0.05000
		alllllll	0.02000
		mllllllll	0.05000
		fllllllll	0.00200
		tllllllll	0.01800
		cllllllll	0.05000
		jllllllll	0.10500
		lll	0.02500
		dlll	0.02500
		ellllll	0.02500
		sllllll	0.02500
		bullll	0.02500
		morlll	0.02600
		cllllll	0.02500
		hlllllll	0.04200
		fllllll	0.02500
		mllllll	0.02800
		chllllll	0.02500
		cullllll	0.02500
		callllll	0.02800
		rlllllll	0.02500
		challlll	0.02500
		chellllll	0.02500
		crllllll	0.02500
		allllllll	0.01800
		lllllllll	0.02500
		mllllllllll	0.05000

Table 5.7: Patterns reported for the CountyName attribute of the hospital dataset (cleaned version)

laptop as a reference point. The larger size of the dataset poses a challenge for other tools, preventing them from performing the required computations. FIONA's ability to handle such

System's Decision Making Confidence (%)	**Generalised Strings**	**Occurrences**
98.0002	sd	{"-1":44,"-2":10}
60.3803	d	{"0":180308,"2":326,"3":385, "4":403,"5":458,"6":447,"7":471, "8":458,"9":548}

Table 5.8: Outliers reported by FIONA for victim_age column of crime dataset

Generic Patterns	**Minimum Ensured Coverage**	**Specific Patterns**	**Minimum Ensured Coverage**
d+	0.75144	10	0.00085
		11	0.00127
		12	0.00186
		13	0.00253
		14	0.00305
		15	0.00360
		[...]	[...]
		96	0.00009
		97	0.00008
		98	0.00008
		99	0.00037

Table 5.9: Patterns reported by FIONA for victim_age column of crime dataset

data volumes and successfully identify outliers in these scenarios is particularly noteworthy.

In Table 5.10, it is evident that the FIONA framework can also be applied to somewhat numerical data. The specific attribute in focus is the "price" which indicates the cost of a night's stay in a particular accommodation. FIONA identifies prices in the thousands as outliers, which may not necessarily be accurate in the context of the dataset, which pertains to accommodation in Amsterdam. However, without specific context, it is indeed unusual for a person to rent a place that costs 2100 dollars per night, making it a potential outlier. In many cases, such high prices could indicate a digit being misplaced, and a more reasonable price may be in the range of 210 dollars per night. Similarly, an observation with a price of 0 dollars can also be considered an outlier, depending on the appropriate context.

System's Decision Making Confidence (%)	**Generalised Strings**	**Occurrences**
65.5753		"":759
42.6191	sdsd+sd+	"$2,163.00":132, "$2,100.00":1, "$2,106.00":1," $2,650.00":1, "$2,500.00":158, "$2,442.00":213,", [...], $1,485.00":1, "$1,415.00":1, "$1,417.00":1," $1,419.00":2, "$1,413.00":123, "$3,000.00":2," $3,605.00":52, "$6,000.00":2, "$7,900.00":365," $7,077.00":1, "$7,585.00":3
38.7179	sdsd+	"$0.00":1825

Table 5.10: Outliers reported by FIONA for price column of calendar dataset

Table 5.11 shows why cleaning a dataset can be a challenging task. In the case of the Rayyan dataset, it is observed that even without the presence of a domain knowledge expert, the dataset is found to be far from clean. This indicates that the ground truth of the dataset has been significantly altered, making it unreliable for evaluating precision. Given the unreliable ground truth and the lack of a clear benchmark against which precision can be measured, it is inappropriate to report FIONA's precision in this particular context. As a result, the precision value in table 5.2 is left blank, reflecting the limitations in assessing precision in the presence of unreliable ground truth.

For instance, upon examining the clean dataset, it becomes apparent that it contains numerous empty cells for important attributes such as journal title. Additionally, the attribute "journal ISSN numbers" includes values like "01/01/1968," which are clearly incorrect. For reference, both of the empty cells and the peculiar ISSN numbers are reported by FIONA. This highlights the significance of critically examining the claimed precision or accuracy of datasets, particularly when the dataset itself contains obvious errors. FIONA's ability to identify such inconsistencies and outliers can contribute to improving the overall quality and reliability of outlier detection in datasets.

System's Decision Making Confidence (%)	**Generalised Strings**	**Occurrences**
99		"":316
97.5297	d+sd+UwsUl+swd+sd+lUl+sd+ d+sd+wsl+s d+sd+wd+sd+wsl+s d+sd+w+d+sd+	"0301-620X (Print) 0301-620x":1 "Feb-14":2,"Feb-62":1,"Feb-65":3,"Feb-37":1,"Nov-53":1,"Apr-74":1,"Sep-60":1,"Sep-22":3,"Mar-67":2,"Mar-22":2,"Mar-09":1,"Mar-17":1,"Jan-55":1,"Jan-68":1,"Jan-47":1,"Jan-15":1,"Jan-14":1 "1478-7547 (electronic)":1 "1840-0132 1840-2445 (electronic)":1 "1612-4782 1612-4790":1

Table 5.11: Fragment of the Rayyan Outliers reported by FIONA

Finally, the validation of FIONA on real data, particularly in collaboration with professionals from the University Medical Centre in Utrecht at the Department of Psychiatry, provides valuable insights and further supports the claims made about its effectiveness. Overall, they have expressed positive remarks about FIONA. They have found that the tool effectively identifies outliers, which confirms the tool's validity. However, they have also noted some areas for improvement, which align with the limitations and future work discussed later in this thesis.

Due to privacy restrictions, no specific details about the nature of the data or the specific applications within the Department of Psychiatry can be provided. However, the collaboration with professionals in a real-world setting adds credibility to FIONA's performance and highlights its practical relevance in diverse domains beyond the academic research environment.

Chapter 6

Conclusion

The significance of identifying outliers in databases cannot be overstated. Many datasets contain erroneous values that significantly impact data quality. Detecting such values accurately has been a challenging problem in the field of outlier detection, and numerous approaches have been attempted to address it.

However, previous attempts often overlooked the distinct syntactic structure exhibited by outliers in comparison to normal values. Additionally, many existing tools required intricate tuning of complex hyperparameters, making them challenging to use effectively.

To address these limitations, the FIONA framework is developed. It is a configuration-free system that takes into account the syntactic structure of strings in a dataset, among other factors, enabling the effective detection of erroneous values. FIONA combines a tree structure with a scoring function that performs pairwise comparisons among all the strings describing the elements in the dataset, facilitating the identification of outliers.

The experiments conducted to validate FIONA's performance demonstrate its effectiveness, even in detecting outliers in allegedly cleaned datasets. Furthermore, it boasts a user-friendly interface, making it highly accessible and appealing to end-users and scientists alike.

Overall, FIONA offers a valuable contribution to the field of outlier detection in categorical data and holds promise for improving data quality in various applications.

Advantages of FIONA

FIONA offers several key advantages that contribute to its effectiveness as an outlier detection framework. First, it excels at capturing syntactic patterns within attributes. Its design is specifically geared towards this goal, allowing it to identify outliers based on distinct syntactic structures. This is particularly useful for attributes that contain codes, such as language codes or country codes, where outliers can be easily identified without the need for domain expertise.

Furthermore, it is deterministic, ensuring consistent results across different runs. This deterministic nature allows for reproducibility and enables researchers to rely on the stability of the framework's output.

Another advantage of FIONA is its unsupervised nature. It does not require domain knowledge or annotated data. This eliminates the need for labeled data or manual annotation, making it applicable to a wide range of datasets without the requirement of prior knowledge.

Additionally, this novel framework is configuration-free, which simplifies its usage for scientists and researchers. There is no need for extensive parameter tuning or manual adjustments based on the underlying structure of FIONA. This ease of use allows scientists to quickly apply it to their datasets without diving into its internal workings.

In addition to the previous findings, the experiments demonstrate that it performs efficiently in a big data context. Without relying on cloud computing, the laptop used for its development successfully processed a dataset containing over 700 thousand lines of code with 28 attributes. This highlights the scalability of FIONA and its ability to handle large datasets on a standard machine. The full specifications of the machine used for the experiments can be found in the Appendix, specifically in Section A.2. The successful processing of such a large dataset on a regular laptop emphasizes FIONA's capability to handle substantial volumes of data without requiring specialized infrastructure. This is advantageous for researchers and practitioners who may not have access to high-performance computing resources but still need to analyze and detect outliers in large-scale datasets.

Finally, one of the notable advantages of the framework is its agnostic approach towards words, allowing it to handle different encodings and languages seamlessly. This flexibility ensures that the framework can effectively process datasets containing various encoding schemes without any modifications to the codebase. For instance, FIONA can successfully compute databases that utilize non-ASCII characters, such as the 4-Byte UTF-8 Unicode Encoding (*utf8mb4*), where each character belongs to a special class. This capability sets it apart from other frameworks like SURAGH, which face challenges when dealing with non-ASCII characters, as mentioned in its paper.

In summary, FIONA's advantages lie in its ability to capture syntactic patterns, deterministic behavior, unsupervised nature, configuration-free design, and efficiency in handling big data. These features make FIONA a powerful and accessible tool for outlier detection, empowering researchers and practitioners to analyze and detect outliers effectively across various datasets.

FIONA's Limitations

Indeed, FIONA faces a significant challenge in capturing the semantic structure of data. This limitation becomes apparent when working with datasets that contain attributes related to proper nouns, such as names or postal addresses. The complexity of these attributes, with their inclusion of special characters, white spaces, and a wide range of uppercase and lowercase letters, makes it difficult to impose syntactic constraints or establish a standard pattern. As a result, FIONA struggles to effectively detect outliers in such cases. This issue is highlighted by the low precision observed in Table 6.1.

The difficulty lies in the fact that identifying outliers in attributes like "name" where anything can be permitted heavily relies on their semantic context rather than their syntactic structure. To address this challenge, it becomes necessary to incorporate semantic analysis into the outlier detection process. However, this is currently beyond the scope of FIONA's capabilities, which primarily focus on syntactic patterns.

In addition to the aforementioned limitations, FIONA encounters some memory issues when handling large datasets with millions of records. Despite efforts to optimize memory usage, FIONA remains resource-intensive and may struggle to process such extensive data. However, it is worth mentioning that special attention has been given to ensuring that FIONA can still provide results, as demonstrated in Algorithm 6. Although memory limitations exist and will always exist, FIONA strives to fetch and deliver outcomes within its capacity.

Furthermore, it is important to note that FIONA's behavior may vary depending on the machine it is executed on. While FIONA is deterministic and produces consistent results across runs on the same machine, smaller machines may yield different outcomes due to their

System's Decision Making Confidence (%)	Generalised Strings	Occurrences
99	UUdswUUU	"RC1, LLC":1,"CJ2, INC":1
	UdUdswUUU	"O2A2, LLC":1
	UUdUswUUU	"JS2Y, INC":1
	dddddUwUUUs	"19725A INC.":1
	UUwdddswUUU	"MY 523, LLC":1
	UUdUUUUswUUU	"FS1SSMD, LLC":1
	ddddUUUswUUU	"8455CSC, LLC":1
	$(\cdots)$	$(\cdots)$
	UUUU	"NONE":3
	UUUUUUUUUU	"GREENECORP":1,"RIDGEWELLS":1
	UUUUUUUUUUUUU	"FINGERPRINTING":1

Table 6.1: Fragment of reported outliers for Account Name column of listing dataset

limited computational resources. However, it is worth emphasizing that this limitation is relatively minor and is only mentioned for transparency. Throughout the development and testing of FIONA, instances of out-of-memory errors were infrequent, as efforts were made to ensure compatibility with various machine configurations.

Lastly, FIONA exhibits a worst-case computation time complexity of $\mathcal{O}(n^2)$ as indicated by algorithm 3, which can be a limiting factor for large datasets. However, it is crucial to note that this worst-case scenario is highly improbable and would require a deliberately crafted dataset by someone with extensive domain knowledge. In practice, FIONA's computation time is typically manageable and performs efficiently for datasets encountered in real-world scenarios.

Future Work

FIONA is a solo project, where certain liberties were taken with the code, especially as time constraints became a factor. The frontend implementation, for instance, lacks consideration for best UI/UX practices and primarily reflects the author's limited experience with Typescript. Furthermore, there are opportunities for enhancing the backend codebase, specifically within the Python language itself. Some faster alternatives may have been overlooked due to the project's solo nature.

An important avenue for future work involves integrating the semantic meaning of words into the framework to detect not only syntactic outliers but also semantic outliers. This represents a significant undertaking that has the potential to greatly enhance the application and distinguish it from others in the field.

Lastly, improvements can be explored in the regex representation of outliers and patterns. While the current patterns yield favorable results, it remains uncertain if they are the optimal representation. Further research and experimentation are required in this area to achieve potential advancements.

Bibliography

[1] Charu C Aggarwal et al. *Data mining: the textbook*. Volume 1. Springer, 2015.

[2] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases". In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. 1993, pages 207–216.

[3] Rakesh Agrawal, Ramakrishnan Srikant, et al. "Fast algorithms for mining association rules". In: *Proc. 20th int. conf. very large data bases, VLDB*. Volume 1215. Santiago, Chile. 1994, pages 487–499.

[4] Michael R Anderberg. *Cluster analysis for applications: probability and mathematical statistics: a series of monographs and textbooks*. Volume 19. Academic press, 2014.

[5] Fabrizio Angiulli and Clara Pizzuti. "Fast outlier detection in high dimensional spaces". In: *Principles of Data Mining and Knowledge Discovery: 6th European Conference, PKDD 2002 Helsinki, Finland, August 19–23, 2002 Proceedings 6*. Springer. 2002, pages 15–27.

[6] Animal and Plant Health Agency. *Accident and Incident Data - 2015-2016*. 2016. URL: `https://www.data.gov.uk/dataset/dc4e6d38-2d35-4859-a71f-989efae28585/accident-and-incident-data-2015-2016` (visited on 01/12/2023).

[7] Stephen D Bay and Mark Schwabacher. "Mining distance-based outliers in near linear time with randomization and a simple pruning rule". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pages 29–38.

[8] Richard J Beckman and R Dennis Cook. "Outlier..... s". In: *Technometrics* 25.2 (1983), pages 119–149.

[9] Gill Bejerano and Golan Yona. "Modeling protein families using probabilistic suffix trees". In: *Proceedings of the third annual international conference on Computational molecular biology*. 1999, pages 15–24.

[10] Gill Bejerano and Golan Yona. "Variations on probabilistic suffix trees: statistical modeling and prediction of protein families". In: *Bioinformatics* 17.1 (2001), pages 23–43.

[11] Daniel Bernoulli and CG Allen. "The most probable choice between several discrepant observations and the formation therefrom of the most likely induction". In: *Biometrika* 48.1-2 (1961), pages 3–18.

[12] Kanishka Bhaduri, Bryan L Matthews, and Chris R Giannella. "Algorithms for speeding up distance-based outlier detection". In: *Proceedings of the 17th ACM*

SIGKDD international conference on Knowledge Discovery and Data Mining. 2011, pages 859–867.

[13] Théo Bouganim, Helena Galhardas, and Ioana Manolescu. "Efficiently Identifying Disguised Missing Values in Heterogeneous, Text-Rich Data". In: *Transactions on Large-Scale Data-and Knowledge-Centered Systems LI: Special Issue on Data Management-Principles, Technologies and Applications*. Springer, 2022, pages 97–118.

[14] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pages 93–104.

[15] Toon Calders and Bart Goethals. "Non-derivable itemset mining". In: *Data Mining and Knowledge Discovery* 14 (2007), pages 171–206.

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

[17] Brian Caswell and Jay Beale. *Snort 2.1 intrusion detection*. Elsevier, 2004.

[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pages 1–58.

[19] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Outlier detection: A survey". In: *ACM Computing Surveys* 14 (2007), page 15.

[20] Yen-Liang Chen, Kwei Tang, Ren-Jie Shen, and Ya-Han Hu. "Market basket analysis in a multiple store environment". In: *Decision support systems* 40.2 (2005), pages 339–354.

[21] Noam Chomsky. "Syntactic structures". In: *Syntactic Structures*. De Gruyter Mouton, 2009.

[22] A Christy, G Meera Gandhi, and S Vaithyasubramanian. "Cluster based outlier detection algorithm for healthcare data". In: *Procedia Computer Science* 50 (2015), pages 209–215.

[23] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6 (June 1970), pages 377–387. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: https://doi.org/10.1145/362384.362685.

[24] International Telecommunication Union (ITU) World Telecommunication/ICT Indicators Database. *Individuals using the Internet (% of population)*. 2022. URL: https://data.worldbank.org/indicator/IT.NET.USER.ZS (visited on 01/12/2023).

[25] François Denis. "PAC learning from positive statistical queries". In: *Algorithmic Learning Theory: 9th International Conference, ALT'98 Otzenhausen, Germany, October 8–10, 1998 Proceedings 9*. Springer. 1998, pages 112–126.

[26] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[27] Francis Ysidro Edgeworth. "Xli. on discordant observations". In: *The london, edinburgh, and dublin philosophical magazine and journal of science* 23.143 (1887), pages 364–375.

[28] DB-Engines. *Ranking scores per category in percent, January 2023*. 2023. URL: https://db-engines.com/en/ranking_categories (visited on 01/12/2023).

[29] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd*. Volume 96. 34. 1996, pages 226–231.

[30] William J Frawley, Gregory Piatetsky-Shapiro, and Christopher J Matheus. "Knowledge discovery in databases: An overview". In: *AI magazine* 13.3 (1992), pages 57–57.

[31] Mélissa Gaillard. *CERN Data Centre passes the 200-petabyte milestone*. 2017. URL: `https://home.cern/news/news/computing/cern-data-centre-passes-200-petabyte-milestone` (visited on 01/12/2023).

[32] Amol Ghoting, Srinivasan Parthasarathy, and Matthew Eric Otey. "Fast mining of distance-based outliers in high-dimensional datasets". In: *Data Mining and Knowledge Discovery* 16 (2008), pages 349–364.

[33] E Mark Gold. "Complexity of automaton identification from given data". In: *Information and control* 37.3 (1978), pages 302–320.

[34] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. "ROCK: A robust clustering algorithm for categorical attributes". In: *Information systems* 25.5 (2000), pages 345–366.

[35] Mazhar Hameed, Gerardo Vitagliano, Lan Jiang, and Felix Naumann. "SURAGH: Syntactic Pattern Matching to Identify Ill-Formed Records." In: *EDBT*. 2022, pages 2–143.

[36] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. "Frequent pattern mining: current status and future directions". In: *Data mining and knowledge discovery* 15.1 (2007), pages 55–86.

[37] Jiawei Han, Yongjian Fu, Wei Wang, Krzysztof Koperski, Osmar Zaiane, et al. "DMQL: A data mining query language for relational databases". In: *Proc. 1996 SiGMOD*. Volume 96. Citeseer. 1996, pages 27–34.

[38] Jiawei Han, Micheline Kamber, and Jian Pei. "Data mining concepts and techniques third edition". In: *University of Illinois at Urbana-Champaign Micheline Kamber Jian Pei Simon Fraser University* (2012).

[39] Douglas M Hawkins. *Identification of outliers*. Volume 11. Springer, 1980.

[40] Zengyou He, Xiaofei Xu, and Shengchun Deng. "Discovering cluster-based local outliers". In: *Pattern recognition letters* 24.9-10 (2003), pages 1641–1650.

[41] Zengyou He, Xiaofei Xu, Zhexue Joshua Huang, and Shengchun Deng. "FP-outlier: Frequent pattern based outlier detection". In: *Computer Science and Information Systems* 2.1 (2005), pages 103–118.

[42] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. "Introduction to automata theory, languages, and computation". In: *Acm Sigact News* 32.1 (2001), pages 60–65.

[43] Ming Hua and Jian Pei. "Cleaning disguised missing data: a heuristic approach". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2007, pages 950–958.

[44] Zhexue Huang. "A fast clustering algorithm to cluster very large categorical data sets in data mining." In: *Dmkd* 3.8 (1997), pages 34–39.

[45] Zhexue Huang. "Clustering large data sets with mixed numeric and categorical values". In: *Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(PAKDD)*. Citeseer. 1997, pages 21–34.

[46] Andrew Ilyas, Joana MF da Trindade, Raul Castro Fernandez, and Samuel Madden. "Extracting syntactical patterns from databases". In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE. 2018, pages 41–52.

[47] Ihab F Ilyas and Xu Chu. *Data cleaning*. Morgan & Claypool, 2019.

[48] Harriet Jarlett. *Breaking data records bit by bit*. 2017. URL: https://home.cern/news/news/computing/breaking-data-records-bit-bit (visited on 01/12/2023).

[49] Olaf Kampers, Abdulhakim Qahtan, Swati Mathur, and Yannis Velegrakis. "Manipulation detection in cryptocurrency markets: an anomaly and change detection based approach". In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. 2022, pages 326–329.

[50] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. "Distance-based outliers: algorithms and applications". In: *The VLDB Journal* 8.3 (2000), pages 237–253.

[51] Anna Koufakou, Michael Georgiopoulos, and George Anagnostopoulos. "Detecting outliers in high-dimensional datasets with mixed attributes". In: (2008).

[52] Anna Koufakou, Enrique G Ortiz, Michael Georgiopoulos, Georgios C Anagnostopoulos, and Kenneth M Reynolds. "A scalable and efficient outlier detection strategy for categorical data". In: *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*. Volume 2. IEEE. 2007, pages 210–217.

[53] Anna Koufakou, Jimmy Secretan, and Michael Georgiopoulos. "Non-derivable itemsets for fast outlier detection in large high-dimensional categorical data". In: *Knowledge and information systems* 29 (2011), pages 697–725.

[54] Anna Koufakou, Jimmy Secretan, John Reeder, Kelvin Cardona, and Michael Georgiopoulos. "Fast parallel outlier detection for categorical datasets using mapreduce". In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE. 2008, pages 3298–3304.

[55] Khaled Labib and Rao Vemuri. "NSOM: A real-time network-based intrusion detection system using self-organizing maps". In: *Networks and Security* 21.1 (2002).

[56] Avinash Lakshman and Prashant Malik. "Cassandra: A Structured Storage System on a P2P Network". In: *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures*. SPAA '09. Calgary, AB, Canada: Association for Computing Machinery, 2009, page 47. ISBN: 9781605586069. DOI: 10.1145/1583991.1584009. URL: https://doi.org/10.1145/1583991.1584009.

[57] Godfrey N Lance and William T Williams. "Mixed-Data Classificatory Programs I - Agglomerative Systems". In: *Australian Computer Journal* 1.1 (1967), pages 15–20.

[58] Shuxin Li, Robert Lee, and Sheau-Dong Lang. "Mining distance-based outliers from categorical data". In: *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*. IEEE. 2007, pages 225–230.

[59] Jessica Lin, Eamonn Keogh, Ada Fu, and Helga Van Herle. "Approximations to magic: Finding unusual medical time series". In: *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*. IEEE. 2005, pages 329–334.

[60] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*. Volume 793. John Wiley & Sons, 2019.

[61] J MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proc. 5th Berkeley Symposium on Math., Stat., and Prob.* 1965, page 281.

[62] Mohammad Mahdavi and Ziawasch Abedjan. “Baran: Effective error correction via a unified context representation and transfer learning”. In: *Proceedings of the VLDB Endowment* 13.12 (2020), pages 1948–1961.

[63] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. “Raha: A configuration-free error detection system”. In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, pages 865–882.

[64] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. “Efficient Algorithms for Discovering Association Rules”. In: *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*. AAAIWS’94. Seattle, WA: AAAI Press, 1994, pages 181–192.

[65] Mónica V Martins, Daniel Tolledo, Jorge Machado, Luués MT Baptista, and Valentim Realinho. “Early Prediction of student’s Performance in Higher Education: A Case Study”. In: *Trends and Applications in Information Systems and Technologies: Volume 1 9*. Springer. 2021, pages 166–175.

[66] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. “Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond”. In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pages 1303–1316.

[67] MongoDB. *1.0 GA Released*. 2009. URL: `https://www.mongodb.com/blog/post/10-ga-released` (visited on 01/12/2023).

[68] Eric WT Ngai, Yong Hu, Yiu Hing Wong, Yijun Chen, and Xin Sun. “The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature”. In: *Decision support systems* 50.3 (2011), pages 559–569.

[69] Caleb C Noble and Diane J Cook. “Graph-based anomaly detection”. In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pages 631–636.

[70] Matthew Eric Otey, Amol Ghoting, and Srinivasan Parthasarathy. “Fast distributed outlier detection in mixed-attribute data sets”. In: *Data mining and knowledge discovery* 12 (2006), pages 203–228.

[71] Jong Soo Park, Ming-Syan Chen, and Philip S Yu. “An effective hash-based algorithm for mining association rules”. In: *Acm sigmod record* 24.2 (1995), pages 175–186.

[72] Ronald K Pearson. “The problem of disguised missing data”. In: *Acm Sigkdd Explorations Newsletter* 8.1 (2006), pages 83–92.

[73] Clement Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam Madden. “Outlier detection in heterogeneous datasets using automatic tuple expansion”. In: (2016).

[74] Abdulhakim A Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. “FAHES: A robust disguised missing values detector”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pages 2100–2109.

[75] Henri Ralambondrainy. “A conceptual version of the k-means algorithm”. In: *Pattern Recognition Letters* 16.11 (1995), pages 1147–1157.

[76] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. “Efficient algorithms for mining outliers from large data sets”. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pages 427–438.

[77] Dana Ron, Yoram Singer, and Naftali Tishby. "The power of amnesia: Learning probabilistic automata with variable memory length". In: *Machine learning* 25 (1996), pages 117–149.

[78] Salvatore Sanfilippo. *LMDB – First version of Redis written in Tcl*. 2009. URL: `https://gist.github.com/antirez/6ca04dd191bdb82aad9fb241013e88a8` (visited on 01/12/2023).

[79] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. "Integrating association rule mining with relational database systems: Alternatives and implications". In: *ACM SIGMOD Record* 27.2 (1998), pages 343–354.

[80] Arif Sari et al. "A review of anomaly detection systems in cloud networks and survey of cloud security measures in cloud storage applications". In: *Journal of Information Security* 6.02 (2015), page 142.

[81] Ashok Savasere, Edward Robert Omiecinski, and Shamkant B Navathe. *An efficient algorithm for mining association rules in large databases*. Technical report. Georgia Institute of Technology, 1995.

[82] Mei-Ling Shyu, Kanoksri Sarinnapakorn, Indika Kuruppu-Appuhamilage, Shu-Ching Chen, LiWu Chang, and Thomas Goldring. "Handling nominal features in anomaly intrusion detection problems". In: *15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)*. IEEE. 2005, pages 55–62.

[83] Karanjit Singh and Shuchita Upadhyaya. "Outlier detection: applications and techniques". In: *International Journal of Computer Science Issues (IJCSI)* 9.1 (2012), page 307.

[84] Ramakrishnan Srikant and Rakesh Agrawal. *Mining generalized association rules*. 1995.

[85] Stuart Staniford, James A Hoagland, and Joseph M McAlerney. "Practical automated detection of stealthy portscans". In: *Journal of Computer Security* 10.1-2 (2002), pages 105–136.

[86] Statista. *Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030*. 2022. URL: `https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/` (visited on 01/12/2023).

[87] Pei Sun, Sanjay Chawla, and Bavani Arunasalam. "Mining for outliers in sequential databases". In: *Proceedings of the 2006 SIAM international conference on data mining*. SIAM. 2006, pages 94–105.

[88] NNR Ranga Suri, M Narasimha Murty, and Gopalasamy Athithan. "An algorithm for mining outliers in categorical data through ranking". In: *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*. IEEE. 2012, pages 247–252.

[89] Ayman Taha and Ali S Hadi. "A general approach for automating outliers identification in categorical data". In: *2013 ACS International Conference on Computer Systems and Applications (AICCSA)*. IEEE. 2013, pages 1–8.

[90] Ayman Taha and Osman M Hegazy. "A proposed outliers identification algorithm for categorical data sets". In: *2010 the 7th international conference on informatics and systems (INFOS)*. IEEE. 2010, pages 1–5.

[91] Swee Chuan Tan, Si Hao Yip, and Ashfaqur Rahman. "One pass outlier detection for streaming categorical data". In: *The 3rd International Workshop on Intelligent Data Analysis and Management*. Springer. 2013, pages 35–42.

[92] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, and David W Cheung. "Enhancing effectiveness of outlier detections for low density patterns". In: *Advances in Knowledge Discovery and Data Mining: 6th Pacific-Asia Conference, PAKDD 2002 Taipei, Taiwan, May 6–8, 2002 Proceedings 6*. Springer. 2002, pages 535–548.

[93] David MJ Tax and Robert PW Duin. "Support vector data description". In: *Machine learning* 54 (2004), pages 45–66.

[94] Leslie G Valiant. "A theory of the learnable". In: *Communications of the ACM* 27.11 (1984), pages 1134–1142.

[95] Ke Yan, Xiaoming You, Xiaobo Ji, Guangqiang Yin, and Fan Yang. "A hybrid outlier detection method for health care big data". In: *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*. IEEE. 2016, pages 157–162.

[96] Jeffrey Xu Yu, Weining Qian, Hongjun Lu, and Aoying Zhou. "Finding centric local outliers in categorical/numerical spaces". In: *Knowledge and Information Systems* 9 (2006), pages 309–338.

[97] Krzysztof Zaraska. "Prelude IDS: current state and development perspectives". In: *URL http://www. prelude-ids. org/download/misc/pingwinaria/2003/paper. pdf* (2003).

[98] Bangzuo Zhang and Wanli Zuo. "Learning from positive and unlabeled examples: A survey". In: *2008 International Symposiums on Information Processing*. IEEE. 2008, pages 650–654.

Appendix A

Additional Information

A.1 Datasets used

A.1.1 Students

The dataset used in this study is derived from a higher education institution and encompasses students enrolled in various undergraduate degree programs. The dataset comprises information collected from multiple distinct databases and includes details about students' academic paths, demographics, socioeconomic factors, and their academic performance at the end of the first and second semesters.

This dataset is fetched from UCI Machine Learning Repository. It is based on the work of [65]. More information about the dataset can be found at https://doi.org/10.24432/C5MC89.

A.1.2 Calendar

Calendar contains listings fetched from the Airbnb website for the city of Amsterdam, for the quarter of December - March. This data is fetched from the Inside Airbnb dataset repository. The Inside Airbnb project collects and provides publicly available data about Airbnb listings in various cities around the world. The dataset repository includes information such as listing details, host information, availability, pricing, and reviews. More information about Inside Airbnb can be found at http://insideairbnb.com/about/

Table A.1 provides a small sample of the contents of this dataset.

listing id	date	available	price	adjusted price	minimum nights	maximum nights
2818	09/03/2023	f	$69.00	$69.00	3	1125
2818	10/03/2023	f	$69.00	$69.00	3	1125
2818	11/03/2023	f	$69.00	$69.00	3	1125

Table A.1: Fragment of the calendar dataset

A.1.3 Listing

Companies licensed by Montgomery County allowed sell alcoholic beverages. The dataset contains the name, address, and account number of the licensee. It is updated monthly.

Table A.2 shows a fragment of the listing dataset.

Licensee Name	Street	City	State	Zip	Licensee Number
TWIN VALLEY DISTILLERS	1029 E. GUDE DR.	ROCKVILLE	MD	20850	OCP123261
FARMSOOK HAPPINESS THAI KITCHEN	800 KING FARM BLVD., #125	ROCKVILLE	MD	20850	BBWLHR100259
MATCHBOX-BETHESDA	7278 WOOD-MONT AVE.	BETHESDA	MD	20814	BBWLHR95262

Table A.2: Fragment of the listing dataset

The dataset can be found at https://catalog.data.gov/dataset/department-of-alcohol-beverage-services-abs-licensee-data

A.1.4 Crime

Incidents of crime in the City of Los Angeles from 2020 to present. As stated in data.gov website the data is transcribed from original crime reports that are typed on paper and is prone to errors making it a perfect candidate for testing FIONA. Table A.3 shows few attributes and few rows of this particular dataset.

DR_NO	Date Rptd	TIME OCC	AREA	AREA NAME	Rpt Dist No
10304468	01/08/2020 00:00:00	2230	3	Southwest	377
1.9E+08	01/02/2020 00:00:00	330	1	Central	163
2E+08	04/14/2020 12:00:00 AM	1200	1	Central	155

Table A.3: Fragment of the crime dataset

More information can be found at https://catalog.data.gov/dataset/crime-data-from-2020-to-present.

A.2 Specifications of machine

The machine used to develop this framework has the following specifications:

- **Operating System**: Microsoft Windows 10 Home
- **OS Version**: 10.0.19045 N/A Build 19045
- **Processor**: Intel64 Family 6 Model 140 Stepping 1 GenuineIntel 2419 Mhz
- **Physical Memory**: 16,183 MB
- **Virtual Memory (Max Size)**: 24,382 MB