UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Detecting outliers and annotating their types with indexing structures

Guilherme Domingos Faria Silva

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

Guilherme Domingos Faria Silva

Detecting outliers and annotating their types with indexing structures

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master of Science. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Robson Leonardo Ferreira Cordeiro

USP – São Carlos
April 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

S586d

Silva, Guilherme Domingos Faria
Detecting outliers and annotating their types with indexing structures / Guilherme Domingos Faria Silva; orientador Robson Leonardo Ferreira Cordeiro. -- São Carlos, 2022.
99 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2022.

1. Detecção de anomalias. 2. Anotação de anomalias. 3. Estruturas de Indexação. 4. Slim-tree. I. Cordeiro, Robson Leonardo Ferreira, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Guilherme Domingos Faria Silva

Detectando anomalias e anotando seus tipos com estruturas de indexação

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Robson Leonardo Ferreira Cordeiro

USP – São Carlos
Abril de 2022

Dedicated to my family, for their constant support, availability and for instilling me to push forward with their hard work. To my friends, who never failed to enlighten me with good mood or remind me of what life really is about. To those who see anomalies as opportunities to raise the ordinary.

ACKNOWLEDGEMENTS

Thanks to my advisor, Robson Cordeiro, for his trust, constant availability, helpful criticism and his extensive guidance during the development of this MSc work.

Thanks to Leman Akoglu, for the important discussions, new ideas and for opening my mind to many possibilities to make this work reach new heights. Her advice and knowledge transfer played a large role in this project.

I am also very thankful to all members of GBDI. Their support and hints helped this work progress faster and prevented many setbacks.

Finally, I would like to thank the financial support from the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Finance Code 001; the Fundação de Amparo à Pesquisa do Estado de São Paulo - Brasil (FAPESP) – 2021/05623-2, 2020/07200-9, 2018/05714- 5 and 2016/17078-0, and; the Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil (CNPq).

“We are as much automata as minds.”
(Blaise Pascal)

ABSTRACT

SILVA, G. D. F. **Detecting outliers and annotating their types with indexing structures**. 2022. 99 p. Dissertação (Mestrado em Ciências - Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

The constant increase in the amount of data available on the internet is accentuated with the popularization of technologies such as 5G and Internet of Things. In datasets of large volume there is usually a strong presence of outliers that are not detected or that are just discarded. The outlier detection literature demonstrates that the investigation of these singular instances can provide new insights into the behavior of systems and people. This inspection allows diseases to be identified early, financial market trends to be better interpreted and cybersecurity attacks to be prevented. However, outlier detection techniques carry limitations, being: (1) dependent on the availability of the instances features, which can generate privacy issues; (2) poorly scalable and; (3) capable of providing only a binary separation that allows detecting outliers, but not classifying them so that they are better understood. Starting from an unlabeled dataset for which only the distances between the instances are available, how to detect outliers and categorize them by type efficiently? In the vast literature on outlier detection, there is no work, as far as we know, that deals with the problem of annotating outliers. Outliers can be classified into three large groups: (a) global outliers, instances that are severely different from others in the dataset, such as errors during insertion of information into a database; (b) local outliers, instances that, despite being similar to the others in the dataset as a whole, have minimal variations that make them different in a smaller scope, for instance, a football player who makes many mistakes while playing in a strong team and; (c) collective outliers, small groups of instances that are, simultaneously, quite different from the rest, such as a denial-of-service cyberattack, with few machines having similar harmful behavior. In this project we introduce C-ALLOUT: a new method for detecting outliers that is also able to categorize them by type. C-ALLOUT is able to maintain itself in terms of equality, or even superiority, when compared to state-of-the-art algorithms, still contributing with the annotation of outliers, a task that competitors are not able to perform. C-ALLOUT is based on Slim-tree, an indexing structure that makes it scalable, with $O(n \log n)$ complexity of time and space. Our proposed method deals with both scenarios: having the features available or limited to distances only. Finally, C-ALLOUT works without depending on any interaction with the user, being parameter-free by default, the ideal for unsupervised tasks like outlier analysis.

Keywords: outlier detection, outlier annotation, anomalies, indexing structures, Slim-tree.

RESUMO

SILVA, G. D. F. **Detectando anomalias e anotando seus tipos com estruturas de indexação**. 2022. 99 p. Dissertação (Mestrado em Ciências - Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

O aumento na quantidade de dados disponíveis na internet se acentua com a popularização de tecnologias como 5G e Internet das Coisas. Em grandes bases de dados costuma haver forte presença de anomalias não detectadas ou apenas descartadas. A literatura de detecção de anomalias demonstra que a investigação dessas instâncias singulares pode fornecer novas perspectivas sobre os comportamentos de sistemas e pessoas. Essa inspeção permite que doenças sejam identificadas precocemente, tendências do mercado financeiro sejam melhor interpretadas e que ataques de segurança digital sejam impedidos. Contudo, as técnicas de detecção de anomalias carregam limitações, sendo: (1) dependentes da disponibilidade dos atributos das instâncias, o que pode gerar problemas de privacidade; (2) pouco escaláveis e; (3) capazes de prover apenas uma separação binária que permite detectar anomalias, mas não classificá-las para que sejam melhor entendidas. Partindo de um conjunto de dados não rotulados para o qual apenas distâncias entre as instâncias estão disponíveis, como detectar anomalias e categorizá-las por tipo de forma eficiente? Na vasta literatura de detecção de anomalias não há trabalhos, até onde sabemos, que lidam com o problema de anotação de anomalias. Anomalias podem ser classificadas em três grandes grupos: (a) anomalias globais, instâncias severamente diferentes das outras, como erros de inserção de informações em uma base de dados; (b) anomalias locais, instâncias parecidas com algumas outras da base de dados como um todo, mas com variações mínimas que as tornam diferentes em um escopo menor, como um jogador de futebol que acerta poucas jogadas jogando em um time forte e; (c) anomalias coletivas, pequenos grupos de instâncias que são, em conjunto, bastante diferentes das restantes, como um ataque cibernético por negação de serviço, com poucas máquinas tendo um comportamento nocivo semelhante. Neste projeto é apresentado o C-ALLOUT: um novo método para detecção e anotação de anomalias. C-ALLOUT é capaz de se manter em nível de igualdade, ou até superioridade, quando comparado aos algoritmos do estado da arte, ainda contribuindo com a anotação das anomalias, uma tarefa que os competidores não são capazes de realizar. C-ALLOUT toma proveito da Slim-tree, uma estrutura de indexação que o torna escalável, atingindo complexidade de tempo e espaço $O(n \log n)$. O método funciona tendo acesso aos atributos das instâncias ou limitado a distâncias. Por fim, C-ALLOUT não depende de nenhuma interação com o usuário, sendo livre de parâmetros por padrão, o ideal para tarefas não-supervisionadas como análise de anomalias.

Palavras-chave: detecção de anomalias, anotação de anomalias, casos de exceção, estruturas de indexação, Slim-tree.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ABOD	Angle-based Outlier Detection
ABOF	Angle-Based Outlier Factor
AP	Average Precision
AUCPR	Area Under the Curve of Precision-Recall
AUCROC	Area Under the Curve of Receiver Operating Characteristic
CBLOF	Cluster-Based Local Outlier Factor
DB-Out	Distance-Based Outlier Detection
DPMM	Dirichlet Process Multinomial Mixture
GLAD	Global and Local Anomaly Detection
GSDPMM	Gibbs Sampling algorithm for the Dirichlet Process Multinomial Mixture
kNN	k-Nearest Neighbors
LOCI	Local Correlation Integral
LOF	Local Outlier Factor
LR	Learning Rate
LRD	Local Reachability Distance
MAM	Metric Access Methods
MDEF	Multi-granularity Deviation Factor
MLP	Multilayer Perceptron
MST	Minimal Spanning Tree
OCSVM	One-Class Support Vector Machine
RAE	Robust Deep Autoencoder
RBF	Radial Basis Function
RD	Reachability Distance
RPCA	Robust Principal Component Anaylsis
SIF	Similarity Isolation Forest
SRA	Spectral Ranking of Anomalies
SVM	Support Vector Machine

CONTENTS

CHAPTER

1

INTRODUCTION

The amount of data available for data mining is increasing by the day. Along with this data availability, anomalous instances are becoming ubiquitous. The outlier analysis can help in many ways to develop useful tools or to reach new insights and conclusions about the data being processed. The sections in this chapter give an introductory view over our work. Our motivation and contributions are discussed and our hypothesis is presented as well as our objective. We remark that, in this work, instances, observations, points and objects are used interchangeably.

1.1 Context

Outlier detection has a number of usages across different fields of study. Many outlier detection techniques have been already proposed and they take advantage of several different aspects of machine learning and data mining. In the literature, one can find various detection techniques broadly classified as those based on distances, density, statistical modeling, clustering, angles, subspaces, etc. (HAN; KAMBER; PEI, 2012). However, a key gap can be found in the literature: methods that are able to not only detect outliers but also to annotate their type. Three main types of outliers can be identified in data mining: global, local and collective as it can be seen in Figure 1(a).

One can argue that there are already many existing methods (BANDARAGODA *et al.*, 2018; BREUNIG *et al.*, 2000; KRIEGEL; SCHUBERT; ZIMEK, 2008; MOONESINGHE; TAN, 2006; PAPADIMITRIOU *et al.*, 2003) capable of detecting all three types of outliers. Still, they are unable to annotate them explicitly. Usually, in currently available outlier detection algorithms, no labels are returned. These algorithms output either binary labels (outlier or inlier), or outlierness scores for each point. Therefore, it is impossible to interpret the outlier type using the output of such methods.

In fact, to the best of our knowledge, there is no outlier mining algorithm able to annotate

outliers of all three types. In this research, we found that Spectral Ranking of Anomalies (SRA) (NIAN *et al.*, 2016) is the algorithm which comes the closest to being able to annotate outliers. Nevertheless, it only provides a single ranking along with a binary flag that differentiates outliers in two types: scattered and clustered. The solution, although useful, cannot satisfactorily address the task (see related work in Chapter 3), and performs poorly in experiments (Chapter 5).

Being key indicators of faults, malicious activities, errors and overall points of interest, outliers not only identified, but even annotated, can provide beneficial information to analysts. Sensemaking, triage, troubleshooting can all be made better or even available using this extracted knowledge. Global outliers, being the most extreme cases, are the simplest form of outlier. The global name comes from the fact that it is far away from every other point in space. Local outliers are distinguished from most points, yet they bear many resemblances to a set of the points. Collective outliers can be understood as a group of similar global outliers: many overall singular observations which form a cluster by having similarities to some others singular observations. Collective outliers cannot be close to an inlier cluster since that would make them an appendix to the inlier cluster.

The different types of outliers can be observed under the same context: a guitar in an orchestra of trumpets is a global outlier, an instrument that is very different from every other; an off-tone trumpet is a local outlier as even though it shares many features with the rest of the orchestra, it is still producing unique sounds, a particular feature; finally, a small set of harmonicas constitute a collective outlier, as they are very different from the orchestra, but still similar to each other, generating a small detachment from the overall band.

While outlier detection has been applied over many different contexts, as to understand the behavior of customers of a company that access its platform (ZHANG; BENYOUCEF, 2016), the different medical observations made over a patient (CRIMINISI, 2016), or even variations in currency prices (ALESSANDRETTI *et al.*, 2018). Real-world applications for outlier annotation, although not yet present in the literature, can be readily found. Global outliers can be a human mistake while inserting values of a transaction in the system, where $1000.00 would be informed as $100000. Local outliers can be a toxic bottle of beverage among many good ones, where the difference would be only the amount of a certain ingredient. Collective outliers may be small groups of cancerous cells having much in common with each other while being very different from the majority of healthy cells.

1.2 Problem and Motivation

Going beyond outlier detection, outlier annotation by type is a more general but also strictly harder problem. Being a sort of classification task, it is still harder than most categorization problems as the task remains an unsupervised one. Outlier detection tasks, in general, do not provide labels. Doing a post-hoc analysis by annotating outliers after detection may seem a good

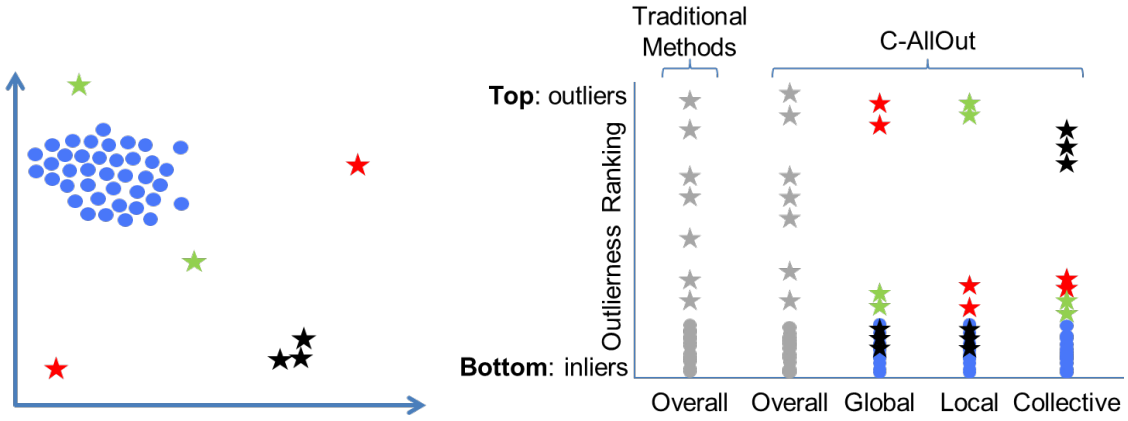
way to go. However, multiple issues arise from that scenario as it would require knowledge about the dataset clusters and similar points in the dataset. Having this knowledge, one could define local outliers as being in the boundaries of the clusters and collective outliers as micro-clusters having points with high outlierness scores. Unfortunately, clustering is not an easy task. Most clustering or distance-based algorithms in the literature require hard to define user hyperparameters, like the number of clusters k in k-Means (STEINLEY, 2006), number of neighbors k in k-Nearest Neighbors (kNN) (RAMASWAMY; RASTOGI; SHIM, 2000) and Local Outlier Factor (LOF) (BREUNIG *et al.*, 2000), or even the minimum density $MinPts$ in Density-Based Spatial Clustering of Applications with Noise (DBSCAN) (ESTER *et al.*, 1996). Data sometimes have clusters with different shapes, sizes, densities, distributions and, sometimes, do not even have clusters. All of these variables are probably why there is not post-hoc attempts on outlier annotation in the literature.

It is important to remark that, to the best of our knowledge, the problem of outlier annotation by type has not been tackled yet in the literature. Importantly, the method proposed in this work is self-contained. This means that we do not do any post-hoc analysis on any outlier detection algorithm output. Instead, we provide a new algorithm, C-ALLOUT, that can detect and annotate outliers at the same time.

To visualize how C-ALLOUT works in comparison with common outlier detection algorithms, we show in Figure 1(a) the plot of a two-dimensional dataset having outliers that is used as input for general outlier detection algorithms. In the figure, we can see the red global outliers very far from the blue inliers, with the green local anomalies lying closer to the blue cluster and, finally, a small group of black collective outliers. Meanwhile, Figure 1(b) shows first the output of traditional methods for outlier detection, having no separation between outlier types, just a single outlier ranking. C-ALLOUT, on the other hand, returns four rankings: one overall (gray) that refers to the outlier detection aspect of our algorithm, and three type-specific rankings, where outliers are identified by their type, using the same colors described earlier. Note that for type-specific rankings, say the global ranking, the goal is to put global anomalies in the top and let the other anomalies remain together with inliers, at the bottom positions.

Using the Slim-tree (TRAINA *et al.*, 2000), an efficient data indexing structure, we introduce novel measures that can effectively attribute outlierness scores for each point by type, be it a global, local or collective outlier. The Slim-tree supports our method with log-linear time complexity and a construction phase that needs to query only $O(n \log n)$ pairwise distances between the points being inserted in the structure.

As a desirable side-effect, our solution can deal with specific, and sometimes troublesome, settings for which only pairwise distances are available. In these cases, the points may not fit in a feature space, but their distances may still be calculated. Domains with complex objects, such as galaxies, graphs, fingerprints or videos, may have distances between objects that can be more convenient to provide for domain experts than measuring or attributing features to them.



(a) Example input. (b) Example output.

Figure 1 – (a) Example toy dataset with inliers (circles) and 3 types of outliers (stars): global (red), local (green), and collective (black); (b) Existing methods provide an overall outlier ranking regardless of type. C-ALLOUT also provides individual outlierness rankings by outlier type.

In other scenarios, for privacy reasons, as in medicine or finance, providing only pairwise distances can be safer than revealing the sensitive feature values. Note that, while addressing these scenarios, our work is not limited by them and can handle traditional settings where features are accessible. Distance functions that can find distance measures in feature spaces are plentiful.

Conclusively, we hypothesize that with the support of an indexing tree data structure, it is possible to detect outliers and annotate their types with sub-quadratic scalability, without losing in accuracy to state-of-the-art outlier detection algorithms.

1.3 Objective

The main objective of this MSc work is to employ tree-based data indexing structures to detect and annotate outliers in multidimensional sets of data. Additionally, we work on the treatment of the issues of efficiency and scalability, still incipient in the literature for outlier detection. To make this viable, we propose a new algorithm based on the usage of metric access methods and a state-of-the-art indexing structure. Nonetheless, we work on establishing new grounds for outlier type annotation while maintaining the progress already solidified by the literature's best outlier detection algorithms.

1.4 Contributions

Following the hypothesis previously mentioned, we propose a novel algorithm capable of addressing both outlier detection and outlier annotation tasks. In accordance to our experiments, our main contributions are summarized as:

- **New outlier detection algorithm:** we introduce C-ALLOUT, a novel algorithm that can

effectively detect and annotate outliers by type. To our knowledge, there is no algorithm in the literature able to perform both tasks.

- **Outlier annotation:** C-ALLOUT provides four individual rankings (as in Figure 1(b)): one overall ranking and three type-specific rankings for local, global and collective outlierness scores. All rankings are built in an efficient manner by leveraging C-ALLOUT's indexing data structure.

- **Effectiveness along with new benchmark datasets:** C-ALLOUT is compared against state-of-the-art detectors on traditional benchmark datasets. Our method holds on par or significantly better performance in our experiments. Considering carefully designed testbeds having annotated outliers, we show the effectiveness of C-ALLOUT at outlier annotation, a task no competitor can really address. SRA (NIAN *et al.*, 2016), having very limited capabilities on outlier annotation is surpassed by C-ALLOUT.

- **Miscellaneous desirable properties:** Several other interesting properties are maintained by C-ALLOUT. Firstly, it is a virtually parameter-free algorithm requiring no user inputs. It can deal with settings in which only pairwise similarities are accessible (without a feature space). Lastly, it is scalable, bearing log-linear time and space complexity for model construction and scoring.

1.5 Final Considerations

This chapter provided context and brief information about the state of the art in detecting outliers. The problem and motivation were also presented, followed by the objective and contributions. With this introductory view settled, we organize this monograph in six chapters. Chapter 2 presents a basis for the main detection techniques for outliers, as well as the conceptualization of the techniques that were used in this research. In Chapter 3, related works are discussed and compared with our solution. Then, the methodology of this work is addressed by Chapter 4. Chapter 5 reports the obtained results along with their discussion. Lastly, in Chapter 6 we share our final thoughts and discuss the main contributions of this MSc work.

For reproducibility purposes and to encourage future work on both outlier detection and annotation, we make available the code for C-ALLOUT, all datasets used in experiments, data generators and also the data and satellite imagery used in our case study. Everything can be found at <https://bit.ly/3JezWe4>.

CHAPTER

2

PRELIMINARIES AND THEORETICAL FOUNDATION

The numerous available outlier detection techniques are often bounded by resource limitations that renders them not applicable to large databases. This is specially true for distance-based outlier detection, which in many iterations include clustering techniques as well. To detect outliers using distances, methods need to compare each point with the whole set of points or with large subsets of the dataset. Eventually, many detection algorithms can be close to, or even above, quadratic in time performance, as it is shown in this chapter. Due to this context of finite resources, new outlier detection techniques are being proposed with the intent of being scalable (LIU; TING; ZHOU, 2008; JUNIOR; CORDEIRO, 2019b; CABRAL; CORDEIRO, 2020). In distance-based outlier annotation, as is our case, multiple distance computations are needed to call out the outliers by their type. Having eyes on the presented issue as well, our proposal is grounded on the usage of an indexing structure that makes it more scalable and faster overall.

This chapter sets up the preliminaries of our work. Definitions are provided for all the foundations of our solution: outliers and their types, common outlier detection approaches, metric spaces and the Slim-tree indexing structure. Slim-tree working aspect is discussed to cover its construction and internal schemes. Finally, a problem definition and an overview on the outlier annotation task is conveyed.

2.1 Outlier Detection

In the large and interdisciplinary literature encompassing outlier detection some groups can be outlined according to the context for which the methods were devised. Originally an statistical task and later a mostly computed-based one, outlier mining has already set paths on many different fields. We can see the existence of outliers being investigated in finance (TRIPATHI *et al.*, 2018), social networks (BINDU; THILAGAM; AHUJA, 2017), information

security (TRIPATHI *et al.*, 2018), cancer diagnosis (DURAJ; CHOMATEK, 2017) and others. Being a general task with many applications, incompatibilities arise between the proposed techniques, which can be very field specific. For each area of interest, an anomaly can take deviations in definition. The outlier definition dilemma is also noticeable in the different detection approaches.

Although it is hard to formally define what is an outlier, all definitions have a common ground. One of the first endeavors on defining outliers was made by Hawkins (1980): "An outlier is an observation, which *deviates so much* from the other observations as to arouse suspicions that it was generated by a different mechanism". Another interesting attempt was arranged by Barnett and Lewis (1974): "An outlier is an observation (or subset of observations) which *appears to be inconsistent* with the remainder of that set of data". A more recent bid done by Chandola, Banerjee and Kumar (2009) was also not able to be objective: "Those measurements that *significantly deviate* from the normal pattern of sensed data".

To try to avoid the lack of objectivity and dissonance across outlier definitions, a three-level index was proposed by Aggarwal (2015) to classify observations into three groups. Denominated as outlierness score, the index sets boundaries to help discretize data: (i) inliers: observations that describe normal, or expected, behavior for the domain in which they are found; (ii) noise: observations that define the semantic limit between an expected observation for the domain and an unexpected observation; (iii) outliers: observations that are considerably different than expected for the domain in which they are found. The subjectivity of these definitions can be understood as inherent to the nature of outliers. It is always going to be necessary to set a limit of inconsistency for an object to be stated as an outlier inside its group.

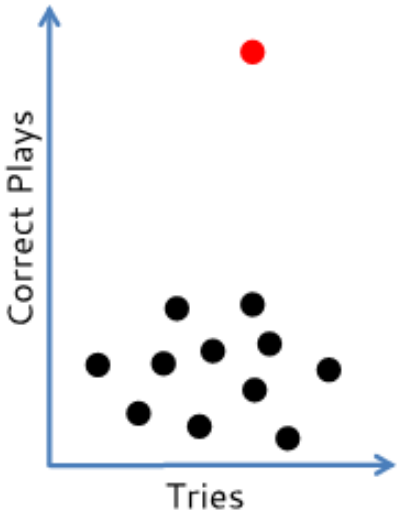
In certain scenarios, the domain expert can categorize a noise as an anomaly, in others, as an observation close enough to the standards. In this sense, the outlier definition will always need input from the user or domain expert and there is no one-above-all definition for outliers. In time, it is possible to say that outlier analysis always follows a common objective direction: every outlier data must be able to provide additional information about the behavior of a system, which the normal data is not able to express.

Going further deep on the outlier definition scrutiny, some types of outliers are often differentiated during analysis (AJITHA; CHANDRA, 2015). Three main types of outliers can be cited: global, local and collective. Informal definitions for the three types are given below, and they are also illustrated in Figure 2:

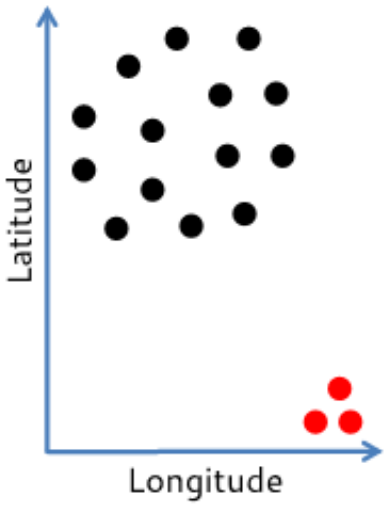
- **Global outlier:** when an observation deviates from the rest of the dataset as a whole. An example is a denial of service cyberattack, when attackers send a huge amount of hits to a server in a short time. The number of accesses to the server during the day of the attack will be considerably greater than the number of legitimate accesses to the server during an average day.

- **Local outlier:** when an observation deviates only from other observations close to it, but not necessarily from the rest of the dataset as a whole. Observations on broadband consumption can be considered local outliers when the user downloads a file from the internet. In this example, the file download temporarily increases bandwidth consumption, however, this behavior is always repeated when the user downloads files. Therefore, if only the bandwidth consumption during downloads is considered, all local outliers become inliers.

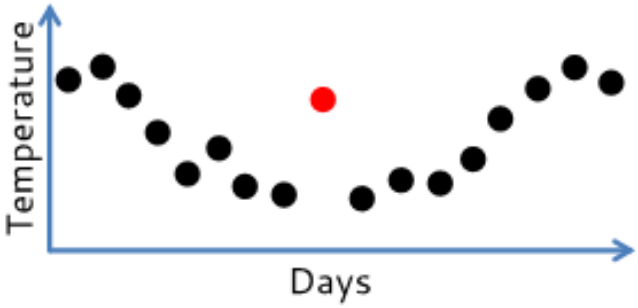
- **Collective outlier:** when a group of observations deviates from the dataset as a whole. The maximum size of a group to be considered a collective outlier and not an inlier cluster is difficult to define and it depends on the analysis of a domain specialist. In general, collective outliers cannot be large enough to establish a new expected pattern. For example, players who use the same malicious software to cheat in a game form a collective outlier group. They exhibit similar behavior with each other, but not with the significantly larger portion of normal players.



(a) Global outlier.



(b) Collective outlier.



(c) Local outlier.

Figure 2 – The three outlier types in our scope.

The detection of outliers must be applied to a set of instances, commonly referred to as a dataset. Each instance may contain one or more dimensions, being called multidimensional. In multidimensional datasets, the input instances have a uniform representation, all being expressed with the same dimensionality. However, it is also possible for the data to be "dimensionless", when each instance can be represented by a different number of dimensions. Some numerical representations of graphs are dimensionless. For example, if each vertex in a graph is an instance and each one of its edges are attributes, we reach a dimensionless instance since the number of edges around vertexes can vary a lot.

If the outlier analysis is done by calculating the distance between instances, some common distance functions can be used, such as, Euclidean, Mahalanobis (LEYS *et al.*, 2018) or Minkowski (FIGUEROA-GARCÍA; MELGAREJO-REY; HERNÁNDEZ-PÉREZ, 2018). Both multidimensional and dimensionless data can be analyzed in metric spaces because they can be represented using a distance function that calculates their dissimilarity following the metric space properties. Since the base data structure of C-ALLOUT, Slim-tree (TRAINA *et al.*, 2000), is based on metric distances, it provides C-ALLOUT with the ability to find outliers in both multidimensional and dimensionless realms. Further details on Section 2.3 and Section 2.4.

Among the different methods for detecting outliers, the most common in the literature are those based on distance. Their popularity is due, for the most part, to the ease of interpretation of the results and instinctive understanding of the techniques. It is usually more intuitive to understand an outlier as an object distant from the others than as an object that escapes the distribution of the dataset. Another factor that contributes to the popularity of distance-based techniques is the existence of algorithms that are used for other purposes that can be easily adapted to identify outliers, for instance, k-Nearest Neighbors (kNN) (MALINI; PUSHPA, 2017).

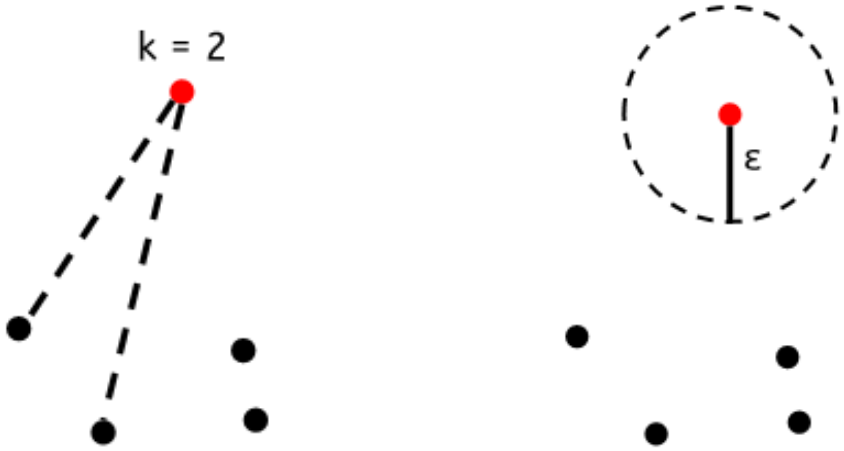
The next subsections discuss the main subdivisions of unsupervised outlier detection methods. Less common groupings of methods and the supervised ones are not reviewed here. Some of these approaches include, linear models (SHAWE-TAYLOR; ŽLIČAR, 2015), algorithms based on information theory (WU; WANG, 2011) and genetic algorithms (SHI; LI, 2013).

2.1.1 Neighborhood-based Outlier Detection

Being the most common type of outlier detection algorithm, the neighborhood-based approach uses the closest objects to an object of interest to reach an outlierness score. A neighborhood must be defined using a distance, or similarity, function capable of summarizing the differences in the features of the objects. Usually, if objects are multidimensional, we can say that the distance function is going to indicate that the neighbors of an object are the nearest objects in a feature space. Sometimes, feature values are not available, and sometimes, the data is not multidimensional. For these special cases, the distance function must find a way to tell

how close one object is to another (i.e. how close a word is to another word). The Hamming distance is an example of a distance function that does not need a feature space to work.

The two most common ways to define a neighborhood is by either considering as neighbors the k nearest objects or considering as neighbors the objects that are closer than a defined threshold ε. These approaches are also known as nearest neighbor queries and range queries, respectively. Even if both approaches are similar, they can lead to very different results. Take Figure 3a for instance, even if looking at the same data, a k-nearest neighbor approach is always going to find neighbors, no matter the k value (how far they are). On the other hand, as we can see in Figure 3b, a range approach is not guaranteed to find neighbors if ε is too small.



(a) k-Nearest Neighbors with $k = 2$. (b) ε-neighborhood.

Figure 3 – Outlier detection using neighborhood-based approaches. Outlier in red and inliers in black.

Nearest neighbors approaches need less information about the dataset and they will find neighbors no matter the density or dimensionality of the data. Range-based approaches need initial information on the data to work properly. If the data is too sparse in the space, or has a very high dimensionality, parameter ε needs to compensate for that. This is an important factor because neighborhood-based outlier detection is understood as useful because it does not require prior knowledge on data distribution.

The connection of these two main approaches for neighborhood-based detection can be understood when defining that, for a given instance, the distance to its k^{th} nearest neighbor is equivalent to the radius ε of a hypersphere centered at this given instance (CHANDOLA; BANERJEE; KUMAR, 2009).

The first algorithm to introduce the range-based approach to outlier detection was Distance-Based Outlier Detection (DB-Out) (KNORR; NG, 1998). In the first iteration of the proposed algorithm, a radius ε is provided for it to count how many neighbors exist around each instance within the maximum distance ε. Another parameter provided by the user to the algorithm is the fraction p. If the instance studied has less than a p fraction of the whole dataset inside its ε radius, it is flagged as outlier.

Later on, another version of DB-Out was proposed where p is removed and instead of flagging outliers, the algorithm scores outliers. The fraction of instances inside a ε radius around a studied instance is the outlierness score of such studied instance. Other methods tried to take further the idea of DB-Out by being able to deal with categorical data (LI; LEE; LANG, 2007) and by using the connectivity property instead of a distance function (WEI *et al.*, 2003).

KNNOut (RAMASWAMY; RASTOGI; SHIM, 2000) was the first to algorithm to use the nearest neighbor concept in the outlier mining field. Being an adaptation of the popular classification algorithm kNN (PATRICK; III, 1970), KNNOut works by using the distance to the k^{th} neighbor as outlierness score for the object of interest. In its naive implementation, it has time complexity of $O(n^2)$, however, many indexing structures have being employed to reduce its average-case time complexity to $O(n \log n)$.

Local Outlier Factor (LOF) took one step further from KNNOut seeing its inability to deal with datasets having varying densities. This is done through a concept called average reachability distance, in which LOF defines how sparse, or how far from a cluster, a point of interest and its k neighbors are. Points with high LOF, or high outlierness score, are the ones that have the largest average reachability distance in comparison with their k neighbors. More details on KNNOut and LOF can be found in Chapter 3.

Another important neighborhood-based algorithm for outlier detection is Local Correlation Integral (LOCI) (PAPADIMITRIOU *et al.*, 2003). Since finding a good value for k is not a trivial task, LOCI introduces the concept of Multi-granularity Deviation Factor (MDEF) and goes in a different direction. It works in a similar fashion to LOF, using ε-neighborhoods instead of k-nearest neighbors. Anyhow, LOCI has a problematic time complexity and hence, an approximation version was also proposed in the same article.

Many approximation versions have also been proposed to kNN for dealing with outliers, for instance, HilOut (ANGIULLI; PIZZUTI, 2002). It approximates the nearest neighbor concept by using the Hilbert curve to reduce the dimensionality of the data to one. Since the Hilbert curve returns the points ordered by proximity in this single-dimensional representation, HilOut can use it to find the k-nearest neighbors of a studied point.

Overall, even if being able to have a good performance in quality measures, neighborhood-based outlier detectors have a hard time with their time complexities. It is usual to find time complexities of $O(n^2)$ for these algorithms when no indexing structure is incorporated. If defining a distance function to support these methods is not considered a hard task, the definition of k by the user is very tricky by itself. Tuning hyperparameters for neighborhood-based anomaly detectors is problematic considering that, after all, neighborhood-based anomaly detection is usually an unsupervised task.

2.1.2 Clustering-based Outlier Detection

Algorithms based on clustering are intuitive solutions for outlier detection, taking into account that their objective is also to group instances by similarity (JAIN; DUBES, 1988). Clustering-based outlier detection also bears many intersections with neighborhood-based approaches. The clustering process begins with the input dataset being placed over a multidimensional space. Each new entry is placed under a group, normally called a cluster. A cluster contains instances similar to each other. The number of clusters is usually defined in advance with a user-defined hyperparameter or using some heuristic. The first objects received as input form the first clusters. Clusters can be resized and redefined according to the degree of similarity across all objects of each cluster internally and also externally.

Clustering algorithms can provide the outliers of a dataset as a by-product of their execution. In some cases, the algorithms fail to insert some objects into a cluster, due to their dissimilarity with the features that standardize the existing clusters. These objects can be classified as outliers, as it can be seen in Figure 4. In general, clustering-based outlier detection is performed by following one of these strategies: (CHANDOLA; BANERJEE; KUMAR, 2009)

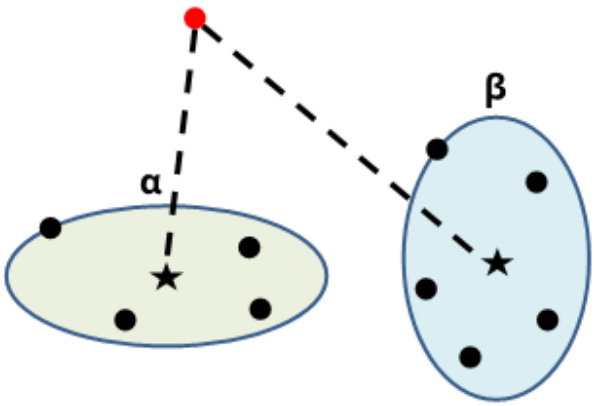


Figure 4 – Outlier detection using a clustering-based approach. Outlier as red circle, inliers as black circles and black stars as cluster centroids. α and β are clusters.

- Consider that inliers always belong to a cluster, while outliers do not belong to any. In this approach, a maximum distance for the size of a cluster is defined, such as the radius of a circle. Therefore, when an instance exceeds the radius of all clusters, it is not inserted into any cluster and is considered an outlier. An example of an algorithm based on this premise is DBSCAN (ESTER *et al.*, 1996).

- Consider that inliers are located close to the centroids of the clusters, while outliers are far from the centroids. This notion is used in outlier detection based on KMeans (STEINLEY, 2006). In this case, a threshold distance to the centroid of the cluster is defined so that if an instance goes beyond the threshold, it is considered an outlier, even if it belongs to a cluster.

- Consider that inliers are in large or dense clusters, while outliers are in small or sparse clusters. The FindCBLOF (HE; XU; DENG, 2003) algorithm, for example, defines the metric Cluster-Based Local Outlier Factor (CBLOF) that can represent the size of the cluster in which the instance is located, and also the distance from the instance to the centroid of the cluster.

Not needing labels to establish a model, outlier detection by clustering is an unsupervised technique. Because of this, the algorithm has a natural advantage for the detection of outliers, especially when the database analyzed is dynamic, with a data flow that constantly updates the dataset with new observations. In addition, clustering algorithms can also be adapted to work with mixed data types, as long as it is possible to establish a degree of similarity between the data, using distance functions, for example.

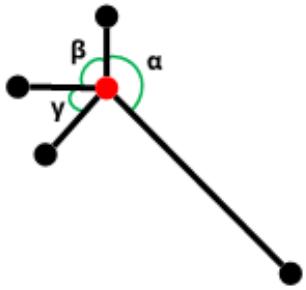
As clustering methods seek to find patterns in data by means of grouping, their original proposal was not to detect anomalous observations that do not fit any foreseen pattern. Thus, for these algorithms to bring results that indicate outliers, they need to be adapted for this practice. In some cases, however, the clustering algorithms assume that all observations must be part of some clustering, such as KMeans. This setting makes it difficult, although still possible (BARADARAN; REDDY; THAKUR, 2017), to classify an observation as outlier, since all observations are forced to match some pattern. Anyway, setting thresholds to draw a line between inlier and outlier observations is also not a trivial task and usually requires prior knowledge on the data.

2.1.3 Angle-based Outlier Detection

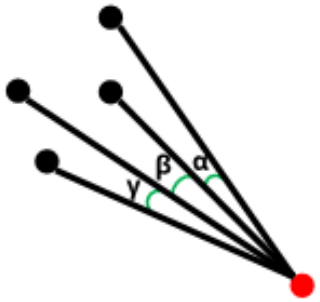
In the outlier analysis based on angles, comparisons between objects are performed until all objects in a selected scope are compared. However, the comparisons do not consider the object features themselves, but rather the relationships between the objects at a higher level. These relationships, which cannot be seen as intrinsic to the data, are the angles formed by triplets of objects. To study these angles, the observations need to be placed on a vector space, where each of one of them is represented by a point whose coordinates are the observation feature values. The analysis seeks to observe the variation in the angle between two points using a third point as a reference, called pivot. The proposal for this detection model is based on a principle that states that if a point is surrounded by others, that is, it is part of a group, the angles formed between that point and the other points have a high variation in degrees.

Inlier observation forms large and small angles with other observations (Figure 5a). Outlier observations have little variation in their angles because they are far away from the other instances (Figure 5b). Hence, the smaller the variation in the possible angles around a point, the greater the chances of the point being an outlier. The angular variation can be defined as the outlierness score for that point. A threshold can be established based on the angular variation

to define which points should be considered normal and which ones should be considered anomalous. Otherwise, a ranking can be provided with the outlierness scores sorted. Kriegel, Schubert and Zimek (2008) established an outlierness score called Angle-Based Outlier Factor (ABOF) which weights the angle variation using the related distances.



(a) Inlier observation having large variations across angles α, β and γ.



(b) Outlier observation having small variations across angles α, β and γ.

Figure 5 – Outlier detection using an angle-based approach. Point of interest in red and others in black.

Angle-based Outlier Detection (ABOD) (KRIEGEL; SCHUBERT; ZIMEK, 2008) aims to produce better results for high-dimensional datasets, as the variation in angles is not much affected by the increase in the number of dimensions of the data, differently from the distance-based approach. Like this, the curse of dimensionality (HAMMER, 1962) can be avoided. However, as angles between two points depend on a reference point, three points are needed for the analysis of each possible angle. Therefore, the time complexity of the algorithm is cubic, $O(n^3)$, since the algorithm needs to cover all possible triplets of points. To mitigate this issue, the authors proposed approximations for ABOD, including FastABOD which calculates angles only considering the k-nearest neighbors of the point of interest. There is also a lower bound approximation for ABOF, LB-ABOF, that makes ABOD have quadratic time complexity, overall keeping the same accuracy. FastVOA (PHAM; PAGH, 2012) even provides a near-linear approximation for ABOD, but other issues with the technique remain hard to tackle.

Angle-based outlier detection has some limitations that prevents it from being generalist. With collective outliers, that is, small groups of objects that stand out from the rest, the analysis is impaired because the outlying instances have a high angular variation with their neighbors. Another factor to be taken into account is that datasets can take the form of "lines" of points. The points at the limits of the lines are going to have small angle deviance but they are still in the lines. In this way, they will, but should not, get flagged as outliers. Consequently, even if the principle of angle-based detection algorithms works better with high-dimensional data, there are generality concerns.

2.2 Outlier Annotation

In the broad literature on outlier detection, even if many applications for outlier analysis have been considered (AGGARWAL; YU, 2001), there is little to no discussion on outlier type differentiation. Even if outliers have already been set asunder by type, what has mostly been discussed is that superior outlier detection algorithms should be able to detect all types of outliers. This objective comes from the idea that outliers are false or noise data that should be removed from the dataset in order to keep it clean and, therefore, useful. However, outlier analysis can benefit from studying outliers and even more by categorizing them. The examples provided along with the outlier definitions in Section 2.1 give a short perspective on the gains of outlier annotation.

Between the techniques developed to have the ability to detect multiple types of outliers, we can cite the recent work Global and Local Anomaly Detection (GLAD). GLAD is able to detect both local and global outliers at the same time, but cannot annotate the outlier types. Differently from C-ALLOUT, GLAD is a not a distance-based method, but a deep learning model that uses autoencoders on image data, specifically. For local outlier detection, the proposal is that when autoencoders reconstruct the original local outlier image fed to them, in the decoding phase, they will return an image with a high error rate. That is because the network is not trained enough to work with outlying examples. On the other hand, global outlier detection happens in the encoding phase of the autoencoders. Deep learning autoencoders compress the input and reduce its dimensionality when encoding. Global outliers can be detected because in their reduced version the differences between them and inlier data is substantial. Even though the detection step for global and local outliers is independent, it is not clear if the algorithm is returning only global outliers in the global outlier detection step. The same goes for the local outlier detection step.

Another example on the robust outlier detectors, able to find multiple types of outliers at once, is Spectral Ranking for Anomalies (SRA) (NIAN *et al.*, 2016). This one is even able to marginally separate outliers by type. Using the principal components obtained from the similarity matrix of the dataset, the method is able to find inlier clusters in vector space and to calculate an outlierness score for every point based on the distance to the center of the inlier clusters. At the end, the algorithm also provides a flag which determines if the outlier type found is scattered (local and global) or clustered (collective). However, this flag is binary, so all outliers in the dataset are said to be of the same type. Since SRA can provide limited results on outlier categorization, we selected it as a competitor and it is further discussed in Section 3.2.

Even if some older algorithms, like LOF (BREUNIG *et al.*, 2000) and its variations, were theoretically able to detect multiple types of outliers, the objective was robustness and not outlier annotation. This overview granted context for the literature gap that we explore in this work.

2.3 MAM and Tree-based Indexing Structures

Metric Access Methods (MAM) support the execution of queries and comparisons for data collections represented in a metric space (TRAINA *et al.*, 2002). A metric space is defined by a domain X and a distance function $d : X \times X \to \mathbb{R}$, following the properties:

- Non-negativity: for all $x, y \in X$, $0 \leq d(x,y) < \infty$
- Identity: for all $x, y \in X$, $d(x,y) = 0 \iff x = y$
- Symmetry: for all $x, y \in X$, $d(x,y) = d(y,x)$
- Triangle inequality: for all $x, y, z \in X$, $d(x,y) \leq d(x,z) + d(z,y)$

Many distance functions follow the stated properties. To name a few, we have: L^p-norms, like the Euclidean distance, the Manhattan distance and the Hamming distance. All of them being commonly used in machine learning algorithms.

MAM structures are based solely on distances as the metric properties do not allow the ordering of points. Slim-tree (TRAINA *et al.*, 2000), along with other state-of-the-art MAMs, is developed in the form of a tree. Slim-tree in particular can hold several levels until it indexes the entire dataset. The tree levels are composed of nodes that can be either leaf nodes or index nodes. Intermediate level nodes are index nodes and the ones belonging to the last level of the tree are the leaf nodes. While leaf nodes hold subsets of actual data points, index nodes hold representative points (indexes) of its children nodes. Representative points are only copies of the real data that lies in the leaf nodes.

To determine the representative points and also which points are going to be inserted on each node, the metric distance is fundamental. Metric distances, unlike many other distance measures, have the triangle inequality property that allow the Slim-tree to take shortcuts while computing distances and organize itself much faster. Selection of representative points and node creation are detailed further in the next sections of this chapter.

2.4 The Slim-tree in Detail

Slim-trees (TRAINA *et al.*, 2000) are complex data structures and have many components, algorithms and heuristics for its inner schemes. The following subsections yield definitions and details for its pieces and expose the concepts behind the tree construction.

2.4.1 Defining Slim-tree's Composing Elements

Slim-tree is an indexing tree-based data structure. It relies on distance (or similarity) measures $\mathtt{d}(\mathbf{x}_i, \mathbf{x}_j)$ between pairs of objects $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$, and can be built in $O(n \log n)$ time using

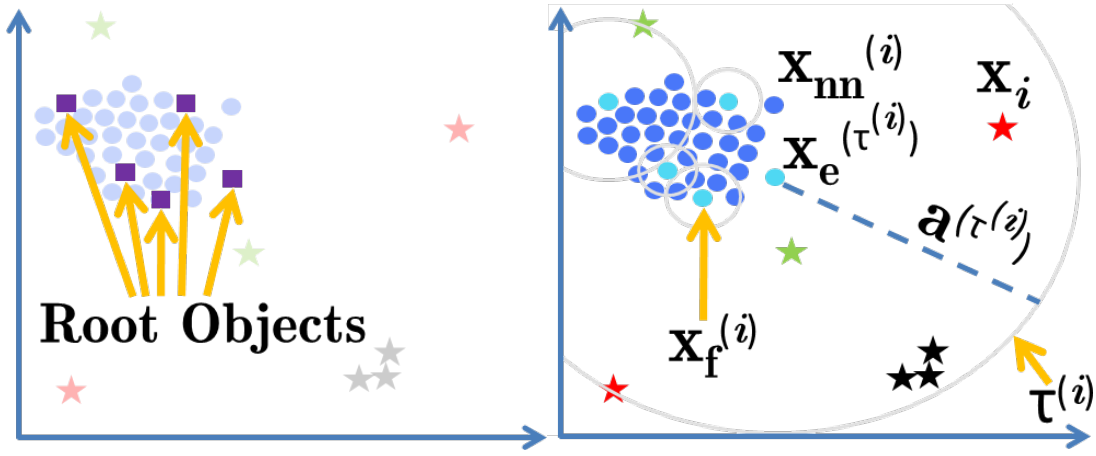
any metric distance function in a feature space. Distances can be either given to the tree without using features (precalculated), or computed directly from the features while building the tree.

A Slim-tree $\mathscr{T}$ organizes its nodes hierarchically into levels. As per Definitions 2.4.1, 2.4.2, 2.4.3 and 2.4.4, $\mathscr{T}$ has always only one root node ρ. Root node ρ holds at most $c \in \mathbb{Z}^+$ points from dataset $\mathbf{X}$, into a set $\mathbf{R}$, where c is the maximum node capacity. Each object $\mathbf{x}_e^{(\eta)} \in \mathbf{R}$ is said to be the representative of a child node η of the root; thus, root node ρ has $|\mathbf{R}|$ children nodes. Node η also follows the same structure: it holds at most c objects from $\mathbf{X}$ into a set $\mathbf{N}$. It is always true that $\mathbf{R} \cap \mathbf{N} = \{\mathbf{x}_e^{(\eta)}\}$, which means that $\mathbf{x}_e^{(\eta)}$ is the one and only object stored redundantly both in η and in its direct ancestor $\alpha_\eta = \rho$.

The representative object $\mathbf{x}_e^{(\eta)}$ is always nearby the *most centralized object* in $\mathbf{N}$. The algorithms that build Slim-tree always try to have as representative the most centralized point in a node. Additionally, it is correct to say that $\alpha_\eta = \rho$ is the direct ancestor of node η. The tree hierarchy continues recursively with node η having $|\mathbf{N}|$ child nodes, each also having their own children nodes, until reaching a leaf node τ. Leaf τ has no children nodes and do not hold representative points, but the actual data points from $\mathbf{X}$. Note that every point $\mathbf{x}_i \in \mathbf{X}$ is stored into the set $\mathbf{L}^{(i)}$ of one and only one leaf node $\tau^{(i)}$.

Definition 2.4.1 (Root). *The root node ρ of a Slim-tree $\mathscr{T}$ built for a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ is the single node existing at the highest level of $\mathscr{T}$. The set of (root) points stored in ρ is denoted by $\mathbf{R}$, where $\mathbf{R} \subseteq \mathbf{X}$.*

Definition 2.4.2 (Leaf). *The leaf node of a point of interest $\mathbf{x}_i \in \mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n\}$ in a Slim-tree $\mathscr{T}$ is the node $\tau^{(i)}$ existing at the lowest level of $\mathscr{T}$ such that $\mathbf{x}_i \in \mathbf{L}^{(i)}$, where $\mathbf{L}^{(i)} \subseteq \mathbf{X}$ is the set of points stored in $\tau^{(i)}$.*



(a) Root node. (b) Leaf nodes.

Figure 6 – Slim-tree $\mathscr{T}$ for the toy data in Figure 1(a) of Chapter 1.

Definition 2.4.3 (Direct Ancestor). *The direct ancestor node α_η of a node $\eta \neq \rho$ in a Slim-tree $\mathscr{T}$ is the node that directly points to η from the previous level of $\mathscr{T}$.*

Definition 2.4.4 (Representative)**.** *The representative object* $\mathbf{x}_{\mathrm{e}}^{(\eta)} \in \mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n\}$ *of a node* $\eta \neq \rho$ *in a Slim-tree* $\mathscr{T}$ *built for a dataset* $\mathbf{X}$ *is the only object both in* η *and in* α_η*.*

Note that, for Definitions 2.4.3 and 2.4.4, since ρ is at the top level of Slim-tree $\mathscr{T}$, it has neither ancestor node nor representative object defined for it. Hence, for the Definitions to be valid, $\eta \neq \rho$. Moreover, ρ is also an index node, as every other node in $\mathscr{T}$ that is not a leaf node $\tau^{(i)}$.

Figure 6 shows a Slim-tree $\mathscr{T}$ built for the toy dataset in Figure 1(a) of Chapter 1. $\mathscr{T}$ has three levels: root node level (the only index node level), leaf node level and object level. Each of the 3 root objects in (a) represents one leaf node in (b). One of them is leaf $\tau^{(i)}$ with radius $\mathrm{a}^{(\tau^{(i)})}$. Object $\mathbf{x}_i$ is in $\tau^{(i)}$; its nearest neighbor is $\mathbf{x}_{\mathrm{nn}}^{(i)}$. The representative of $\tau^{(i)}$ is $\mathbf{x}_{\mathrm{e}}^{(\tau^{(i)})}$, a real object copied to the root node to be a root object. $\mathbf{x}_{\mathrm{f}}^{(i)}$ is the closest representative object to $\mathbf{x}_{\mathrm{e}}^{(\tau^{(i)})}$, called foreign representative (later defined in Definition 4.2.2).

2.4.2 A Functional Overview of Slim-tree

Slim-tree, as many other metric trees, is built in a bottom-up manner. New levels are created just when the leaf nodes $\tau^{(i)}$ are full and have to be split in two. The Slim-tree $\mathscr{T}$ indexes data in abstract inner data structures called pages. The pages are based on hard disk drive pages as Slim-tree was originally proposed to work with large datasets that would only fit in secondary memory. However, pages work more as a concept, since Slim-tree can work using only the main memory.

Pages can have any size, provided that its bytes can be sequentially stored in the main memory of the machine. All of the pages of a Slim-tree $\mathscr{T}$ must have the same maximum size but they can vary in size, according to their current number of entries. Pages are divided in entries, each entry in a page has one, and only one, object related to it. Every leaf and index nodes, including the root, has an associated page and they use their pages to store their content that can be divided in header space and data space. In the following, we present an overview on the information hold by a page (TRAINA *et al.*, 2000):

1. **Index node**:
   a) *Header*:
      i. Type: the type of the node. In this case, it will always be an index node;
      ii. Occupation: the number of entries stored in this node.
   b) *For each entry*:
      i. PageID: the identifier for the page storing the root of the subtree contained by this entry;

ii. Distance: the distance between the object related to this entry and the representative object for the index node;
iii. Number of entries: the number of objects in the subtree contained by this entry;
iv. Radius: the radius of the subtree contained by this entry;
v. Offset: the byte offset to the position of the serialized object related to this entry in the page.

c) *Object*: an array of bytes that stores the serialized version of the object and can be used to rebuild the object to its original form. It is always uniquely related to an entry.

2. **Leaf node**:

a) *Header*:
i. Type: the type of the node. In this case, it will always be a leaf node;
ii. Occupation: the number of entries stored in this node.

b) *For each entry*:
i. Distance: the distance between the object related to this entry and the representative object for the leaf node;
ii. Offset: the byte offset to the position of the serialized object related to this entry in the page.

c) *Object*: an array of bytes that stores the serialized version of the object and can be used to rebuild the object to its original form. It is always uniquely related to an entry.

The above information is used by the Slim-tree to locate any object, or any data related to it, using recursive search algorithms, starting from the root node.

2.4.2.1 Slim-tree's Building Methods

Slim-tree follows some heuristics while the tree is being built. These heuristics help the tree to be balanced and with a good data distribution among the nodes. When an object needs to be inserted into the tree, it searches for all nodes that can cover that object. This means that if the radius of a node covers the space in which the object is located, it is a qualifying node. The search starts from the root node and goes through every index node level until it reaches the leaf node level. Since an index node radius covers all of its child nodes' radii, if an index node radius covers the object being inserted, one of its child nodes also does and the search goes on to the level below. If, in the end, there are no qualifying leaf nodes, the object is assigned to the leaf node whose representative is the closest to the object. When this happens, the radius of the node increases to accommodate the inserted object.

In the case where there is more than one qualifying node, the *ChooseMethod* algorithm is applied to select one qualifying node. This algorithm has four options for selecting the node, or subtree, that will cover the object:

- **biased**: choose the first of the qualifying nodes;
- **random**: choose any of the qualifying nodes at random;
- **mindist**: choose the qualifying node which has its center, or representative object, closest to the new object. This is the option used in C-ALLOUT since it provides more accurate selection of nodes, even if a bit more expensive;
- **minoccup**: choose the qualifying node which has the current minimum occupancy.

When a full node is selected to be the node to insert the new object, the node has to be split, by dividing all of its members in two different new nodes. The *SplittingMethod* algorithm is used to split a node and it has three options for splitting:

- **random**: randomly select any two objects in the node to be the representatives (centers) of the two new nodes. All of the remaining objects will be distributed between the two nodes. Each object goes to the node for which it has the minimum distance to its center.
- **minMax**: all pairs of objects in the full node are considered potential representatives for the two new nodes. The pair that minimizes the covering radius from the pair dual center to the most distant of the remaining objects is chosen. This method has the best query performance with a built tree, but it is also the slowest for building the tree, due to its $\Theta(c^3)$ complexity (TRAINA *et al.*, 2000). It is the default splitting method for the M-tree (CIACCIA; PATELLA; ZEZULA, 1997; CIACCIA *et al.*, 1997), another metric tree in which the Slim-tree is based upon.
- **MST**: a Minimal Spanning Tree (MST) (KRUSKAL, 1956) is built using all objects in the node to be split. The MST's nodes are the objects and the MST's edges are the distances calculated between the nodes. The longest edge from the MST is cut. The resulting two connected components become the two new Slim-tree nodes. The representative of the two new nodes is the object that minimizes the covering radius. Figure 7 illustrates the MST method. This method was proposed along with the Slim-tree and can build the tree at a much faster rate ($O(c^2 \log c)$), with minimal negative effect on query performance (TRAINA *et al.*, 2000). This is the option used in C-ALLOUT.

2.4.2.2 Slim-tree's Optimization Methods

The insertion of new objects through the building methods is not perfect and can lead to an unbalanced Slim-tree. One of the issues that can happen during object insertions is the overlapping between two nodes. This happens when the radius of two given nodes in the same level of the tree can reach the same object. In vector spaces, the overlap between two nodes is the volume of their intersection. However, in metric spaces there is no definition of volume.

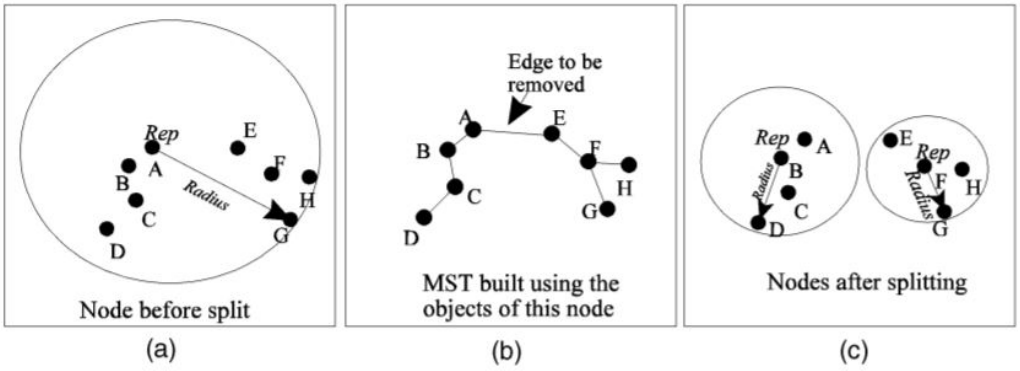


Figure 7 – MST algorithm revised for the Slim-tree. Adapted from Traina *et al.* (2002).

Therefore, Slim-tree's authors proposed to calculate the overlap between two nodes as the amount of objects covered by both region boundaries. This definition of node overlapping for metric trees allows optimization techniques used in vector spaces to be applied on metric trees (TRAINA *et al.*, 2000).

In Slim-tree's original proposal, when querying for any object, the quickest way to perform the query would be to access the minimum number of disk pages necessary. This is due to the fact that retrieving disk pages is expensive. After all, this notion also applies when dealing only with main memory since accessing less nodes also means less distance calculations to check where to go next. When there are nodes overlapping with each other and covering the same space related to the queried object, all of the overlapping nodes need to be examined to successfully locate the object. Consequently, the less amount of overlapping in a tree, the better.

To calculate the amount of overlapping in a tree, the *FatFactor* metric was developed:

$$FatFactor(\mathscr{T}) = \frac{I_C - H \times N}{N} \times \frac{1}{(M - H)}.$$

Where $\mathscr{T}$ is a Slim-tree; H is the height of the tree; $M \geq 1$ is the number of nodes in the tree; N is the number of objects in the tree, and; I_C is the total number of disk access needed to perform a point query on all of the objects in the tree. A point query is a range query with radius zero, that returns one, and only one, node from the level where the query is performed. The most optimized Slim-tree allows that only one node for each tree level is returned to find any queried object. The worst optimized Slim-tree requires that all nodes for every tree level are returned to find any queried object. Encapsulating the formula, a lower *FatFactor* means a more optimized Slim-tree.

To minimize the *FatFactor* of a Slim-tree, the *Slim-down* algorithm was proposed. Generally, it reduces the radius of the nodes by moving their most distant objects to other nodes that can cover those objects. As it can be seen in Figure 8, when instance c is transferred from node i to node j, the intersection region between the two nodes is reduced, thus minimizing I_C (node overlapping).

The slim-down procedure is overviewed in the self-explanatory Algorithm 1. The algorithm can be applied in many situations at many stages of the tree construction, some of them

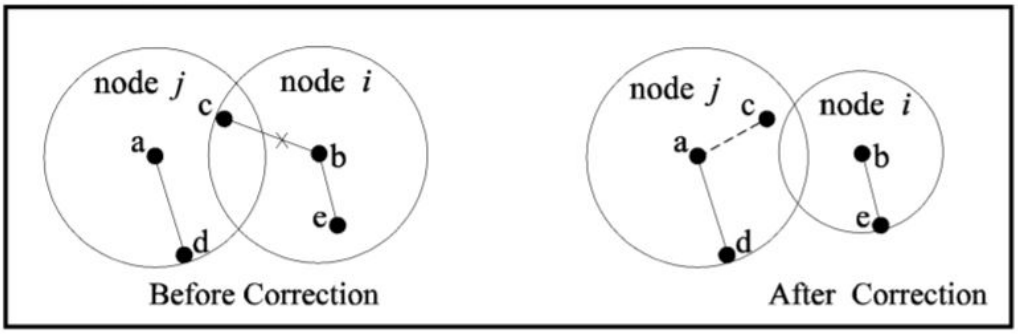


Figure 8 – Slim-down algorithm visualized. Adapted from Traina *et al.* (2002).

are:

- When the building phase of the tree is finished, across all leaf nodes;
- When the building phase of the tree is finished, across all index nodes, with needed adaptations;
- When one of the direct descendants of a node splits, apply it to that node;
- When one object needs to be inserted into a node that is full, to relocate the farthest object instead of splitting the node.

Algorithm 1: Slim-down algorithm. Adapted from Traina *et al.* (2002).

Input: T: the built metric tree to be optimized, h: level of the tree to be optimized
Output: T': an optimized metric tree

```
1  for each node i in the level h of the tree T do
2      Find the farthest object c from the representative b
3      Find the sibling node j of i that also covers object c
4      if node j exists and it is not full then
5          Remove object c from node i
6          Insert object c into node j
7      end
8      if node i is not empty then
9          Correct radius of node i
10     else
11         Delete node i
12     end
13 end
14 if any object moves from one node to another then
15     Restart from line 1
16 end
```

The distance calculation function is one of the main aspects of the Slim-tree and it can be defined during its instantiation. The distance function must be metric and it must return a numeric value that indicates the distance between two compared objects.

2.5 Final Considerations

This chapter discussed the outliers and outlier types, providing informal definitions for both while also overviewing the literature which gave meaning and usage to outlier analysis. The most popular outlier detectors were briefly presented while discussing how they participate in the literature. By presenting three different outlier detection strategies in groups, the outlier detection methods were contextualized so that we could also present the idea for the outlier annotation problem and its formal definitions.

The metric space properties were introduced and through that we were able to define some elements of Slim-tree. The inner workings of the tree structure were explained so that the background for our proposed C-ALLOUT could be solidified. Some important insights on the reasoning behind Slim-tree choices and also our choices for this work were given as a preliminary step towards presenting C-ALLOUT and its related works.

In the next chapter, the related work is presented giving an overview of what has been achieved on both outlier detection and annotation. We thoroughly review specific related works as they are not only comparable to our proposal, but also selected as our competitors in our experimental evaluation.

CHAPTER

3

RELATED WORK

There is a large literature behind outlier detection where multiple algorithms and methodologies are presented to solve numerous real-world problems (AGGARWAL, 2015). The detectors use a vast range of different measures to find outliers. These measure are usually based on analyzing distances, densities, angles, depths, subspaces, clusterings and statistical distributions in the data (HAN; KAMBER; PEI, 2012).

It is important to remark that, our proposal is not only to detect, but also to annotate outliers. The outliers annotations can separate outlier in three types, as discussed before: global, local and collective. We can find in the literature different techniques that search for a specific type of outlier. Anomaly points far from the majority of the data are looked for by most outlier detectors (KNORR; NG, 1998; KRIEGEL; SCHUBERT; ZIMEK, 2008; RAMASWAMY; RASTOGI; SHIM, 2000; SCHöLKOPF *et al.*, 1999) and are known as global outliers. There are also local outlier detectors, looking for those that deviate not from the majority of the data, but from the majority of their neighborhood (BREUNIG *et al.*, 2000; DANG *et al.*, 2013; KRIEGEL *et al.*, 2009). Another category of outlier mining algorithms has those that hunt for collective anomalies, the outlier-filled groups that are isolated from the large clusters of inliers (HAN; KAMBER; PEI, 2012; NIAN *et al.*, 2016).

Our proposed method C-ALLOUT fills the annotation gap in the literature. In the next section we review important outlier detectors, delivering more details on the ones closely related to our project (and present in our experiments): SIF, KNNOut, LOF, OCSVM and SRA.

3.1 Outlier Detection Techniques

In many occasions outlier detectors can find multiple types of outliers without separating them. That is the case for kNN (RAMASWAMY; RASTOGI; SHIM, 2000) and Isolation Forest (LIU; TING; ZHOU, 2008), both capable of attributing higher outlierness scores for local

outliers in comparison with inliers. kNN can also find collective outliers if the value provided for parameter k is larger than the number of points in a collective outlier micro-cluster. LOF (BREUNIG *et al.*, 2000), even if proposed for local outlier detection, also assigns large scores for global outliers. Because of this, all of these cited methods are present in our experiments where we evaluate overall outlier detection capabilities. Likewise, the literature also provides some methods developed to detect, but not annotate, all three outlier types (BANDARAGODA *et al.*, 2018; MOONESINGHE; TAN, 2006; PAPADIMITRIOU *et al.*, 2003).

3.1.1 Clustering

Yin and Wang (2016) proposed the collapsed Gibbs Sampling algorithm for the Dirichlet Process Multinomial Mixture (GSDPMM) clustering model taking the assumption from Dirichlet Process Multinomial Mixture (DPMM) that there is an infinite amount of possible clusters but only a finite amount is needed to describe a document. From this assumption, the model does not need to have a exogenous parameter to determine the number of clusters needed to describe a dataset or a document. Then, the number of clusters dynamically grows as more instances are received as input.

The GSDPMM model basically can calculate the probability of a document (instance) to choose an existing cluster or establish a new one, given the similarity of its terms with the ones from other documents of the existing clusters. The model works to support two predefined rules based on two objectives defined in Rosenberg and Hirschberg (2007):

- Rule 1: "Choose a cluster with more documents" (YIN; WANG, 2016), as the more documents a cluster has, the more terms in common it will have with the document chosen. This is in conformance with the completeness objective, "all members of a ground true group are assigned to the same cluster" (ROSENBERG; HIRSCHBERG, 2007).

- Rule 2: "Choose a cluster whose documents share more words with the current document" (YIN; WANG, 2016), as the clusters with more terms related to a document are more likely to be chosen by the document for having more in common. This conforms with the homogeneity objective, "each cluster contains only members of a single ground true group" (ROSENBERG; HIRSCHBERG, 2007).

Initially, all the documents are assigned to the same cluster. In each iteration of GSDPMM, every document from the corpus gets the chance to choose another cluster or to create a new one. The model assumes that outlier documents are more likely to choose new clusters than existing ones, as they do not share much terms with the existing clusters. Other documents will also likely not choose to compose the outlier cluster, as they also do not share much terms with the outlier cluster. Most empty clusters indicate outliers.

3.1.2 Neural Networks

Most neural networks are limited to working with numerical data with a predetermined dimensionality. However, there are some networks that are able to extract numerical data from image, graph or text data.

Observing the extractor potential in Convolutional Neural Networks (CNN), Gorokhov, Petrovskiy and Mashechkin (2017) developed a detection method for outliers which is based on the combination of convolutional filters at first with clustering of Radial Basis Function (RBF) (PARK; SANDBERG, 1991) later. Using a CNN as a feature extractor, it is possible to later group these features in a multidimensional space. The grouping is performed with the addition of RBFs in the penultimate layer of the proposed neural network. Then, the last layer is able to determine if the instance inserted in the network is part of any of the groups created by the RBFs. If not, the instance is considered outlier.

3.1.3 Isolation Forest

Going on a different direction than most outlier detection algorithms, Isolation Forest, also known as iForest (LIU; TING; ZHOU, 2008), is an algorithm that tries to detect outliers using knowledge obtained by a group of tree structures, each one looking at different subsets of data. The main concept behind iForest is that, in a feature space, if random cuts are done using a single feature, the probability of outliers being isolated by a cut in space is higher than that of inliers. This notion comes from the fact that outliers are already isolated in space, so a sequence of random cuts will detach them from inlier clusters sooner.

To make random cuts happen, binary trees are used. Every node of a tree is split in two subtrees using a randomly selected attribute from the data. By having the minimum and maximum values for this attribute, it is possible to define a split value where all points in the data can be separated in two groups, each one to be passed to each subtree. This is a summary of what happens in Algorithm 2. Note that, eventually, every point is going to be isolated, but outliers are isolated in a subtree earlier. That happens because outliers have very distinct feature values. The other main concept to note about Algorithm 2 is that nodes of the tree are split only until one of three criteria is reached:

- The node is in a tree level corresponding to tree height limit l;
- The number of points in the node is below 2;
- All points in the node are equal.

Tree height limit l is calculated by iForest using hyperparameter ψ, provided by the user. ψ has a default value of 256 recommended by the authors. ψ is used as in Line 2 from Algorithm 3, the pseudocode describing the higher level of iForest. The empirical outcome noted

Algorithm 2: $iTree(X, e, l)$. Adapted from Liu, Ting and Zhou (2008).

Input: Input dataset $X \in \mathbb{R}^{n \times d}$, current tree height e, height limit l
Output: an iTree

if $e \geq l$ *or* $|X| \leq 1$ **then**
  return $exNode\{Size = |X|\}$
else
  let Q be the list of attributes in X
  randomly select an attribute $q \in Q$
  randomly select a split point p from *min* and *max* values of attribute q in X
  $X_l = filter(X, q < p)$
  $X_r = filter(X, q \geq p)$
  return $inNode\{Left = iTree(X_l, e+1, l),$
    $Right = iTree(X_r, e+1, l),$
    $SplitAtt = q,$
    $SplitValue = p\}$
end

by the authors to set a tree height limit is that, usually, outliers are isolated in the very first node splits. Thus, it is unnecessary to keep splitting nodes after a certain point, when outliers are already isolated. ψ is also the number of samples taken from the dataset and passed to each tree in the forest.

Algorithm 3: $iForest(X, t, \psi)$. Adapted from Liu, Ting and Zhou (2008).

Input: Input dataset $X \in \mathbb{R}^{n \times d}$, number of trees t, subsampling size ψ
Output: a set of t *iTrees*

Initialize *Forest*
$l = ceiling(\log_2 \psi)$
for $i = 1$ *to* t **do**
  $X' = sample(X, \psi)$
  $Forest = Forest \cup iTree(X', 0, l)$
end
return *Forest*

Another user-defined hyperparameter is t, the number of trees in the forest. The main motivation to use a collection of trees instead of just one is that each tree is constructed using different subsets (of ψ size) of the dataset. Therefore, overfitting can be avoided as each tree is learning different characteristics about the data. Conceptually, with more trees it is possible to get better knowledge extraction, however, each tree adds more cost to iForest. Experiments done by the authors suggest that at around $t = 100$, the number of trees is not worth increasing anymore.

All the knowledge obtained by each tree is joined in the outlierness score of the points. The score comes from the average path length to find the point in every tree in the forest. As shown in Algorithm 4, the path length of a point is calculated by traversing the tree until its

last level is reached (Line 1 to 3). To traverse the tree, the algorithm chooses the path by using the current node information on which attribute and value were used to split it. In Line 2 of Algorithm 4 we can see that the value returned for path length is $e + c(T.size)$. e is the path length until the last level of the tree is reached and $c(T.size)$ is an estimation of how many edges there would be if the tree was expanded fully:

$$c(n) = 2H(n-1) - (2(n-1)/n)$$

Where $H(i)$ is the harmonic number estimated by $\ln(i) + e$.

Algorithm 4: $PathLength(x, T, e)$. Adapted from Liu, Ting and Zhou (2008).

Input: Instance $x \in X$, an iTree T, current path length e; to be initialized to zero when first called
Output: path length of x

```
1 if T is an external node then
2 |   return e + c(T.size)
3 end
4 a = T.splitAtt
5 if x_a < T.splitValue then
6 |   return PathLength(x, T.left, e + 1)
7 else
8 |   return PathLength(x, T.right, e + 1)
9 end
```

3.1.4 Similarity Isolation Forest

Even though Isolation Forest has become increasingly popular in machine learning, it is still not based on pairwise similarities and need the actual features to cut the feature space. As stated in Chapter 1, this is problematic if the features are not available. Having this setting in mind, Sathe and Aggarwal (2017) proposed an adaptation to the Random Forests algorithm (BREIMAN, 2001). In this adaptation, the proposed Similarity Forest works with similarities instead of data features.

The new objective is achieved by doing a new type of split where, in the current subsample of points, two are randomly picked to be used as pivots, p and q. Then, using these pivots, all other points in the subsample are projected into the imaginary line between p and q. It is possible to use the Gini-index, or any other entropy calculation method, to find a cut-point in the line that will best divide the points into two groups, minimizing entropy. Points closer to q will belong to subtree q and the remainder go to subtree p. As it can be noticed, to analyze entropies labels are needed and Similarity Forest is indeed a supervised classification algorithm.

Observing the new possibility brought by Similarity Forests, Czekalski and Morzy (2021) decided to unify the Similarity Forest idea with iForest. Now, two fronts can be tackled

simultaneously: bringing the Similarity Forest concepts to the realm of unsupervised learning and making it possible for iForest to work with similarities only.

The combination of Similarity and Isolation Forest is conveniently called Similarity Isolation Forest (SIF). We can summarize the needed changes by taking Algorithm 5 from Line 4 downwards. Note that, Algorithm 5 describes one approach where instead of using the dot product to get projections of points to the line between p and q, the Euclidean distance is used. This is not only supported by the algorithm, but is also aligned with C-ALLOUT that also uses the Euclidean distance. With the two algorithms using the same distance measure, the comparison between them in Chapter 5 becomes fair. In Line 6 of Algorithm 5, the projection of points in the line connecting p and q is obtained by subtracting the vectors of Euclidean distances. In the similarities vector, points closer to p have negative value; points closer to q have positive value. Nonetheless, to keep the random principle of Isolation Forest, the split point for the node is selected at random (Line 8).

The main advantages of iForest that are inherited by SIF are its time complexities of $O(\log n)$ to reach any point in a tree, and $O(n)$ to build a tree, and its ability to have easily defined user hyperparameters, since the algorithm performance is not much affected by changes in them.

Algorithm 5: $sifTree(X, e, l)$

Input: Input dataset $X \in \mathbb{R}^{n \times d}$, current tree height e, height limit l
Output: an iTree

1 **if** $e \geq l$ *or* $|X| \leq 1$ **then**
2 | return $exNode\{Size = |X|\}$
3 **else**
4 | let p and q be two random points in X
5 | p_{euc} = Euclidean distance between all points in X and p
6 | q_{euc} = Euclidean distance between all points in X and q
7 | $similarities = q_{euc} - p_{euc}$
8 | $splitValue = random(min(similarities), max(similarities))$
9 | $X_l = filter(X, similarities \leq splitValue)$
10 | $X_r = filter(X, similarities > splitValue)$
11 | return $inNode\{Left = sifTree(X_l, e+1, l),$
12 | $Right = sifTree(X_r, e+1, l),$
13 | $SplitAtt = q,$
14 | $SplitValue = p\}$
15 **end**

3.1.5 k-Nearest Neighbors

Being a very popular method for classification tasks, k-Nearest Neighbors (kNN) (FIX; HODGES, 1989) has gone through many adaptations to work with many different applications. One of these applications is outlier detection, where the common clustering of the points in a training phase is not needed. Since outlier detection does not necessarily need to classify data

between inlier and outlier, the most successful adaptation of kNN to work with outliers has been a ranking-based one. In this version, KNNOutlier (RAMASWAMY; RASTOGI; SHIM, 2000) returns a single ranking having outlierness scores. In this case, the score of a point is defined as the distance between the point and its k^{th} neighbor, considering the entire dataset. The greater the distance to the k^{th} neighbor, the more outlier is the point. Algorithm 6 shows the overall steps taken by KNNOutlier.

Algorithm 6: Brute-force kNNOutlier

Input: Input dataset $X \in \mathbb{R}^{n \times d}$, nearest neighbor k to use to set the scores, distance function d
Output: Ranking vector $\mathbf{r} \in \mathbb{R}^n$ with larger values being more outlier

```
r = ∅
for each point x in X do
    for each point y in X do
        x_n.append(d(x,y))
    end
    Sort x_n in ascending order
    k_dist = x_n[k]
    r.append(k_dist)
end
Return r
```

As it can be seen, KNNOutlier needs and depends heavily on a distance function. Defining a distance function to be used in machine learning algorithms is not a hard task and any distance function can be used with the brute force version of KNNOutlier shown in Algorithm 6. However, brute-force KNNOutlier has O(n^2) time complexity. For that reason, many different data structures have been proposed to speed up KNNOutlier. Amongst the examples of such structures, we can name the Ball Tree (or Metric Tree) (OMOHUNDRO, 1989) and the k-d Tree (BENTLEY, 1975). Some of these structures may have constraints on their distance functions, for instance, Ball Tree only works with metric distance functions. Although these data structures guarantee an average time complexity of $O(n \log n)$ to measure the outlierness score of all points, they are still exact methods. Hence, in the worst case, time complexity still is $O(n^2)$. Finally, it is also important to note that these data structures have their own set of hyperparameters as well.

3.1.6 Local Outlier Factor

Local Outlier Factor (LOF) (BREUNIG *et al.*, 2000) was proposed to improve on KNNOutlier shortcomings, while maintaining its principles. One of the major limitations of KNNOutlier is its inability to work with densities. This limitation could possibly make KNNOutlier ignore outliers in dense neighborhoods, as the points belonging to more sparse regions of the vector space would eventually receive larger outlierness scores. So, there should be a normalizing factor to make up for differences in neighborhood densities. LOF's concept is to

find how reachable a point is in comparison to the rest of the dataset, while also considering the densities involved.

To calculate the Local Reachability Distance (LRD) of a point, the algorithm must first find the pairwise Reachability Distance (RD) between the point and all of its k neighbors. RD is defined as:

$$RD(x_i, x_j) = max(k\_distance(x_j), d(x_i, x_j))$$

With X being the analyzed dataset, $x_i, x_j \in X$; $k\_distance$ is the distance to the k^{th} neighbor and; d is the distance function defined by the user.

The maximum between the two is used as the RD to statistically minimize the statistical fluctuations for distances in the local neighborhood of a point (BREUNIG *et al.*, 2000). Therefore, by substituting the distance between the point analyzed and its neighbors by its $k\_distance$, a smoothing effect is applied to the closest distances.

LRD is then defined using RD:

$$LRD_k(x_i) = \frac{1}{\sum_{x_j \in N_k(x_i)} \frac{RD(x_i, x_j)}{\|N_k(x_i)\|}}$$

With X being the analyzed dataset, $x_i, x_j \in X$ and; $N_k(x_i)$ being the neighborhood of x_i.

We can interpret that LRD is the inverse of the average RD between a point and its k neighbors. Then, the larger the average RD, the smaller the density around a point. A low LRD implies low density and also that the point is far from the closest dense group, i.e., inlier cluster.

Lastly, the final LOF measure is defined as:

$$LOF_k(x_i) = \frac{\sum_{x_j \in N_k(x_i)} LRD_k(x_j)}{\|N_k(x_i)\|} \times \frac{1}{LRD_k(x_i)}$$

We can say that the LOF metric is a comparison between the average LRD of the k neighbors of a point and the LRD of the point itself. Thus, if LOF is close to 1, that means that the density of a point is roughly the same as the density of its neighbors. But, if the LRD value is smaller for the point, its low density will throw the LOF measure high. This is, overall, a local outlier detection approach, where the neighboring instances define the outlierness score of a point. The LOF method is presented in Algorithm 7.

3.1.7 Ensembles

fastText (GRAVE *et al.*, 2017) is a popular algorithm for extracting semantic features from texts with the support of deep learning. With the text data represented in feature vectors, Walkowiak, Datko and Maciejewski (2019) proposed to ignore the linear model based on a soft-max classifier, originally expected to be used with fastText. In its place, the authors argue

Algorithm 7: Local Outlier Factor

Input: Input dataset $X \in \mathbb{R}^{n \times d}$, nearest neighbor k to use to set the scores
Output: Ranking vector $\mathbf{r} \in \mathbb{R}^n$ with local density factors

$\mathbf{r} = N_k = LRD_k = \emptyset$
for *each point* x_i *in* X **do**
  $N_k.append(k\_distance(x_i, k))$
  $LRD_k.append(local\_reach\_dist(x_i, N_k(x_i), k))$
end
for *each point* x_i *in* X **do**
  $lof = 0$
  for *each point* x_j *in* $N_k(x_i)$ **do**
    $lof = lof + LRD_k(x_j)$
  end
  $lof = \frac{lof}{\|N_k(x_i)\|} \times \frac{1}{LRD_k(x_i)}$
  $\mathbf{r}.append(lof)$
end
Return $\mathbf{r}$

that other classifiers, such as the Multilayer Perceptron (MLP) can perform better in datasets having outliers. After the training and tuning of the MLP classifier, LOF (BREUNIG *et al.*, 2000) comes as the third piece of the ensemble to validate the neural network's output.

As thoroughly explained previously, the LOF method has an homonym metric used to check the density of the data in a region. If an instance has low density in its region, it is considered an outlier. A LOF clustering structure is instantiated using the MLP training data. If the MLP labels a test instance as outlier, but the same instance get in a cluster with a different label when input to LOF, the outlier status is removed from the instance. In experiments, even though not increasing much the MLP scores for the original dataset, when outliers participate in another dataset, the LOF correction boosts the MLP model performance.

3.1.8 One-Class Support Vector Machines

As a popular classification method, Support Vector Machine (SVM)s (CORTES; VAPNIK, 1995) has gone through many variations to attend many different purposes. SVMs have the notorious capacity of drawing a hyperplane capable of separating linearly inseparable points from different classes. This is achieved through the usage of kernel functions that can project the points into a higher dimension. In high-dimensional feature spaces points have a greater chance of being linearly separable. After a hyperplane is created in this high-dimensional feature space, it can be brought over to the original low-dimensional feature space. In the end, the new complex shape originated from the hyperplane keeps separating the points even in the low-dimensional feature space. The hyperplane can be defined by a vector ω and an offset to the origin b:

$$\omega^T x + b = 0$$

With F being the feature space, x is a data point from $X \in F$, $\omega \in F$ and $b \in \mathbb{R}$.

For the One-Class Support Vector Machine (OCSVM) (SCHöLKOPF *et al.*, 1999) there are only two classes being considered, one having outlier points and one having inlier points. We can interpret that the hyperplane sets a margin dividing both classes. The hyperplane's main purpose is to maximize the margin separating the points in each class. The margin is limited by the points from both classes that are the closest to the hyperplane. To minimize the overfitting that can happen with OCSVM, additional slack variables ξ can be added to allow some points from both classes to lie within the margin. The points within the margin become the training errors.

In the OCSVM version described by Schölkopf *et al.* (1999) the hyperplane separates all training points from the origin while maximizing the distance between the hyperplane and the origin. Hence, we have two regions with outliers going to be located near the origin and inliers on the other side. The classifier can be described by the quadratic programming minimization function:

$$\min_{\omega,\xi,\rho} \frac{1}{2}\|\omega\|^2 - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i$$

subject to:

$$(\omega \cdot \phi(x_i)) \geq \rho - \xi_i \qquad \text{for all } i = 1,...,n$$
$$\xi_i \geq 0 \qquad \text{for all } i = 1,...,n$$

Where ω are the weights of the classifier; $\phi(x_i)$ is a kernel function taking point x_i to a higher dimension and; ρ is an additional parameter to bound the increase in slack variables in the sum.

Most importantly, ν is a user-defined parameter which adds some smoothness to the margin of the hyperplane. Therefore, ν determines the trade-off between the margin maximization and the number of training points allowed to be within the margin. Because of this, ν sets an upper bound on the fraction of outliers (training points viewed as wrongly placed) and, at the same time, a lower bound on the number of training points used as support vectors. One can interpret ν as a penalty for the OCSVM. The larger it is, the more penalty the OCSVM will receive for a mistake. Thus, a large ν creates a narrower margin that results in less penalties and few support vectors used.

The support vectors are the points used to define the margin itself, being the points in the two classes which are the closest to the hyperplane. Support vectors are obtained when the quadratic programming problem is solved using Lagrange multipliers. Lagrange multipliers α_i "support" the machine by contributing with weights for OCSVM decision function which classifies the points:

$$f(x) = sgn((\omega \cdot \phi(x_i)) - \rho) = sgn(\sum_{i=1}^{n}\alpha_i K(x,x_i) - \rho)$$

Where sgn is the sign function; $K(x,x_i)$ is the kernel function using support vector x to take point x_i to a higher dimensional space and; n is the size of the training data.

The kernel function $K(x,x_i) = \phi(x)^T \phi(x_i)$ can be one of many different functions, since the feature space F can be of unlimited dimensions. Anyway, complex kernel functions are avoided since they have their own set of parameters and can increase the complexity of OCSVM. Usual kernel functions are: linear, polynomial, sigmoidal and Gaussian Radial Basis Function. The kernel function needs to return a similarity measure between the two points passed to it. Since we are using the Euclidean distance for all methods in our experiments, to obtain reasonable comparisons, the most related kernel function to consider would be the linear one:

$$K(x,x_i) = x^T \cdot x_i$$

In the end, OCSVM will separate two classes, the least populated class is the one holding outliers. In our experiments we noticed that OCSVM is very sensitive to changes in its hyperparameters. Although it was not under our evaluation, the kernel function choice is a tricky one (KEERTHI; LIN, 2003). OCSVM is also known for not having good performance on large datasets and for having issues with noisy data or overlap between classes.

3.1.9 Robust Deep Autoencoders

Robust Deep Autoencoder (RAE) (ZHOU; PAFFENROTH, 2017) was proposed to enhance the robustness of autoencoders when dealing with noisy or anomalous data. As proposed, RAE can be used as a more robust classifier or as an outlier detector. Just as most autoencoders, the method distinguishes outliers from inliers using the reconstruction error of the input. In the encoding phase, the usual autoencoder learns the features of the data so that in the decoding phase it can reconstruct the data using only the features learned. Points with large reconstruction errors have high outlierness scores. RAE, on the other hand, takes a step beyond and use the methodology of Robust Principal Component Anaylsis (RPCA) to define that the input to be encoded needs to first be filtered of outliers. Outliers cannot be represented in reduced dimensions by the encoder because they naturally fall out of the expected patterns.

RPCA is incorporated into RAE with the addition of a filter layer to the autoencoder network. This new layer acts similarly to RPCA and the input matrix X is factored:

$$X = L_D + S$$

where L_D is the part of the input matrix that is clean of anomalies and is well-represented by the encoding layers of the autoencoder and S is the matrix having anomalies.

To achieve the best factorization, the authors proposed two optimization problems for dealing with datasets composed of instances as lines and features as columns. One is able to find outlying features and the other one, more related to our work, can find outlying instances:

$$\min_{\theta,S} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda \left\|S^T\right\|_{2,1}$$

subject to:

$$X - L_D - S = 0$$

where D_θ and E_θ are, respectively, the decoder and encoder associated with the autoencoder; $\|\cdot\|_2$ and $\|\cdot\|_{2,1}$ are, respectively, the $L2$ and $L2,1$ norms; S^T is the transposed of S and λ is a hyperparameter used for tuning the sparsity of S.

The $L2,1$ norm is used as a regularization method for S and λ works by inducing the amount of instances to be isolated in S. The smaller λ is, the less objects are considered outliers and moved to S. This sensitivity hyperparameter allows to balance between false-positives and false-negatives, since larger values for λ makes the algorithm more sensible and set more points as outliers. The L_D and S matrices are updated through many iterations until a defined threshold or number of iterations is reached. Following the literature for RPCA, RAE optimizes both matrices individually, as separated problems. L_D is optimized using the backpropagation algorithm in the autoencoder and S is optimized using proximal methods for the norm used (the norm used varies according to the application of the method). To find the outliers, a $L1$ norm must be executed over the final S matrix.

3.1.10 Indexing Data Structures

In general, outlier detection methods have limitations when applied to large databases, obtaining poor performance when applied to real databases. In addition, the methods are mostly effective in a specific context, but have notable drops in accuracy when tested on datasets that differ from the baseline for which they were designed. Considering mainly these two points, Junior and Cordeiro (2019a) found in data indexing structures a way to make detection more scalable and generalized. The presented angle-based outlier detection algorithm, MetricABOD, uses the Slim-tree (TRAINA *et al.*, 2000) indexing structure as means for indexing analyzed datasets.

Unlike traditional angle-based methods, which check the angle variation of all elements, MetricABOD checks only instances that represent a group of elements. These representative instances are acquired from the tree structure of the Slim-tree, which sets representative objects during its construction. This selection of elements significantly reduces the amount of angle calculations required. Even if cutting down a lot of comparisons between instances, the method is sometimes able to achieve better performance, in matters of runtime and also accuracy, than other methods for outlier detection popular in the literature, not based on indexing structures.

The way an outlier candidate P is compared with all the representative objects (A, B, C and D) from the Slim-tree structure can be seen in Figure 9. The other points in the figure represent unimportant instances of data and are marked with a red cross to emphasize that they are being ignored in the analysis.

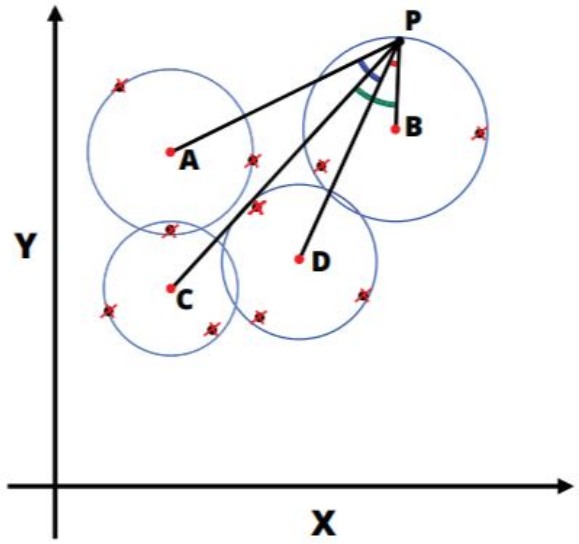


Figure 9 – Angle-based outlier detection using the Slim-tree representative objects. Adapted from Junior and Cordeiro (2019a).

3.2 Outlier Annotation Techniques

As our closest related work, Spectral Ranking for Anomalies (SRA) (NIAN *et al.*, 2016) was proposed as an algorithm capable of detecting outliers and also flagging outlier types between scattered and clustered. The technique is, to the extent of our knowledge, the only algorithm capable of doing any sort of outlier type categorization. The algorithm separates the feature space in two partitions, scoring the points considering their similarity with the centers of the partitions. SRA only seeks for two types of outliers, scattered (or point outliers) and clustered (or collective outliers). The definition given for collective outliers in its original paper is similar to ours: groups of anomalies that are not only close to each other but also numerous enough to be possibly misinterpreted as inlier clusters. Point outliers, on the other hand, are outliers not grouped together. Point outliers enclose both our definitions for global and local outliers. In this manner, we can say that SRA is able to detect all three types of outliers we have defined.

Although being able to detect both local and global outliers, the method is unable to distinguish them from each other. The method only returns a single ranking with outlierness scores and a flag telling if the algorithm considers the outliers to be point or collective ones. This points to another limitation of the algorithm, it is not able to annotate multiple types of outliers at once. It can only tag all outliers as being point or collective. Even with limited capacities on the outlier annotation task, we consider the algorithm to be our only reasonable competitor in this endeavor. Algorithm 8 shows the general steps performed by SRA.

In the first line of the algorithm, a Laplacian matrix L (in fact, the symmetric normalized Laplacian) is formed using similarity matrix W. L is a simplified graph representation of the dataset that can be used to find bi-partitions for the graph. This is the usual approach for spectral clustering. So, in the second line, we can see the first non-principal eigenvector (the second

Algorithm 8: Spectral Ranking for Anomalies. Adapted from Nian *et al.* (2016).

Input: Similarity matrix $W \in \mathbb{R}^{n \times n}$, anomaly ratio upper bound χ
Output: Ranking vector $\mathbf{f}^* \in \mathbb{R}^n$ with larger values being more outlier, flag for outlier type in ranking *mFLAG*

Form Laplacian matrix $L = I - D^{-1/2} W D^{-1/2}$
Compute $\mathbf{g}_1^*$ (the first non-principal eigenvector for L)
Compute $\mathbf{z}^* = D^{1/2}\mathbf{g}_1^*$
Let $C_+ = \{i : \mathbf{z}_i^* \geq 0\}$ and $C_- = \{i : \mathbf{z}_i^* < 0\}$
if $min\{\frac{|C_+|}{n}, \frac{|C_-|}{n}\} \geq \chi$ **then**
    $mFLAG = 1$, $\mathbf{f}^* = max(|\mathbf{z}^*|) - |\mathbf{z}^*|$ % ranking point outliers
else if $|C_+| > |C_-|$ **then**
    $mFLAG = 0$, $\mathbf{f}^* = -\mathbf{z}^*$ % ranking collective outliers
else
    $mFLAG = 0$, $\mathbf{f}^* = \mathbf{z}^*$ % ranking collective outliers
end

eigenvector sorted by the eigenvalues magnitudes) for L being calculated as $\mathbf{g}_1^*$. With $\mathbf{g}_1^*$ it is possible to find a cut in the graph representation which will create the bi-partitioned graph. With this, we can find two clusters, one for each partition, one on each side of $\mathbf{g}_1^*$.

However, instead of assigning points to both clusters that were determined through the bi-partition, SRA intends to rank all points using eigenvector $\mathbf{g}_1^*$. $\mathbf{z}^*$ gives us, in a sense, the similarity to $\mathbf{g}_1^*$. Using this similarity measure, we can reach the two clusters C_+ and C_-, where each one is going to have the points that belong to each partition. $\mathbf{z}^*$ is used as the outlierness score for the points and, therefore, the scores are the similarity to the clusters previously obtained. The percentage of anomalies χ is used as a threshold for deciding what type of outlier is being dealt with. If both clusters percentage of points are larger or equal than χ, then both clusters are considered inlier clusters, and outliers are point outliers. If one of the clusters percentage of points is not larger than χ, the smallest one is an outlier cluster, or a collective outlier group, and the ranking is adapted as needed with outliers being collective outliers. The previous condition is what determines the value of *mFLAG*, that tells which type of outlier was found.

3.3 Final Considerations

In this chapter we have done a general review over the outlier detection literature. The review covered many detectors that work with similarities, subspaces, neural networks, clustering and many other techniques. Nonetheless, in this review, we focused on these most related works, i.e., outlier detectors able to work with distances, or similarities, alone and one algorithm capable of doing a very limited outlier annotation. We thoroughly explained SIF, kNN, LOF, OCSVM, RAE and SRA, and we provided algorithms for most of them. Going further to the experimental analysis, these six algorithms appear again as we put our method C-ALLOUT to the test.

Table 1 – Summary on the presented features in C-ALLOUT and its most related works.

Algorithm	Deterministic	Approach	Outlier annotation	Data indexing	Default HP	Scalable
kNN	✓	Nearest Neighbors	No	Possible	✗	Possible
OCSVM	✓	Hyperplane & kernel function	No	No	✗	No
LOF	✓	Nearest Neighbors & density	No	Possible	✗	Possible
SIF	✗	Random subspaces	No	Naturally	✓	Naturally
RAE	✗	Autoencoder	No	No	✗	Possible
SRA	✓	Clustering	Very limited	No	✗	No
C-ALLOUT	✓	Indexing-tree distances	Naturally	Naturally	✓	Naturally

Table 1 presents a summary for our most related work, it contains descriptive and desirable features on columns. Deterministic are methods that always return the same response given the same input and hyperparameters. Approach suggest the main technique employed during outlier detection. Outlier annotation indicates if the algorithm is able to annotate outlier types. Data indexing points out if the method readily accepts indexing structures. Default HP shows us the methods that have recommended default hyperparameters that can make the method virtually parameter-free. Scalable if the proposed solution is scalable by either its own proposal or by using adequate data structures. Possible indicates if the presence of the feature is implementation dependent and naturally if it is not implementation dependent.

As it can be seen, C-ALLOUT has many desirable properties for outlier detection methods and just like every other method on the table, can work with just pairwise distances. With the exception of SIF, all other proposals have hard to define hyperparameters, meaning that a reasonable amount of prior knowledge on the data is required to select the best configuration. It is also important to note that most anomaly detectors are quite sensitive to hyperparameter choices (GOLDSTEIN; UCHIDA, 2016).

It is unclear how to select the best configuration for hyperparameters for unsupervised learning since ground-truth labels do not exist. Algorithms based on nearest neighbors are the most prevalent techniques, specially when considering the scenario where only pairwise distances are provided, but they are computationally expensive and very sensitive to their hyperparameter settings. We remark that C-ALLOUT is free of these challenges, while also being able to separate outliers by type.

In the next chapter we present C-ALLOUT and provide definitions for its main components and the ones from Slim-tree related to it. We discuss how the method ranks outliers and we also analyze its time and space complexity.

CHAPTER

4

C-ALLOUT: CATCHING AND CALLING OUTLIERS BY TYPE

This section starts by defining outliers and the problem of outlier annotation. Next, there is a general overview highlighting C-ALLOUT's main steps. We also provide definitions for the overall and the type-specific rankings returned by our method. Optimizations for Slim-tree's structure are discussed and, lastly, we analyze time and space complexity.

4.1 Problem Statement

Since, to the best of our knowledge, C-ALLOUT is the pioneer algorithm on outlier annotation, we first formalize the outlier types and the outlier annotation problem.

Definition 4.1.1 (**Outlier**). *Given a point-cloud dataset* $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ *with points mostly generated by one of K distributions* $f_1(\mathbf{x};\Theta_1), \ldots, f_K(\mathbf{x};\Theta_K)$*, where* Θ_j *is the parameter of distribution* $f_j(\mathbf{x};\Theta_j)$*, and* $P_{i,j}$ *is the probability of a point* $\mathbf{x}_i \in \mathbf{X}$ *be generated by a distribution* $f_j(\mathbf{x};\Theta_j)$*; An outlier is a point* $\mathbf{x}_i \in \mathbf{X}$ *whose probability* $P_i = max_j\ P_{i,j}$ *is smaller than the corresponding probabilities of the vast majority of points in* $\mathbf{X}$*.*

Definition 4.1.2 (**Global Outlier**). *A global outlier is an outlier* $\mathbf{x}_i$ *from a dataset* $\mathbf{X}$ *whose probability* P_i *is much smaller than those of the vast majority of points in* $\mathbf{X}$*, thus it arouses the suspicion that* $\mathbf{x}_i$ *is a result of one undefined process. And, there is no point* $\mathbf{x}_j \in \mathbf{X} \setminus \{\mathbf{x}_i\}$ *whose distance* $\mathtt{d}(\mathbf{x}_i, \mathbf{x}_j)$ *to* $\mathbf{x}_i$ *is similar to the distances between most points in* $\mathbf{X}$ *and their closest neighbors.*

Definition 4.1.3 (**Collective Outlier**). *A collective outlier is an outlier* $\mathbf{x}_i$ *from a dataset* $\mathbf{X}$*, such that at least another outlier* $\mathbf{x}_j \in \mathbf{X} \setminus \{\mathbf{x}_i\}$ *has a distance* $\mathtt{d}(\mathbf{x}_i, \mathbf{x}_j)$ *to* $\mathbf{x}_i$ *that is similar to the distances between most points in* $\mathbf{X}$ *and their closest neighbors.*

Definition 4.1.4 (**Local Outlier**). *A local outlier is an outlier* $\mathbf{x}_i$ *from a dataset* $\mathbf{X}$ *whose probability* P_i *is small enough to be suspicious, when compared with the corresponding probabilities of the vast majority of points in* $\mathbf{X}$*, but it is still feasible that* $\mathbf{x}_i$ *is a result of at least one of the distributions* $f_1(\mathbf{x};\Theta_1),\ldots,f_K(\mathbf{x};\Theta_K)$ *seen in* $\mathbf{X}$*. And, there is no point* $\mathbf{x}_j \in \mathbf{X}\setminus\{\mathbf{x}_i\}$ *whose distance* $\mathtt{d}(\mathbf{x}_i,\mathbf{x}_j)$ *to* $\mathbf{x}_i$ *is similar to the distances between most points in* $\mathbf{X}$ *and their closest neighbors.*

Definition 4.1.5 (**Outlier Annotation Problem**). *Given a point-cloud dataset* $\mathbf{X}=\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ *in which only the distances, or similarities,* $\mathtt{d}(\mathbf{x}_i,\mathbf{x}_j)$ *between any two objects* $\mathbf{x}_i$ *and* $\mathbf{x}_j$ *can be accessed; Provide four types of ranking of the points by outlierness degree: 1)* $\mathscr{R}_o$*, an overall ranking of outliers regardless of their type; as well as a ranking by 2) global, 3) local, and 4) collective outlierness, respectively denoted by* $\mathscr{R}_g$*,* $\mathscr{R}_l$ *and* $\mathscr{R}_c$*.*

Definition 4.1.6 (**Ranking of Outliers**). *A ranking of outliers* $\mathscr{R}=(r_1,r_2,\ldots,r_n)$ *is a permutation of the set* $\{1,2,\ldots,n\}$*. It defines an ordering of the instances in a dataset* $\mathbf{X}=\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ *by separating the outliers* $\{\mathbf{x}_{r_1},\mathbf{x}_{r_2},\ldots,\mathbf{x}_{r_m}\}$ *from the inliers* $\{\mathbf{x}_{r_{m+1}},\mathbf{x}_{r_{m+2}},\ldots,\mathbf{x}_{r_n}\}$*, where* $m \ll n$ *is the number of outliers in* $\mathbf{X}$*.*

4.2 C-AllOut in a Nutshell

The main idea behind our method is to tackle the outlier annotation problem by leveraging the information present in a tree-based indexing structure, namely the Slim-tree. Particularly, C-AllOut takes advantage of the ability of this data structure to group together similar instances and keep apart distinct instances, with varying "degrees" of similarity being captured by means of distinct tree levels. The pseudocode of C-AllOut is in Algorithm 9.

In line with the problem statement in Definition 4.1.5, C-AllOut receives a dataset $\mathbf{X}$ as input, and produces four rankings of outliers $\mathscr{R}_o$, $\mathscr{R}_g$, $\mathscr{R}_l$ and $\mathscr{R}_c$, respectively reflecting outlierness overall, as well as global, local and collective outlierness.

In a nutshell, C-AllOut has two main phases. It begins in Phase 1 with a data-driven procedure that automatically creates and fine-tunes the base tree structure; see Lines 1-4 in Algorithm 9. In this phase, we refine the tree using the overall ranking produced from a preliminary tree. Once a fine-tuned tree is obtained, Phase 2 takes place; see Lines 5-10 in Algorithm 9. It extracts from the refined tree the information required to rank outliers by type, and produces the final output. The next two subsections give details on each of these phases.

4.2.1 Building and Refining the Tree

We build and refine a Slim tree as the base to both rank and annotate the outliers. For construction, the tree relies solely on pairwise distances and can be built using any metric distance function. This property makes C-AllOut applicable to settings where only distances

Algorithm 9: C-ALLOUT

Input: Dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n\}$
Output: Overall ranking $\mathscr{R}_o$; Global ranking $\mathscr{R}_g$; Local ranking $\mathscr{R}_l$; Collective ranking $\mathscr{R}_c$.

// Phase 1: refining the tree
$\mathscr{R} = (1, 2, \ldots, n)$ // given order of objects
$\mathscr{T} = \text{CREATETREE}(\mathbf{X}, \mathscr{R})$
$\mathscr{R}_o = \text{OVERALL}(\mathbf{X}, \mathscr{T})$ // Ranking 1 (§4.2.2)
$reverse\_\mathscr{R}_o = sort\_ascending(\mathscr{R}_o)$
// Phase 2: outlier-type focused ranking of objects
$\mathscr{T} = \text{CREATETREE}(\mathbf{X}, reverse\_\mathscr{R}_o)$
$\mathscr{R}_o = \text{OVERALL}(\mathbf{X}, \mathscr{T})$
$\mathscr{R}_g = \text{GLOBAL}(\mathbf{X}, \mathscr{T})$ // Ranking 2 (§4.2.4.1)
$\mathscr{R}_c = \text{COLLECTIVE}(\mathbf{X}, \mathscr{T})$ // Ranking 3 (§4.2.4.2)
$\mathscr{R}_l = \text{LOCAL}(\mathbf{X}, \mathscr{T})$ // Ranking 4 (§4.2.4.3)
return $\mathscr{R}_o, \mathscr{R}_g, \mathscr{R}_l, \mathscr{R}_c$

between objects are available, but not explicit features. Importantly, not all but only $O(n \log n)$ pairwise distances are required for construction, making it scalable to large datasets.

In the following two subsections, we (1) describe how to use the tree to create an overall outlierness ranking of the objects, which is then used to (2) improve the tree structure by ensuring that inliers are more likely to be inserted early on.

4.2.2 Overall Ranking of Outliers

Our overall ranking $\mathscr{R}_o$ is based on two structural components of the tree in Definitions 4.2.1 and 4.2.2 which are presented first.

Definition 4.2.1 (Closest Root Object)**.** *The closest root object* $\mathbf{x}_\mathtt{r}^{(i)}$ *of an object* $\mathbf{x}_i \in \mathbf{X}$ *inside a tree* $\mathscr{T}$ *is given by* $\mathbf{x}_\mathtt{r}^{(i)} := \{\mathbf{x}_j \in \mathbf{R} \mid \arg\min_j \mathtt{d}(\mathbf{x}_i, \mathbf{x}_j)\}$.

Definition 4.2.2 (Foreign Representative)**.** *The foreign representative* $\mathbf{x}_\mathtt{f}^{(i)}$ *of an object* $\mathbf{x}_i \in \mathbf{X}$ *indexed by a tree* $\mathscr{T}$ *is given by* $\mathbf{x}_\mathtt{f}^{(i)} := \{\mathbf{x}_j \in \mathbf{E} \mid \arg\min_j \mathtt{d}(\mathbf{x}_i, \mathbf{x}_j)\}$, *where* $\mathbf{E} = \{\mathbf{x}_\mathtt{e}^{(\eta)} \mid \eta \in \mathscr{T} \wedge \alpha_\eta = \alpha_{\tau^{(i)}}\}$.

Ranking 1 (Overall Ranking). The overall ranking of outliers $\mathscr{R}_o = (r_1, r_2, \ldots, r_n)$ for a dataset $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ indexed by a tree $\mathscr{T}$ is the ranking in which $\mathtt{s}_\mathtt{o}^{(r_i)} \geq \mathtt{s}_\mathtt{o}^{(r_{i+1})}\ \forall i \in \{1, 2, \ldots, n-1\}$, where

$$\mathtt{s}_\mathtt{o}^{(j)} = \mathtt{d}(\mathbf{x}_j, \mathbf{x}_\mathtt{r}^{(j)}) \cdot \mathtt{d}(\mathbf{x}_j, \mathbf{x}_\mathtt{f}^{(j)}) \tag{4.1}$$

The main idea is to leverage the set of objects $\mathbf{R}$ that are stored in the root node ρ of the tree $\mathscr{T}$ built from the input dataset $\mathbf{X}$, by recognizing that $\mathbf{R}$ is in fact a small and data-driven

sample (i.e., subset of objects from $\mathbf{X}$) containing only the objects from $\mathbf{X}$ that best summarize the main clusters of inliers. As described in Section 2.4, the leaf nodes of $\mathscr{T}$ store all the objects in $\mathbf{X}$, with each leaf having only objects that are very similar to each other. Distinctly, an internal node solely has one object from each one of its child nodes: the most centralized object in the child. It means that any internal node is essentially a cluster of clusters.

As a consequence, it is reasonable to expect that an object $\mathbf{x}_i \in \mathbf{X}$ that clearly belongs to a cluster of inliers tends to be the one selected as the representative $\mathbf{x}_{\mathtt{e}}^{(\tau^{(i)})}$ of its leaf node $\tau^{(i)}$, and then again selected to be the representative of the direct ancestor $\alpha_{\tau^{(i)}}$ of leaf $\tau^{(i)}$, and still continue being selected as a representative up to the root node ρ, thus becoming a member of set $\mathbf{R}$. This line of thought lead us to believe that each object in $\mathbf{R}$ represents one of the clusters of inliers in $\mathbf{X}$, where the larger is the cluster, the more representatives it has[1]. Following Definition 4.2.1, let $\mathbf{x}_{\mathtt{r}}^{(i)}$ be the object in $\mathbf{R}$ that is the closest to an object of interest $\mathbf{x}_i$. For short, we simply say that $\mathbf{x}_{\mathtt{r}}^{(i)}$ is the closest root object of $\mathbf{x}_i$. Note that it helps us to distinguish outliers from inliers: if $\mathbf{x}_i$ is an inlier, then it tends to be closer to $\mathbf{x}_{\mathtt{r}}^{(i)}$ than it would be if it were an outlier.

Nonetheless, some small clusters of inliers may be underrepresented in the root node. In this particular case, the distance to the closest root object becomes less helpful. Fortunately, there also exists the tendency that a cluster of inliers requires more than one leaf node to be stored, since any tiny cluster that fits entirely into a single leaf is more likely to contain collective outliers than inliers. This tendency also helps us to distinguish outliers from inliers: if an object $\mathbf{x}_i$ is close to at least one leaf other than leaf $\tau^{(i)}$, it tends to be an inlier; otherwise, $\mathbf{x}_i$ is probably an outlier. Following Definition 4.2.2, we efficiently take advantage of this idea through the concept of foreign representative $\mathbf{x}_{\mathtt{f}}^{(i)}$ of an object $\mathbf{x}_i$. The overall ranking of outliers $\mathscr{R}_o$ shown in Ranking 1 builds on the aforementioned concepts by using a new score $\mathtt{s}_{\mathtt{o}}^{(i)}$ as a base. It capitalizes on the proximity of an object $\mathbf{x}_i$ both to its closest root object $\mathbf{x}_{\mathtt{r}}^{(i)}$ and to its foreign representative $\mathbf{x}_{\mathtt{f}}^{(i)}$ to accurately and efficiently rank outliers of any type.

4.2.3 Refinement of Rankings

The tree structure that we leverage is built by a sequential insertion of objects. It has its own heuristics to efficiently avoid unfortunate selections of node representatives that nonetheless may occur depending on the order in which the objects are given for insertion. When a node splits in the tree, the two new representatives are going to be the most centralized objects from both newly-created nodes. These two new representative are also the ones that are going to be promoted to the parent node. In other words, only representatives, or most centralized objects, are going to have the chance to become representatives in the parent node. This behavior is repeated across all levels of the Slim-tree.

Although objects can be promoted, they cannot be demoted. In summary, if an outlier

[1]Note that $\mathbf{R}$ typically has a few hundreds of objects, so we expect $k \ll |\mathbf{R}|$, where k is the number of inlier clusters in $\mathbf{X}$.

object gets to the root of the tree by being the first one of its neighboring space to be inserted, i.e., by being considered the most centralized object in a premature stage of the tree construction, it can get promoted to the root, affecting our rankings. Because of this, it is best to insert first objects that are inliers and later outlier objects. It is possible to improve the initial tree created by insertions of randomly ordered objects, reaching a fine-tuned tree by creating it from a better ordering of the objects. Consequentially, C-ALLOUT becomes robust to any original order of objects given for a dataset $\mathbf{X}$.

There are multiple ways of achieving a good ordering of objects, but this is not an easy task. First of all, putting inliers first in a list of objects would already be an outlier detection task. Secondly, even if we try to approximate this ordering by considering the most centered objects in a dataset as inliers, we would still need to perform a task that usually has a time complexity of $O(n^2)$. Besides that, C-ALLOUT would not be a self-contained method anymore. A particularly good option, for not depending on methods other than our own, is to use our own overall score as a measure to sort the permutation.

As can be seen in Lines 1-4 of Algorithm 9, we propose to achieve this goal by performing a refinement of rankings. Specifically, an initial tree $\mathscr{T}$ is created by following the given order of objects $\mathscr{R}$; then, $\mathscr{T}$ can be refined by exploiting the overall ranking $\mathscr{R}_o$ produced from it. Note that function CREATETREE($\mathbf{X}$, $\mathscr{R}$) in Line 5 of Algorithm 9 uses the *reverse* order of $\mathscr{R}_o$ to insert objects in $\mathscr{T}$. The idea behind this procedure is to insert clear inliers first in $\mathscr{T}$, so as to avoid the selection of an outlier to be a node representative.

4.2.4 Outlier-type Focused Ranking of Objects

Moving further from an overall ranking, we next describe how C-ALLOUT ranks outliers of each particular type; focusing first on global outliers, and then the collective and local outlier ranking.

4.2.4.1 Global Ranking of Outliers

Here we build on the fact that the overall ranking by Equation (4.1) tends to rank global and collective outliers ahead of the other objects. Recall that the overall ranking $\mathscr{R}_o$ ranks each object according to its distance to the closest cluster of inliers. Both global and collective outliers tend to be far away from any of such clusters; distinctly, local outliers are close to at least one cluster, while the inliers themselves are even closer. Consequently, we expect that $\mathscr{R}_o = (r_1, r_2, \ldots, r_n)$ sorts any two objects $\mathbf{x}_{r_i}, \mathbf{x}_{r_j} \in \mathbf{X}$ such that $i < j$ only when $\mathbf{x}_{r_i}$ is either a global or a collective outlier, and $\mathbf{x}_{r_j}$ is not. Thus, the main challenge here is to differentiate the global outliers from the collective ones.

Fortunately, these two types of outliers can be distinguished: collective outliers have close neighbors; global ones do not. Building on Definitions 4.2.3 and 4.2.4, we form the ranking

$\mathscr{R}_g$ of global outliers by combining (a) the score $\mathtt{s}_\mathtt{o}^{(i)}$ of an object $\mathbf{x}_i$, which is the core of our overall ranking; with (b) the normalized distance $\mathtt{d}_\mathtt{nn}^{(i)}$ between $\mathbf{x}_i$ and its nearest neighbor $\mathbf{x}_\mathtt{nn}^{(i)}$, as shown in Equation (4.3).

Intuitively, score $\mathtt{s}_\mathtt{o}$ separates global and collective outliers from other objects, while distance $\mathtt{d}_\mathtt{nn}$ differentiates outliers of these two types. Note that the normalization in Equation (4.2)[2] balances the distances according to the local neighborhood of $\mathbf{x}_i$. Also, the nearest neighbor $\mathbf{x}_\mathtt{nn}^{(i)}$ is in fact an approximate one; it is efficiently obtained by comparing $\mathbf{x}_i$ only with the few objects in leaf $\tau^{(i)}$. It tends to be a good approximation since our base tree is built to have each leaf storing only objects that are very similar to each other.

Definition 4.2.3 (Approximate Nearest Neighbor)**.** *The approximate nearest neighbor* $\mathbf{x}_\mathtt{nn}^{(i)}$ *of object* $\mathbf{x}_i \in \mathbf{X}$ *in a tree* $\mathscr{T}$ *is* $\mathbf{x}_\mathtt{nn}^{(i)} := \{\mathbf{x}_j \in \mathbf{L}^{(i)} \mid \arg\min_{j \neq i} \mathtt{d}(\mathbf{x}_i, \mathbf{x}_j)\}$*, where* $\mathbf{L}^{(i)}$ *is the set of objects stored in leaf* $\tau^{(i)}$*.*

Definition 4.2.4 (Normalized 1nn Distance)**.** *The normalized nearest neighbor distance* $\mathtt{d}_\mathtt{nn}^{(i)}$ *of an object* $\mathbf{x}_i \in \mathbf{X}$ *indexed in a tree* $\mathscr{T}$ *is defined as*

$$\mathtt{d}_\mathtt{nn}^{(i)} = \frac{\mathtt{d}(\mathbf{x}_i, \mathbf{x}_\mathtt{nn}^{(i)})}{1 + \mathtt{d}(\mathbf{x}_\mathtt{e}^{(\tau^{(i)})}, \mathbf{x}_\mathtt{nn}^{(\tau^{(i)})})} \tag{4.2}$$

where $\mathbf{x}_\mathtt{nn}^{(\tau^{(i)})} := \mathbf{x}_\mathtt{nn}^{(j)}$ *s.t.* $\mathbf{x}_j$ *is equivalent to* $\mathbf{x}_\mathtt{e}^{(\tau^{(i)})}$*.*

Ranking 2 (Global Ranking). The global ranking $\mathscr{R}_g = (r_1, \ldots, r_n)$ for dataset $\mathbf{X}$ is the ranking in which $\mathtt{s}_\mathtt{g}^{(r_i)} \geq \mathtt{s}_\mathtt{g}^{(r_{i+1})}\ \forall i \in \{1, 2, \ldots, n-1\}$, where

$$\mathtt{s}_\mathtt{g}^{(j)} = \mathtt{s}_\mathtt{o}^{(j)} \cdot \mathtt{d}_\mathtt{nn}^{(j)} \tag{4.3}$$

4.2.4.2 Collective Ranking of Outliers

The ranking $\mathscr{R}_c$ of collective outliers builds naturally from the ideas discussed in the previous section, where we exploit the tendency of global and collective outliers to have the highest scores $\mathtt{s}_\mathtt{o}$, and then "amplify" each object $\mathbf{x}_i$ that is a global outlier by multiplying $\mathtt{s}_\mathtt{o}^{(i)}$ by the nearest neighbor distance $\mathtt{d}_\mathtt{nn}^{(i)}$. In a similar fashion, we bump up collective outliers simply by replacing the multiplication with its reverse operation division[2], as presented in Equation (4.4).

Ranking 3 (Collective Ranking). The collective ranking $\mathscr{R}_c = (r_1, \ldots, r_n)$ for dataset $\mathbf{X}$ is the ranking in which $\mathtt{s}_\mathtt{c}^{(r_i)} \geq \mathtt{s}_\mathtt{c}^{(r_{i+1})}\ \forall i \in \{1, \ldots, n-1\}$, where

$$\mathtt{s}_\mathtt{c}^{(j)} = \frac{\mathtt{s}_\mathtt{o}^{(j)}}{1 + \mathtt{d}_\mathtt{nn}^{(j)}} \tag{4.4}$$

[2]Note that value 1 in the denominator avoids division by zero.

4.2.4.3 *Local Ranking of Outliers*

The local ranking of outliers $\mathscr{R}_l$ is formally given in Ranking 4 below, which builds on Definitions 4.2.5 and 4.2.6.

Definition 4.2.5 (Normalized Leaf Radius)**.** *The normalized leaf radius* $\mathtt{a}^{(\tau)}$ *of a leaf* τ *in a tree* $\mathscr{T}$ *is*

$$\mathtt{a}^{(\tau)} = \frac{\mathtt{d}(\mathbf{x}_\mathtt{e}^{(\tau)}, \mathbf{x}_\mathtt{a}^{(\tau)})}{1+\mathtt{d}(\mathbf{x}_\mathtt{e}^{(\tau)}, \mathbf{x}_\mathtt{nn}^{(\tau)})} \tag{4.5}$$

where $\mathbf{x}_\mathtt{a}^{(\tau)} := \{\mathbf{x}_i \in \mathbf{L}^{(\tau)} | \arg\max_i \mathtt{d}(\mathbf{x}_\mathtt{e}^{(\tau)}, \mathbf{x}_i)\}$ *is the farthest from the leaf representative among objects* $\mathbf{L}^{(\tau)}$ *in leaf* τ*, and* $\mathbf{x}_\mathtt{nn}^{(\tau)} := \mathbf{x}_\mathtt{nn}^{(j)}$ *s.t.* $\mathbf{x}_j$ *is equiv. to* $\mathbf{x}_\mathtt{e}^{(\tau)}$*.*

Definition 4.2.6 (Knee Radius)**.** *The knee radius* $\mathtt{a}_\mathtt{kr}$ *for a dataset* $\mathbf{X} = \{\mathbf{x}_1, \ldots \mathbf{x}_n\}$ *in a tree* $\mathscr{T}$ *is defined as*

$$\mathtt{a}_\mathtt{kr} = \mathtt{kneedle}(\mathtt{a}_1, \ldots, \mathtt{a}_{|\mathtt{A}|}) \tag{4.6}$$

s.t. $\mathtt{A} = \{\mathtt{a}^{(\tau^{(1)})}, \ldots, \mathtt{a}^{(\tau^{(n)})}\}$*, and* $\mathtt{a}_j < \mathtt{a}_{j+1}\ \forall j \in \{1, \ldots, |\mathtt{A}|-1\}$*. Function* `kneedle` *refers to algorithm Kneedle (SATOPAA et al., 2011); it takes a sorted list* $(\mathtt{a}_1, \ldots, \mathtt{a}_{|\mathtt{A}|})$ *as input and returns one of its values* $\mathtt{a}_p$*.*

Ranking 4 (Local Ranking). The local ranking for dataset $\mathbf{X}$ is denoted by $\mathscr{R}_l$. Let $\mathbf{P} = \{\mathbf{x}_i \mid \mathtt{d}_\mathtt{nn}^{(i)} \geq \mathtt{a}_\mathtt{kr}\}$. Then, $\mathscr{R}_l = (r_1, \ldots, r_n)$ is the ranking in which

$$\mathbf{x}_{r_j} \in \begin{cases} \mathbf{P}, & \textit{if } \ 1 \leq j \leq |\mathbf{P}| \\ \mathbf{X} \setminus \mathbf{P}, & \textit{otherwise} \end{cases} \tag{4.7}$$

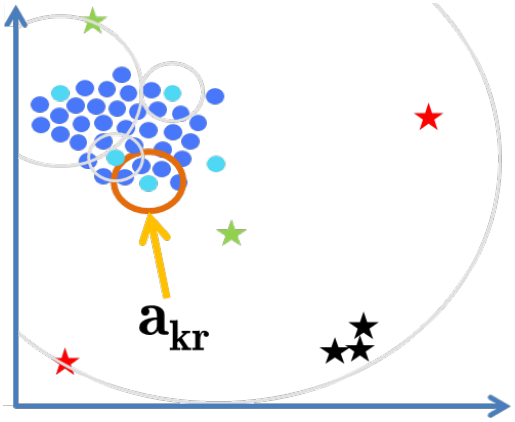
where $\mathtt{s}_\mathtt{g}^{(r_p)} \leq \mathtt{s}_\mathtt{g}^{(r_{p+1})}\ \forall p \in \{1, \ldots, |\mathbf{P}|-1\}$ and $\mathtt{s}_\mathtt{g}^{(r_q)} \geq \mathtt{s}_\mathtt{g}^{(r_{q+1})}\ \forall q \in \{|\mathbf{P}|+1, \ldots, n-1\}$.

The main idea is to distinguish local outliers from other objects by using the normalized leaf radius $\mathtt{a}^{(\tau)}$ of each leaf $\tau \in \mathscr{T}$. Particularly, we expect $\mathtt{a}^{(\tau)}$ to be: (a) small, if the object $\mathbf{x}_\mathtt{a}^{(\tau)} \in \mathbf{X}$ that defines the radius is an inlier; (b) larger, if $\mathbf{x}_\mathtt{a}^{(\tau)}$ is a local outlier; (c) very large, if $\mathbf{x}_\mathtt{a}^{(\tau)}$ is a global outlier; and (d) either very small or large[3], if $\mathbf{x}_\mathtt{a}^{(\tau)}$ is a collective outlier.

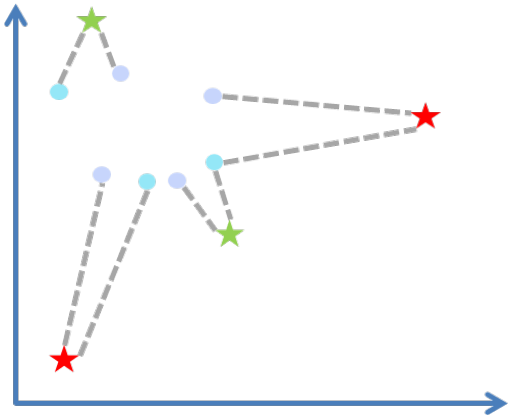
As a result, the vast majority of leaf radii are expected to be small, mostly coming from case (a) and some from case (d) above, with only a few large radius leaves. Thus, when the radii in $\mathtt{A} = \{\mathtt{a}^{(\tau^{(1)})}, \ldots, \mathtt{a}^{(\tau^{(n)})}\}$ are sorted into a list $(\mathtt{a}_1, \ldots, \mathtt{a}_{|\mathtt{A}|})$, there must exist a "knee" value $\mathtt{a}_\mathtt{kr}$ that clearly separates the small radii from the rest. Intuitively, $\mathtt{a}_\mathtt{kr}$ is approximately the smallest nearest neighbor distance $\mathtt{d}_\mathtt{nn}^{(i)}$ considering each local outlier object $\in \mathbf{X}$. Therefore, all objects which have their closest neighbor farther than $\mathtt{a}_\mathtt{kr}$, can be considered either local or global outliers. Our Ranking 4 computes $\mathtt{a}_\mathtt{kr}$ using an off-the-shelf "knee"-detector algorithm; then, it separates large-radii outliers in set $\mathbf{P}$, and reuses our global score to rank them in increasing order of $\mathtt{s}_\mathtt{g}$, effectively placing local outliers ahead of the other objects.

[3] If the collective outliers are in a leaf of their own, $\mathtt{a}^{(\tau)}$ would be very small, and very large otherwise.

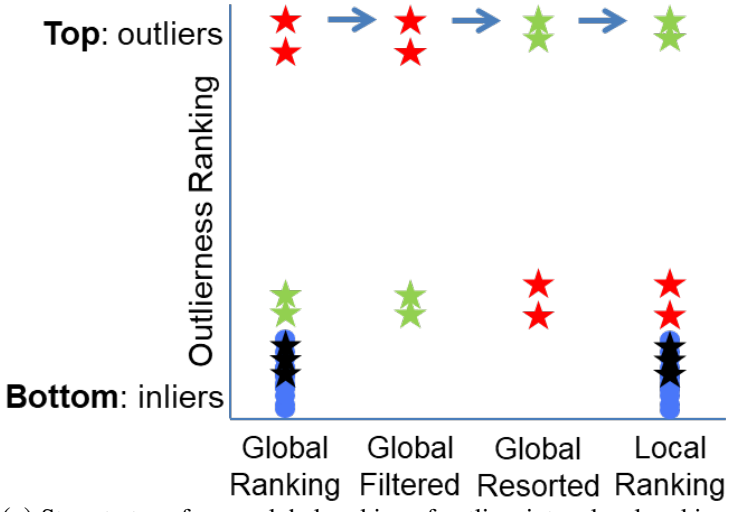
Figure 10 helps visualize the steps done to reach the local ranking of outliers $\mathscr{R}_l$. In Figure 10a we have the same input dataset presented in Figure 1 with the knee leaf radius a_{kr} returned by function `kneedle` with a orange circle. It is selected as the knee radius because it is the knee point in a curve intersecting the sorted list of leaf radius values. Figure 10b shows the remaining points after filtering using a_{kr} as opaque points. The filter removes every point $\mathbf{x}_i \in \mathbf{X}$ that has its nearest neighbor closer than a_{kr}. Therefore, the only remaining points are the green local outliers and the red global outliers. For reference, the nearest neighbor points used in the filtering step, and also in the global score s_g, are kept in transparent color along with the representative point of each not filtered point. Figure 10c illustrates how filtered points are ignored in the global ranking so that the remaining points (local and global outliers) can be resorted in ascending order using s_g. Finally, local outliers take the top positions of the local ranking and previously filtered points are appended to the end of the ranking.



(a) Data points and Slim-tree leaves with the knee radius a_{kr} highlighted in orange.



(b) Data points after filtering with a_{kr}. Transparent points kept for reference of s_g.



(c) Steps to transform a global ranking of outliers into a local ranking of outliers.

Figure 10 – The local ranking of outliers $\mathscr{R}_l$ visualized.

4.2.5 Time and Space Complexity

Lemma 4.2.1 (Time Complexity). *The overall time complexity of* C-ALLOUT *is* $O(n \log n)$.

Proof. As shown in Algorithm 9, C-ALLOUT has two phases. The cost of Phase 1 is mostly in Lines 2 and 3. The former builds a Slim-tree $\mathscr{T}$ for dataset $\mathbf{X}$; the latter analyzes $\mathscr{T}$ to obtain an overall ranking $\mathscr{R}_o$. As Slim-tree is one state-of-the-art tree-based data structure, it requires only $O(n \log n)$ time for construction. The time to build $\mathscr{R}_o$ is $O(n)$, since C-ALLOUT needs to find $\mathbf{x}_{\mathtt{r}}^{(i)}$ and $\mathbf{x}_{\mathtt{f}}^{(i)}$ by comparing each object $\mathbf{x}_i \in \mathbf{X}$ with the objects stored in only two nodes, i.e., ρ and $\alpha_{\tau^{(i)}}$, whose sizes are limited by a small constant c. Thus, the total time of Phase 1 is $O(n \log n)$. In Phase 2, the rankings $\mathscr{R}_g$ and $\mathscr{R}_c$ are computed together in $O(n)$ time since they both reuse the score $\mathtt{s}_{\mathtt{o}}^{(i)}$ from $\mathscr{R}_o$ and calculate $\mathtt{d}_{\mathtt{nn}}^{(i)}$ by comparing objects from $\alpha_{\tau^{(i)}}$ with each object $\mathbf{x}_i \in \mathbf{X}$ to identify $\mathbf{x}_{\mathtt{nn}}^{(i)}$ and $\mathbf{x}_{\mathtt{nn}}^{(\tau^{(i)})}$. Finally, local ranking $\mathscr{R}_l$ is computed in $O(n + |\mathtt{A}|^2)$ time by computing $\mathtt{a}^{(\tau^{(i)})}$ for every object $\mathbf{x}_i \in \mathbf{X}$ in $O(n)$ time; then, identifying $\mathtt{a}_{\mathtt{kr}}$ in $O(|\mathtt{A}|^2)$ time. Provided that $|\mathtt{A}|$ is expected to be a small constant, the total time of Phase 2 is $O(n)$. It leads to the conclusion that the time complexity of C-ALLOUT is $O(n \log n)$. □

Lemma 4.2.2 (Space Complexity). *The overall space complexity of* C-ALLOUT *is* $O(n \log n)$.

Proof. The space required by C-ALLOUT refers to the storage of Slim-tree $\mathscr{T}$ and of the four rankings $\mathscr{R}_o$, $\mathscr{R}_g$, $\mathscr{R}_c$ and $\mathscr{R}_l$. As Slim-tree is one state-of-the-art tree-based data structure, it requires $O(n \log n)$ space to be stored. Each of the rankings $\mathscr{R}_o$, $\mathscr{R}_g$, $\mathscr{R}_c$ and $\mathscr{R}_l$ are in fact a permutation of set $\{1, 2, \ldots, n\}$; therefore, they require $O(n)$ space to be stored. It leads to the conclusion that the space complexity of C-ALLOUT is $O(n \log n)$. □

4.3 Final Considerations

In this chapter we presented C-ALLOUT, our novel method for outlier detection and annotation. We have shown more details on the Slim-tree and how we are able to extract information from its structure built with any dataset. We have provided definitions which, while being crucial to our rankings, are not original concepts of the Slim-tree. As it could be seen, our solution outputs four rankings, one overall ranking and three other for specific outlier types. These four rankings are evaluated separately in three different testbeds we prepared for our experiments. Next chapter yields our experimental evaluation having C-ALLOUT and the algorithms present in Table 1 from Section 3.3. The chapter also serves pertinent details on synthetic data generation and the characteristics of the datasets.

CHAPTER

5

RESULTS AND DISCUSSIONS

We designed experiments to evaluate C-ALLOUT with respect to both overall detection as well as outlier annotation tasks, aiming to answer the following.

Q1: Overall detection performance: How does C-ALLOUT compare to state-of-the-art baselines on benchmark datasets with respect to overall detection performance, regardless of outlier type?

Q2: Outlier annotation performance: How well does C-ALLOUT perform on outlier annotation, that is, with respect to its ranking of the outliers by type?

5.1 Overall Outlier Detection Performance

5.1.1 Experiment Setup

First we aim to showcase that, while specializing on outlier annotation, C-ALLOUT is competitive in terms of overall outlier detection on standard benchmark datasets compared to state-of-the-art baselines. We remark that C-ALLOUT's overall ranking $\mathscr{R}_o$ is used for evaluation here, regardless of outlier type.

5.1.1.1 Testbed 1

We create our first testbed using classical benchmark datasets[1,2,3]. Specifically, a total of 20 datasets were chosen at random among those with outlier percentage less than 25%; a detailed list of which is in Table 2.

[1] <http://odds.cs.stonybrook.edu/>
[2] <http://lapad-web.icmc.usp.br/repositories/outlier-evaluation/DAMI/>
[3] <https://archive.ics.uci.edu/ml/datasets.php>

Table 2 – **Testbed 1:** Summary of the 20 benchmark datasets used for overall outlier detection.

Dataset	Points	Outliers	Features
Annthyroid	7,062	534	6
Cardio	2,110	465	21
Glass	213	9	7
Letter	1,598	100	32
Mammography	7,848	253	6
Mnist	7,603	700	100
Musk	3,062	97	166
Optdigits	5,198	132	64
Pageblocks	5,393	510	10
Pendigits	6,870	156	16
Satimage-2	5,801	69	36
Shuttle	1,013	13	9
Shuttle2	49,097	3,511	9
Stamps	340	31	9
Thyroid	3,656	93	6
Vowels	1,452	46	12
Waveform	3,443	100	21
WBC	377	20	30
WDBC	367	10	30
Wilt	4,819	257	5

5.1.1.2 Competitors

We compare against five state-of-the-art similarity or distance-based detection algorithms; namely, LOF (BREUNIG *et al.*, 2000), kNN (RAMASWAMY; RASTOGI; SHIM, 2000), OCSVM (SCHöLKOPF *et al.*, 1999), Similarity Isolation Forest (SIF) (CZEKALSKI; MORZY, 2021) and RAE (ZHOU; PAFFENROTH, 2017). Like C-ALLOUT, the first four are similarity or distance-based algorithms. Specifically, LOF, kNN and SIF employ Euclidean distance and OCSVM uses the linear kernel. RAE is an autoencoder and uses the actual features. It is added as competitor so that we can compare C-ALLOUT to a deep-learning-based and recent state-of-the-art algorithm. Note that SIF is a version of the popular iForest (LIU; TING; ZHOU, 2008), constructed by using pairwise similarities only based on ideas from similarity forests (SATHE; AGGARWAL, 2017). (See Chapter 3)

Unlike C-ALLOUT, all baselines exhibit one or more hyperparameters that are non-trivial to pick without any labels. We run each baseline using a corresponding list of hyperparameter configurations (see Table 3 and Table 4) and report the average performance. In effect, this is equivalent to their expected performance when hyperparameters are selected at random from the list. For the non-deterministic baselines SIF and RAE, the performance is averaged over 10 random runs.

Table 3 – Hyperparameters and configurations for the baseline methods

LOF_k	kNN_k	OCSVM_nu	SIF_n_trees/SIF_ψ
1	1	0.01	100/256
3	3	0.05	-
5	5	0.1	-
10	10	0.15	-
15	15	0.2	-
30	30	0.25	-
50	50	0.3	-
75	75	0.35	-
100	100	0.4	-
-	-	0.45	-
-	-	0.5	-

5.1.1.3 Model Configurations

Table 3 reports the full list of the hyperparameters used for each of our competitors. Column names have the form `method_parameter`, where `method` is the name of the competitor, and `parameter` is the corresponding hyperparameter name. SIF only uses its default values of 100 for number of trees *n_trees* and 256 for ψ. OCSVM hyperparameters range from a very small value up to its conservative default value of 0.5 for an upperbound on the number of outliers. Finally, both LOF and kNN use values of $k \in \{1,3,5,10,15,30,50,75,100\}$ in their searches for the k^{th} nearest neighbors.

Table 4 reports the hyperparameters used for RAE. They were defined to be aligned with the original anomaly detection proposal (ZHOU; PAFFENROTH, 2017). Column names have the hyperparameter names. RAE only uses its default value of 0.001 for the Learning Rate (LR). The autoencoder network architecture is always symmetric.

Table 4 – Hyperparameters for baseline RAE.

n_layers	λ	input_dim_decay	LR	Inner/Outer Iterations
2	$10e-5$	1	0.001	20
3	$7.5e-5$	2	-	50
4	$10e-4$	4	-	-

5.1.1.4 Measures

We evaluate the outlier ranking quality using both Area Under the Curve of Receiver Operating Characteristic (AUCROC) and Area Under the Curve of Precision-Recall (AUCPR), sometimes referred to as Average Precision (AP). In addition, we apply the paired Wilcoxon signed rank test on AUCROC and AUCPR results across datasets to establish statistical significance of the differences.

5.1.2 Results

Table 5 and Table 6 present, respectively, the AUCROC and AUCPR overall detection performance results for all methods across all benchmark datasets. The last lines on both tables have the average ranking of each method for every dataset, lower values are better. Additionally, Table 7 provides the p-values for one-sided significance tests of the differences, testing that the left-side method (C-ALLOUT) is better than the right-side (baseline) against the null hypothesis that they are indifferent. In terms of average AUCROC performance, C-ALLOUT significantly outperforms LOF, OCSVM and SIF and achieves better results than the second best performer kNN (1.900 vs. 2.050) where the differences are not significant ($p \approx 0.1$). Similar results hold for AUCPR, where C-ALLOUT achieves the best average performance. The performance is on par with RAE's and significantly better than the rest.

Table 5 – AUCROC performance on Testbed 1

Dataset	C-ALLOUT-$\mathscr{R}_o$	LOF ▼	kNN	OCSVM ▼	IF ▼	RAE ■
Annth.	0.720	0.718	**0.773**	0.469	0.585	0.677
Cardio	0.539	0.535	0.503	0.380	**0.795**	0.743
Glass	**0.839**	0.738	0.825	0.638	0.634	0.685
Letter	0.779	**0.849**	0.841	0.491	0.219	0.465
Mamm.	**0.807**	0.643	0.802	0.343	0.772	0.703
Mnist	0.850	0.662	0.852	0.408	0.585	**0.861**
Musk	0.972	0.555	0.795	0.415	0.919	**0.997**
Optdig.	**0.519**	0.506	0.437	0.504	0.372	0.443
Pageb.	**0.896**	0.712	0.877	0.523	0.852	0.894
Pendig.	0.859	0.517	0.813	0.615	0.879	**0.918**
Satim.	**0.994**	0.639	0.961	0.527	0.894	0.961
Shuttle	**0.985**	0.851	0.953	0.374	0.765	0.906
Shuttle2	0.759	0.517	0.701	0.647	0.963	**0.991**
Stamps	0.838	0.643	0.856	0.530	**0.859**	0.795
Thyroid	0.952	0.721	0.957	0.514	0.876	**0.963**
Vowels	0.935	0.885	**0.953**	0.549	0.317	0.484
Wavef.	0.690	0.696	**0.742**	0.452	0.647	0.623
WBC	0.940	0.856	**0.947**	0.403	0.914	0.857
WDBC	0.912	0.831	0.923	0.474	**0.927**	0.855
Wilt	0.455	**0.635**	0.541	0.296	0.299	0.466
Avg. Rank	**2.25**	3.95	2.50	5.50	3.80	3.00

5.2 Outlier Annotation Performance

5.2.1 Experiment Setup

Next we aim to show the effectiveness of C-ALLOUT on annotating outliers by type, and specifically the accuracy of its type-specific rankings. However, there exists no benchmark datasets with annotated outliers. Hence, Testbed 1 datasets do not have a characterization of their outliers. To this end, we create two novel testbeds containing specific types of outliers, described as follows.

Table 6 – AUCPR performance on Testbed 1

Dataset	C-ALLOUT-$\mathscr{R}_o$	LOF ▼	kNN ●	OCSVM ▼	IF ■	RAE
Annth.	0.217	0.174	0.214	0.077	0.103	**0.233**
Cardio	0.324	0.272	0.312	0.209	**0.582**	0.504
Glass	0.117	0.130	0.127	0.116	0.058	**0.135**
Letter	0.173	**0.385**	0.258	0.071	0.037	0.071
Mamm.	0.163	0.096	0.165	0.083	**0.183**	0.105
Mnist	0.389	0.233	0.396	0.101	0.158	**0.404**
Musk	0.862	0.178	0.559	0.040	0.754	**0.925**
Optdig.	0.028	0.029	0.021	**0.032**	0.019	0.021
Pageb.	**0.542**	0.303	0.515	0.237	0.420	0.536
Pendig.	0.171	0.036	0.109	0.075	0.135	**0.235**
Satim.	**0.868**	0.050	0.583	0.108	0.170	0.702
Shuttle	**0.315**	0.220	0.271	0.011	0.027	0.164
Shuttle2	0.245	0.101	0.182	0.317	0.776	**0.927**
Stamps	0.271	0.186	0.283	0.258	**0.289**	0.257
Thyroid	0.392	0.137	0.337	0.162	0.237	**0.476**
Vowels	0.393	0.320	**0.477**	0.076	0.021	0.042
Wavef.	0.073	0.079	**0.110**	0.028	0.049	0.053
WBC	0.565	0.397	0.518	0.075	**0.571**	0.532
WDBC	0.558	0.433	0.532	0.101	**0.655**	0.560
Wilt	0.044	**0.079**	0.053	0.048	0.035	0.048
Avg. Rank	**2.60**	4.10	2.90	5.05	3.75	**2.60**

Table 7 – Paired one-sided significance test (Wilcoxon): C-ALLOUT vs. baseline with respect to overall performance on Testbed 1. p-value in **bold** if significant at 0.1.

Methods	AUCROC p-value	AUCPR p-value
C-ALLOUT-$\mathscr{R}_o$ > LOF	**0.001**	**0.001**
C-ALLOUT-$\mathscr{R}_o$ > kNN	0.337	**0.095**
C-ALLOUT-$\mathscr{R}_o$ > OCSVM	**0.001**	**0.001**
C-ALLOUT-$\mathscr{R}_o$ > IF	**0.005**	**0.029**
C-ALLOUT-$\mathscr{R}_o$ > RAE	**0.045**	0.449

5.2.1.1 Testbed 2

We start with simulating three synthetic base datasets, denoted S1, S2 and S3; consisting of Gaussian inlier clusters of varying count, size, standard deviation, and dimensionality. (See Table 9) From each base dataset Sx, we create four versions: Sx_G, Sx_L, Sx_C, and Sx_A respectively with only global, only local, only collective, and all three types of outliers combined. Global and local outliers are simulated from each inlier cluster by up-scaling the respective standard deviation, while each set of collective outliers are simulated from a global outlier as the mean and down-scaled standard deviation.

Table 8 – **Testbeds 2 and 3:** Summary of the 8 base datasets used in for outlier annotation.

Dataset	Points	Outliers	Features
S1	24,713	90	2
S2	13,832	120	5
S3	10,076	60	10
ALOI	48,266	240	27
Glass	234	30	9
skin	14,726	72	3
smtp	71,566	357	3
Waveform	3,391	48	21

5.2.1.2 Synthetic Data Generation

We show in Table 9 the hyperparameters used in our synthetic dataset generator. For every dataset variation having a specific type of outlier, the number of outliers is equal to one third of the total number of outliers.

Table 9 – Parameters used in the synthetic data generator; each outlier type corresponds to a third of the total number.

Dataset	# points	# clusters	# outliers	standard deviation	# dimensions
S1_A	500-5000	10	90	10-50	2
S2_A	500-5000	7	120	10-50	5
S3_A	500-5000	3	60	10-50	10

The sequence of steps in Algorithm 10 gives an overview of the synthetic dataset generator. The synthetic generator creates multiple Gaussians distributed across the space. The points generated in these initial Gaussian distributions become the inlier clusters. The space in which the center of the Gaussians can fall is defined using a range bounded by the number of Gaussians times the maximum standard deviation in all Gaussians. This way, the space can accommodate all Gaussians without too much overlap between them.

Specific types of outliers are generated like this: (1) Local outliers are generated using as Gaussians having the same mean as the inlier Gaussians, only with a magnified standard deviation; (2) Global outliers are generated using the same method as local outliers, only with a larger standard deviation; (3) For collective outliers, we base the number of collectives clusters on the desired number of collective outliers. All collective clusters have a fixed size of ten points, and randomly pick from the list of available global outliers. These selected global outliers are labeled as collective outliers. Then, a small Gaussian is generated having the global points coordinates as mean, only with a standard deviation that is a fraction of the minimum standard deviation for the inlier clusters.

5.2.1.3 Testbed 3

We also create a testbed with real-world data inliers, and simulated outliers based on Steinbuss and Böhm's "realistic synthetic data" generator (STEINBUSS; BÖHM, 2021). The idea is to start with a real-world dataset containing only inliers, to which a Gaussian Mixture Model (GMM) is fit. Outliers are then simulated by up-scaling the variances of the GMM clusters. Their generator does not create collective outliers; to that end, we follow suit with Testbed 2 and simulate small-standard-deviation micro-clusters centered at randomly chosen global outliers. As the base, we use five real-world datasets (after discarding the (non-annotated) ground-truth outliers); namely ALOI, Glass, skin, smtp, Waveform[3] with various sizes and dimensionalities to represent different dataset characteristics (see Table 8). Similar to Testbed 2, we create 4 variants (_G, _L, _C, _A) from each base dataset with various types of (annotated) outliers.

Algorithm 10: Synthetic Dataset Generator

Input: Minimum size for inlier clusters *min_inlier*, maximum size for inlier clusters *max_inlier*, minimum standard deviation for inlier clusters *min_std*, maximum standard deviation for inlier clusters *max_std*, number of inlier clusters to generate *num_inlier_c*, number of local outliers to generate *num_local*, number of global outliers to generate *num_global*, number of collective outlier micro clusters to generate *num_collective_c*, number of collective outliers per micro cluster *num_collective_p*, dimensionality for the data *ndim*.

Output: synthetic dataset Y with outlier type labels.

Define the space range to be used to generate all points based on the hyperparameters *max_std*, *num_inlier_clusters*, *ndim*

$C = gaussian\_inliers(num\_inlier\_c, min\_inlier, max\_inlier, min\_std, max\_std, ndim)$

Based on *max_std*, set the variances to be used for each outlier type generation

for *c in C* **do**

  $locals = gaussian\_locals(c, ndim)$

  $globals = gaussian\_globals(c, ndim)$

  $final\_locals, globals = filter\_close\_outliers(locals, globals, num\_local)$

  $globals = filter\_globals\_near\_inliers(c, globals, num\_global)$

end

$collective\_centers, final\_globals = random\_select(globals, num\_collective\_c)$

$final\_collectives = gaussian\_collectives(collective\_centers, num\_collective\_p, ndim)$

Return dataset Y with inliers and outliers joined, along with respective labels

5.2.1.4 Realistic Data Generation

As it can be seen in Algorithm 11, the realistic data generator has two main phases. In the first one, we alter the dataset generator proposed by Steinbuss and Böhm (2021) so that it can return separate outlier type labels and other necessary information like the predicted densities for the points. In the second phase, we take the output from their generator and work around it to remove some problematic outliers and add collective outliers. The realistic dataset generator starts by taking the inlier data from the base dataset X passed to it. The generator then fits a Gaussian Mixture Model (GMM) *fitted_GMM* to the inlier points. The GMM uses a VEI setting, which means that all fitted Gaussians have a diagonal covariance matrix, varying volume and equal shape.

After the model is finished fitting to the inlier data, the generator starts simulating local and global outliers. For each Gaussian, using its mean and a five times increased standard deviation, a Gaussian cluster of points is created. The generator returns the inlier points along with the simulated outliers with a binary label for outlierness. The outlier binary label have no use in the outlier annotation task, so we label local and global outlier points using their densities provided by the GMM. Lots of generated outlier points are going to be intersecting inlier clusters. By removing a subsample which corresponds to 68.3% of the data, we clear out most of the interior outliers that overlap with inliers. 68.3% is the percentage of points that lie one standard

deviation from a Gaussian mean. A third of the remaining 31.7% of points is also discarded, to create a margin between inliers and outliers.

After subsampling, the remaining outlier points with the lowest density become global outliers. Remaining outlier points with the highest density become local outliers. By selecting the extreme sections to create the outliers, a gap between global and local outliers is naturally formed and they become well-defined in space. The collective outlier micro-clusters are generated around points randomly selected from the global outlier pool.

Algorithm 11: Realistic Dataset Generator

Input: Base dataset X, number of outliers to generate in Phase 1 n_out (same as number of inliers by default), data distribution to use for simulated outliers d (Gaussian by default), number of local outliers to generate num_local, number of global outliers to generate num_global, number of collective outlier micro clusters to generate $num\_collective\_c$, number of collective outliers per micro cluster $num\_collective\_p$.

Output: Realistic dataset Y with labels, GMM metadata Y_var, predicted density for all points using inlier-fitted GMMs Y_dens.

// Phase 1: GMM fitting and outlier generation

Instantiate and fit GMM model $fitted\_GMM$ on X using d

Increase the Gaussians variances learned by the GMM model in $fitted\_GMM$ by 5 times

for g *in* $fitted\_GMM.gaussians$ **do**

    $locals, globals = generate\_outliers(g.variance, n\_out)$

end

Prepare dataset Y with real inliers from X and generated realistic outliers joined, along with each point label and Y_dens predicted density by $fitted\_GMM$ for points in Y

// Phase 2: Outlier filtering and collective outlier generation

Sort all outlier points in Y by their density from Y_dens

Remove 68.3% (1 std from a Gaussian) of the most dense outlier points

Divide the remaining 31.7% in three portions and discard the most dense one

$final\_locals = num\_local$ most dense points in the edited Y

$globals = num\_global + num\_collective\_c$ least dense points in the edited Y

$collective\_centers, final\_globals = random\_select(globals, num\_collective\_c)$

$final\_collectives = gaussian\_collectives(collective\_centers, num\_collective\_p)$

Return dataset Y with inliers and outliers joined, along with respective labels

5.2.1.5 Competitor

There is no baseline for annotating all three types of outliers. We found the spectral SRA (NIAN *et al.*, 2016), also based on pairwise similarities (i.e. Laplacian), as a close attempt (details in Section 3.2). When it identifies a small cluster[4], it raises the flag "clustered" and "scattered" otherwise. It provides a single ranking and does not distinguish between global and local outliers. Thus, SRA is only comparable in some cases.

[4]SRA needs a size threshold for collective outliers; we give it an advantage by setting the threshold equal to the ground-truth size.

5.2.1.6 Measures

We evaluate the outlier ranking quality using, again, both AUCROC and AUCPR. Note that, now there are different AUCROC and AUCPR measures for each one of the 4 rankings that C-ALLOUT provides. That includes the overall ranking $\mathscr{R}_o$, also tested in Testbed 1, and the specific rankings $\mathscr{R}_g$, $\mathscr{R}_l$ and $\mathscr{R}_c$. Since C-ALLOUT is clearly better than SRA, no Wilcoxon signed rank tests were performed.

5.2.2 Results

Table 10 presents AUCROC results on Testbed 2. C-ALLOUT achieves near-ideal overall performance, while SRA ranking is significantly poor. C-ALLOUT type-specific rankings continue to achieve near-perfect performance, while SRA struggles, especially in identifying local outliers with a near-random ranking. Moreover, it fails to flag "clustered" on datasets that contain collective outliers. Results are similar with respect to AUCPR (see Table 11). "NF", or not flagged, cells show that the method was not able to note the existence of that type of outlier described in the name of the dataset.

Table 10 – AUCROC performance on Testbed 2 with annotated outliers. Note that in Sx_A datasets with all three outlier types, type-specific rankings are evaluated against outliers of the corresponding type only.

Dataset	C-ALLOUT-$\mathscr{R}_o$	SRA	C-ALLOUT-$\mathscr{R}_l$	SRA	C-ALLOUT-$\mathscr{R}_g$	SRA	C-ALLOUT-$\mathscr{R}_c$	SRA
S1_A	**0.998**	0.663	**0.997**	0.452	**1**	0.851	**0.999**	NF
S1_L	**0.998**	0.453	**0.999**	0.453	-	-	-	-
S1_G	**1**	0.851	-	-	**1**	0.851	-	-
S1_C	**1**	0.685	-	-	-	-	**1**	NF
S2_A	**0.999**	0.703	**0.997**	0.511	**1**	0.747	**1**	NF
S2_L	**0.999**	0.512	**1**	0.512	-	-	-	-
S2_G	**1**	0.744	-	-	**1**	0.744	-	-
S2_C	**1**	0.849	-	-	-	-	**1**	NF
S3_A	**0.963**	0.599	**0.918**	0.567	**1**	0.729	**1**	NF
S3_L	**0.950**	0.576	**0.993**	0.576	-	-	-	-
S3_G	**1**	0.730	-	-	**1**	0.730	-	-
S3_C	**1**	0.500	-	-	-	-	**1**	NF

Table 11 – AUCPR performance on Testbed 2 with annotated outliers. Note that in Sx_A datasets with all three outlier types, type-specific rankings are evaluated against outliers of the corresponding type only.

Dataset	C-ALLOUT-$\mathscr{R}_o$	SRA	C-ALLOUT-$\mathscr{R}_l$	SRA	C-ALLOUT-$\mathscr{R}_g$	SRA	C-ALLOUT-$\mathscr{R}_c$	SRA
S1_A	**0.862**	0.458	**0.241**	0.001	**1**	0.661	**0.356**	NF
S1_L	**0.740**	0.001	**0.874**	0.001	-	-	-	-
S1_G	**0.988**	0.696	-	-	**1**	0.696	-	-
S1_C	**1**	0.667	-	-	-	-	**1**	NF
S2_A	**0.976**	0.368	**0.292**	0.003	**0.983**	0.086	**0.978**	NF
S2_L	**0.878**	0.017	**0.989**	0.017	-	-	-	-
S2_G	**1**	0.295	-	-	**1**	0.295	-	-
S2_C	**1**	0.751	-	-	-	-	**1**	NF
S3_A	**0.872**	0.444	**0.137**	0.01	**1**	0.597	**1**	NF
S3_L	**0.545**	0.041	**0.692**	0.041	-	-	-	-
S3_G	**1**	0.671	-	-	**1**	0.671	-	-
S3_C	**1**	0.501	-	-	-	-	**1**	NF

AUCROC results on the “realistic synthetic” Testbed 3 are shown in Table 12. C-ALLOUT achieves highly competitive performance and significantly outperforms SRA in all cases. Interestingly, type-specific rankings are better than the overall ranking for those specific outliers, which shows that they are effectively specialized. SRA again fails to flag the collective outliers. Results with respect to AUCPR are similar (see Table 13). “NF", or not flagged, cells show that the method was not able to note the existence of that type of outlier described in the name of the dataset

Table 12 – AUCROC performance on Testbed 3 with annotated outliers. Note that in variant _A datasets with all three outlier types, type-specific rankings are evaluated against outliers of the corresponding type only.

Dataset	C-ALLOUT-$\mathscr{R}_o$	SRA	C-ALLOUT-$\mathscr{R}_l$	SRA	C-ALLOUT-$\mathscr{R}_g$	SRA	C-ALLOUT-$\mathscr{R}_c$	SRA
ALOI_A	**1**	0.943	**0.997**	0.825	**1**	0.999	**1**	NF
ALOI_L	**1**	0.861	**1**	0.861	-	-	-	-
ALOI_G	**1**	**1**	-	-	**1**	**1**	-	-
ALOI_C	**1**	**1**	-	-	-	-	**1**	NF
Glass_A	**0.965**	0.848	**0.904**	0.585	**1**	0.866	**1**	NF
Glass_L	**0.905**	0.686	**0.944**	0.686	-	-	-	-
Glass_G	**1**	**1**	-	-	**1**	**1**	-	-
Glass_C	**1**	**1**	-	-	-	-	**1**	NF
skin_A	**0.999**	0.838	**0.997**	0.745	**1**	0.897	**0.997**	NF
skin_L	**0.997**	0.778	**0.999**	0.778	-	-	-	-
skin_G	**1**	0.902	-	-	**1**	0.902	-	-
skin_C	**1**	0.854	-	-	-	-	**1**	NF
smtp_A	**0.999**	0.987	**0.997**	0.960	**1**	0.998	**0.999**	NF
smtp_L	**0.996**	0.679	**0.999**	0.679	-	-	-	-
smtp_G	**1**	0.771	-	-	**1**	0.771	-	-
smtp_C	**1**	0.852	-	-	-	-	**1**	NF
Waveform_A	**1**	0.613	**0.995**	0.494	**1**	0.547	**1**	NF
Waveform_L	**1**	0.495	**1**	0.495	-	-	-	-
Waveform_G	**1**	0.551	-	-	**1**	0.551	-	-
Waveform_C	**1**	0.789	-	-	-	-	**1**	NF

Table 13 – AUCPR performance on Testbed 3 with annotated outliers. Note that in variant _A datasets with all three outlier types, type-specific rankings are evaluated against outliers of the corresponding type only.

Dataset	C-ALLOUT-$\mathscr{R}_o$	SRA	C-ALLOUT-$\mathscr{R}_l$	SRA	C-ALLOUT-$\mathscr{R}_g$	SRA	C-ALLOUT-$\mathscr{R}_c$	SRA
ALOI_A	**0.995**	0.898	**0.281**	0.099	**1**	0.451	**0.865**	NF
ALOI_L	**0.966**	0.294	**0.996**	0.294	-	-	-	-
ALOI_G	**1**	**1**	-	-	**1**	**1**	-	-
ALOI_C	**1**	**1**	-	-	-	-	**1**	NF
Glass_A	**0.877**	0.801	**0.165**	0.063	**1**	0.207	**1**	NF
Glass_L	**0.352**	0.253	**0.506**	0.253	-	-	-	-
Glass_G	**1**	**1**	-	-	**1**	**1**	-	-
Glass_C	**1**	**1**	-	-	-	-	**1**	NF
skin_A	**0.924**	0.233	**0.253**	0.017	**0.977**	0.541	**0.413**	NF
skin_L	**0.647**	0.024	**0.857**	0.024	-	-	-	-
skin_G	**0.983**	0.558	-	-	**0.983**	0.558	-	-
skin_C	**0.983**	0.022	-	-	-	-	**0.983**	NF
smtp_A	**0.959**	0.939	**0.258**	0.142	**0.989**	0.587	**0.481**	NF
smtp_L	**0.794**	0.006	**0.899**	0.006	-	-	-	-
smtp_G	**0.996**	0.258	-	-	**1**	0.258	-	-
smtp_C	**1**	0.173	-	-	-	-	**1**	NF
Waveform_A	**0.997**	0.127	**0.283**	0.006	**0.944**	0.080	**0.963**	NF
Waveform_L	**0.963**	0.007	**0.963**	0.007	-	-	-	-
Waveform_G	**0.963**	0.081	-	-	**0.963**	0.081	-	-
Waveform_C	**0.963**	0.082	-	-	-	-	**0.963**	NF

5.3 Scalability Performance

Figure 11 shows the scalability performance for C-AllOut using increasing random samples of our largest benchmark dataset, Shuttle2 (See Table 2). Reported runtimes are averages of 10 independent runs and they refer to the time taken to compute each ranking ($\mathscr{R}_o$ in yellow, $\mathscr{R}_g$ in red, $\mathscr{R}_l$ in green, and $\mathscr{R}_c$ in black) and to build the Slim-tree (orange). Total time (gray) is the time needed to build the tree plus the time to find all 4 rankings. In line with Lemma 4.2.1, C-AllOut presents $O(n \log n)$ time complexity. The wall-clock time was measured using a machine having a dual-core Intel Xeon CPU E5-2640 v3 @ 2.6GHz processor, 16 GB RAM, running Ubuntu 18.04.2 OS.

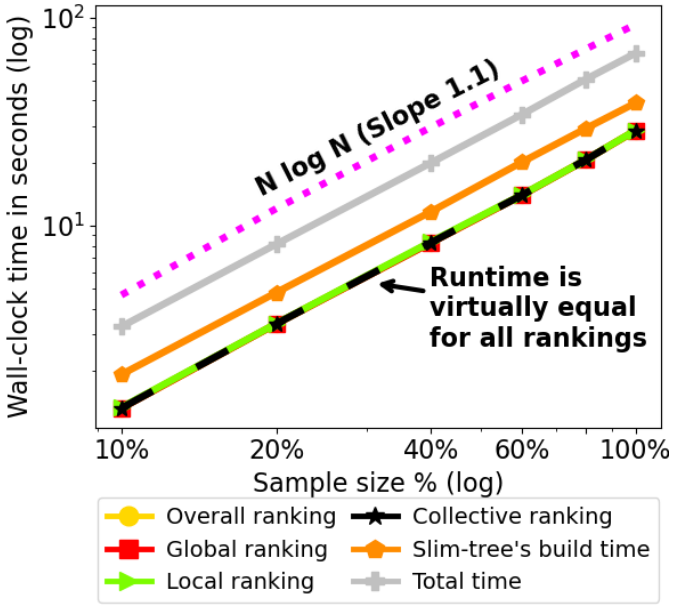


Figure 11 – Wall-clock time (avg. 10 runs) of C-AllOut as the size of random samples taken from Shuttle2 increases.

5.4 C-AllOut at Work: Attention Routing

One of the applications for outlier annotation is to route attention to points of interest. This section shows that C-AllOut can, for instance, automatically identify nature hazards and send helpful coordinates to local fire departments. First, we consider a satellite infrared image showing a large forest fire that spread in Greece, near Athens. A dataset is created based on the image by first dividing it in a 12×12 grid, obtaining 144 image tiles, which become data instances. Then, we extract the average RGB values from each tile, thus the data dimensionality is 3. Figure 12 highlights the overall outliers tiles (yellow) found by C-AllOut, as well as the three specific types of outliers. It is possible to see that the global outliers (red) are extreme image tiles, including those that are singular, such as the water part in the bottom-right corner. The collective ranking identifies 2 groups of outliers (black); one group of fire, and another one of smoke, as highlighted in the image. Meanwhile, the local ranking highlights borderline outliers

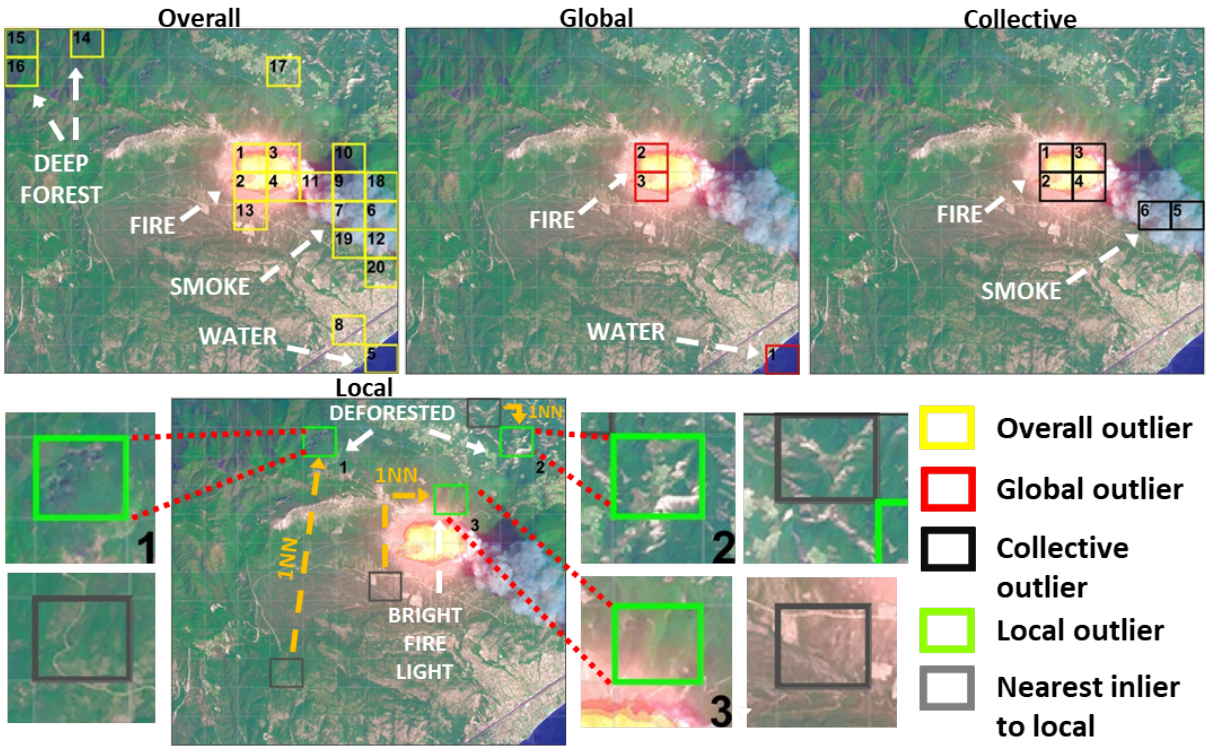


Figure 12 – C-ALLOUT at work to detect a forest fire in Greece. Numbers are outlier positions in rankings.

(green), like deforested zones or tiles with very bright colors. In dark gray are the closest inlier tiles (1NN) to the local outlier tiles. They are highlighted to show how the boundary between some inliers and local outliers can be subtle, making local outliers the hardest challenge in outlier annotation. Since they are borderline inliers, they appear a bit different from the rest of the image.

Using the same methodology, we created another dataset based on an image taken from Tokyo Imperial Palace. The 3 average RGB values constitute the dimensionality of this dataset that has 144 instances. Overall outliers (in yellow) can be seen in Figure 13, along with the specific types of outliers (global in red, local in green, and collective in black). Global ranking highlights the extreme tiles in the image: two structures with a blue shade (Imperial Palace and the dome) and a brown field. Collective ranking identifies three groups of outliers: two structures with a blue shade, two portions of the river around the green area and two tiles having sparse city blocks. Finally, the local ranking focuses on borderline outliers. It highlights image tiles where the river shares the tile with vegetation or city blocks, and another tile with vegetation and the Imperial Palace sharing space. Orange tiles are the closest inlier tiles (1NN) to the local outlier tiles. In this scenario, C-ALLOUT can help to find touristic places for people visiting a city.

5.5 Final Considerations

This chapter evaluated C-ALLOUT, our proposed method and main contribution of this work. We were able to show through our experiments using Testbed 1 that C-ALLOUT is

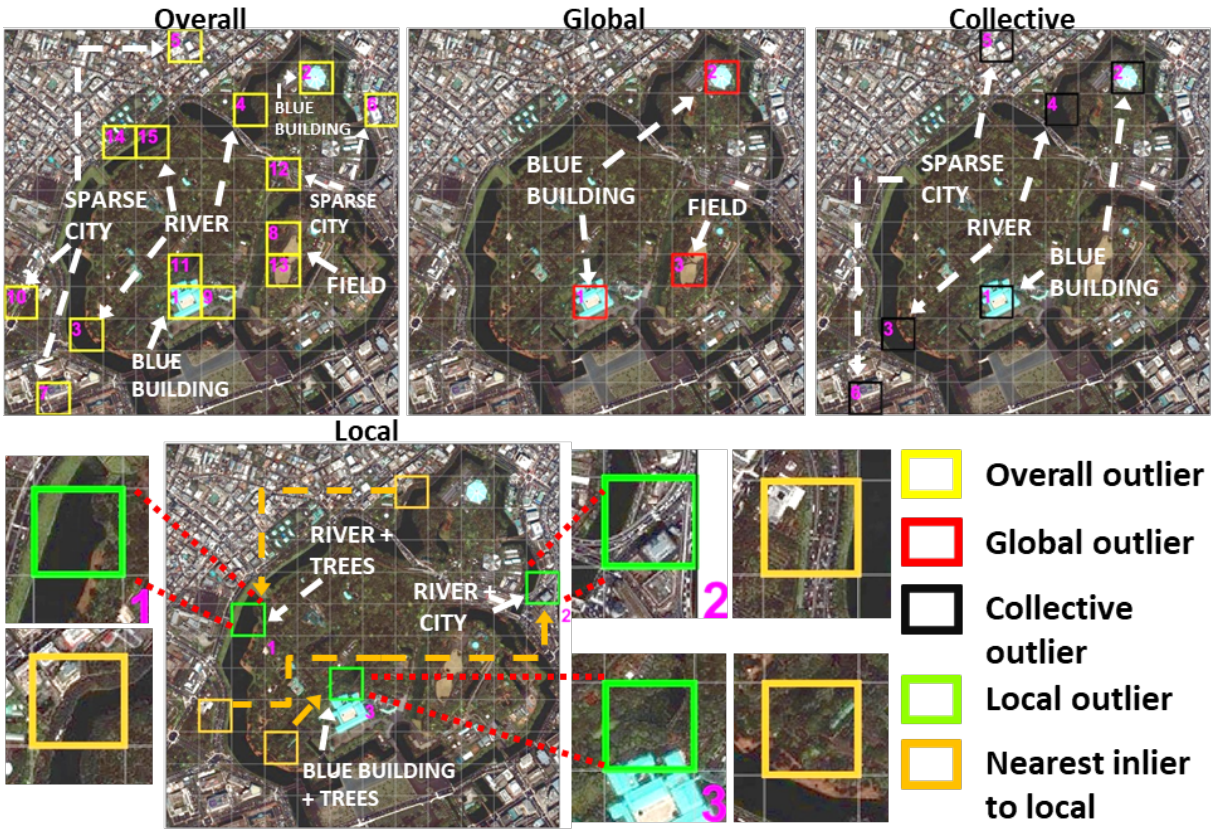


Figure 13 – C-ALLOUT at work to route attention to touristic places in Tokyo Imperial Palace. Numbers are outlier positions in rankings.

capable of maintaining outlier detection quality comparable, if not better, than state-of-the-art competitors. We were also able to show that the outlier annotation problem that we defined is feasible and can be performed using an indexing structure like the Slim-tree. Our results on outlier annotation datasets in Testbed 2 and 3 expose that C-ALLOUT is very much superior to SRA, the algorithm that comes the closest, to the best of out knowledge, to work in the outlier annotation task. Not only that, but through Testbed 3, we were able to show that C-ALLOUT can work with realistic datasets. Realistic datasets are not purely synthetic, they follow the distributions of real data.

The dataset generators for synthetic and realistic data were also discussed in this chapter. We provided details for them and algorithms to guide their routines. The empirically defined parameters used in dataset generation were also showcased to make the generation process more transparent. Our results, specially the ones related to SRA, display that our data generation process is sound and can work with other algorithms besides C-ALLOUT.

On the next chapter we conclude this monograph citing the major contributions of this MSc work. Limitations and difficulties are also discussed and future work is enlightened.

CHAPTER

6

CONCLUSION

In this work, we investigated the outlier detection techniques provided by the literature. Even though having many different solutions for the outlier detection problem, a gap was identified as there were no algorithms capable of categorizing outliers per type. We explored this gap by developing C-ALLOUT, a solution capable of both detecting and annotating outliers. We showed in our experiments that C-ALLOUT is capable of keeping up with state-of-the-art outlier detectors, detecting multiple types of outliers at once and annotating outliers by type as well. Code, datasets and generators developed and used in this work are all available at <https://bit.ly/3JezWe4>. Our main contributions with this work are:

C1: Outlier Annotation: We present definitions for a new problem in outlier mining that can fill an important gap in the literature. Going beyond, we also provide the C-ALLOUT solution to the problem. We settle this first step while providing ranking intuitions and definitions that can help other works in the future;

C2: Scalability: Having both time and space complexity of $O(n \log n)$, according to the proofs presented, we grant a scalable outlier detector and annotator fit to be applied in resource demanding tasks and large datasets;

C3: Explainability: Our rankings have their intuitions proven to be useful and sound by our experiments. These intuitions provide an explainable take on the outlier detection and annotation task and C-ALLOUT's outlierness scores are easily explained for any point of interest;

C4: Simplicity: We developed C-ALLOUT to act on a parameter-free basis, not requiring any input or fine-tuning from the user, making the outlier mining task really unsupervised;

C5: Benchmark Experiments, Datasets and Generators: Using 20 popular outlier detection benchmark datasets with distinct sizes and dimensionalities, we arrange a thorough

comparison across different state-of-the-art algorithms and C-ALLOUT. To be able to evaluate our proposed method, we also developed synthetic and realistic dataset generators being empirically tested to form inliers and outliers labeled by type. We disclose to the community the generators and outlier annotation datasets generated, for public use.

The contents of this MSc work were compiled into a full paper, submitted as:

- Guilherme D. F. Silva, Leman Akoglu, Robson L. F. Cordeiro. C-AllOut: Catching and Calling Outliers By Type. European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases. ECML PKDD 2022. (under revision)

6.1 Limitations and Difficulties

Some difficulties came along during the development of this work which limited our development and took time from our schedule. From the varied list of issues we faced during this MSc work, we highlight the following:

- Since outlier annotation is a novel task, lots of concerns were unveiled and engaged for the first time. No real or synthetic datasets having outlier type annotations were available while this research was carried. This resulted in us having to devise dataset generators to make possible for us to test our algorithms. As we did not want to limit our work dealing only with synthetic data, we had to go further and develop generators that could create realistic datasets. In this way, even if we did not have any domain expert to label our datasets, we were able to synthetically create datasets that would follow the patterns of real ones, as discussed in Chapter 5.

- Another difficulty that the novelty of our work has brought to us was the nonexistence of baseline methods for the task. Having no previous work to base ourselves upon, we had to come up with a ranking method that would, to the best of our knowledge, for the first time, objectively put certain outliers above others, given their type. The lack of competition has limited our experiments, since we do not have any algorithm to compete directly. SRA turns up as the only method capable of returning some sort of outlier annotation, but its limitations made it not a good competitor to C-ALLOUT.

- Midway through this project, while we were focusing on outlier detection in text data, we noticed that C-ALLOUT capabilities were not being used to their full potential. The possibility of annotating outlier types and the contributions that this could bring to the data mining field made us favor a change of plans so that we could exploit C-ALLOUT potential and make it more visible, exceeding the text processing eyes.

6.2 Future Work

We still think that C-ALLOUT's full potential is not reached yet and future works will help us get there. The following list presents some interesting investigations that could go further and find interesting results:

- The Slim-tree gave us many opportunities to extract knowledge from datasets, just working with its original structure. Since we see many "forest" algorithms dominating supervised (Random Forests) and unsupervised learning (Isolation Forests), we think that a group of Slim-trees would probably be able to extract even more knowledge from data.
- There are good intuitions behind each one of our scores, and many of them came along during the course of this work. Other good intuitions may come from understanding the Slim-tree, and other indexing structures, which could result in new scores or modified versions of the ones we proposed.
- Understanding other indexing structures will probably not only provide good intuitions but also check if our proposed scores can also work with them, maybe achieving better results. Since Slim-tree has a fair share of concept overlapping with hierarchical clustering, many parallels and extensions can come from thorough analyses of this connection.

BIBLIOGRAPHY

AGGARWAL, C. C. Outlier analysis. In: SPRINGER. **Data mining**. [S.l.]: Springer, 2015. p. 237–263. ISBN 978-1-4614-6396-2. Citations on pages 34 and 51.

AGGARWAL, C. C.; YU, P. S. Outlier detection for high dimensional data. In: **SIGMOD**. [S.l.: s.n.], 2001. p. 37–46. Citation on page 42.

AJITHA, P.; CHANDRA, E. A survey on outliers detection in distributed data mining for big data. **Journal of Basic and Applied Scentific Research**, v. 5, n. 2, p. 31–38, 2015. Citation on page 34.

ALESSANDRETTI, L.; ELBAHRAWY, A.; AIELLO, L. M.; BARONCHELLI, A. Anticipating cryptocurrency prices using machine learning. **Complexity**, Hindawi, v. 2018, 2018. Citation on page 28.

ANGIULLI, F.; PIZZUTI, C. Fast outlier detection in high dimensional spaces. In: SPRINGER. **European conference on principles of data mining and knowledge discovery**. [S.l.], 2002. p. 15–27. Citation on page 38.

BANDARAGODA, T. R.; TING, K. M.; ALBRECHT, D.; LIU, F. T.; ZHU, Y.; WELLS, J. R. Isolation-based anomaly detection using nearest-neighbor ensembles. **Computational Intelligence**, v. 34, n. 4, 2018. Citations on pages 27 and 52.

BARADARAN, N.; REDDY, A.; THAKUR, R. S. **Anomaly detection with k-means clustering and artificial outlier injection**. [S.l.]: Google Patents, 2017. US Patent App. 14/927,553. Citation on page 40.

BARNETT, V.; LEWIS, T. **Outliers in statistical data**. [S.l.]: Wiley, 1974. Citation on page 34.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. **Communications of the ACM**, ACM New York, NY, USA, v. 18, n. 9, p. 509–517, 1975. Citation on page 57.

BINDU, P.; THILAGAM, P. S.; AHUJA, D. Discovering suspicious behavior in multilayer social networks. **Computers in Human Behavior**, Elsevier, v. 73, p. 568–582, 2017. Citation on page 33.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001. Citation on page 55.

BREUNIG, M. M.; KRIEGEL, H.-P.; NG, R. T.; SANDER, J. Lof: identifying density-based local outliers. In: **Proceedings of the 2000 ACM SIGMOD international conference on Management of data**. New York, NY, USA: Association for Computing Machinery, 2000. v. 29, n. 2, p. 93–104. ISSN 0163-5808. Citations on pages 27, 29, 42, 51, 52, 57, 58, 59, and 78.

CABRAL, E. F.; CORDEIRO, R. L. Fast and scalable outlier detection with sorted hypercubes. In: **Proceedings of the 29th ACM International Conference on Information & Knowledge Management**. [S.l.: s.n.], 2020. p. 95–104. Citation on page 33.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. **ACM computing surveys (CSUR)**, ACM, v. 41, n. 3, p. 15, 2009. Citations on pages 34, 37, and 39.

CIACCIA, P.; PATELLA, M.; RABITTI, F.; ZEZULA, P. Indexing metric spaces with m-tree. In: **SEBD**. [S.l.: s.n.], 1997. v. 97, p. 67–86. Citation on page 47.

CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: CITESEER. **Proceedings of the 23rd VLDB conference, Athens, Greece**. [S.l.], 1997. p. 426–435. Citation on page 47.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine learning**, Springer, v. 20, n. 3, p. 273–297, 1995. Citation on page 59.

CRIMINISI, A. **Machine learning for medical images analysis**. [S.l.]: Elsevier, 2016. Citation on page 28.

CZEKALSKI, S.; MORZY, M. Similarity forests revisited: A swiss army knife for machine learning. In: **PAKDD**. [S.l.: s.n.], 2021. p. 42–53. Citations on pages 55 and 78.

DANG, X. H.; MICENKOVÁ, B.; ASSENT, I.; NG, R. T. Local outlier detection with interpretation. In: SPRINGER. **ECML PKDD**. [S.l.], 2013. p. 304–320. Citation on page 51.

DURAJ, A.; CHOMATEK, L. Supporting breast cancer diagnosis with multi-objective genetic algorithm for outlier detection. In: SPRINGER. **International Conference on Diagnostics of Processes and Systems**. [S.l.], 2017. p. 304–315. Citation on page 34.

ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; XU, X. *et al.* A density-based algorithm for discovering clusters in large spatial databases with noise. In: **Kdd**. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231. Citations on pages 29 and 39.

FIGUEROA-GARCÍA, J. C.; MELGAREJO-REY, M. A.; HERNÁNDEZ-PÉREZ, G. Representation of the minkowski metric as a fuzzy set. **Optimization Letters**, Springer, p. 1–14, 2018. Citation on page 36.

FIX, E.; HODGES, J. L. Discriminatory analysis. nonparametric discrimination: Consistency properties. **International Statistical Review/Revue Internationale de Statistique**, JSTOR, v. 57, n. 3, p. 238–247, 1989. Citation on page 56.

GOLDSTEIN, M.; UCHIDA, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. **PloS one**, v. 11, n. 4, 2016. Citation on page 65.

GOROKHOV, O.; PETROVSKIY, M.; MASHECHKIN, I. Convolutional neural networks for unsupervised anomaly detection in text data. In: SPRINGER. **International Conference on Intelligent Data Engineering and Automated Learning**. [S.l.], 2017. p. 500–507. Citation on page 53.

GRAVE, E.; MIKOLOV, T.; JOULIN, A.; BOJANOWSKI, P. Bag of tricks for efficient text classification. In: **Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers**. [s.n.], 2017. p. 427–431. Available: <https://www.aclweb.org/anthology/E17-2068/>. Citation on page 58.

HAMMER, P. **Adaptive control processes: a guided tour (R. Bellman)**. [S.l.]: Society for Industrial and Applied Mathematics, 1962. Citation on page 41.

HAN, J.; KAMBER, M.; PEI, J. Outlier detection. **Data mining: Concepts and Techniques**, Morgan Kaufmann, 2012. Citations on pages 27 and 51.

HAWKINS, D. M. **Identification of outliers**. [S.l.]: Springer, 1980. Citation on page 34.

HE, Z.; XU, X.; DENG, S. Discovering cluster-based local outliers. **Pattern Recognition Letters**, Elsevier, v. 24, n. 9-10, p. 1641–1650, 2003. Citation on page 40.

JAIN, A. K.; DUBES, R. C. Algorithms for clustering data. **Englewood Cliffs: Prentice Hall, 1988**, 1988. Citation on page 39.

JUNIOR, A. G. B.; CORDEIRO, R. L. F. Fast and scalable outlier detection with metric access methods. In: SPRINGER. **International Conference on Computational Science**. [S.l.], 2019. p. 189–203. Citations on pages 15, 62, and 63.

JUNIOR, A. G. B.; CORDEIRO, R. L. F. Fast and scalable outlier detection with metric access methods. In: SPRINGER. **International Conference on Computational Science**. [S.l.], 2019. p. 189–203. Citation on page 33.

KEERTHI, S. S.; LIN, C.-J. Asymptotic behaviors of support vector machines with gaussian kernel. **Neural computation**, mit press, v. 15, n. 7, p. 1667–1689, 2003. Citation on page 61.

KNORR, E. M.; NG, R. T. Algorithms for mining distance-based outliers in large datasets. In: **VLDB**. [S.l.: s.n.], 1998. v. 98, p. 392–403. Citations on pages 37 and 51.

KRIEGEL, H.-P.; KRÖGER, P.; SCHUBERT, E.; ZIMEK, A. LoOP: local outlier probabilities. In: **CIKM**. [S.l.: s.n.], 2009. p. 1649–1652. Citation on page 51.

KRIEGEL, H.-P.; SCHUBERT, M.; ZIMEK, A. Angle-based outlier detection in high-dimensional data. In: **Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.]: ACM, 2008. p. 444–452. Citations on pages 27, 41, and 51.

KRUSKAL, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. **Proceedings of the American Mathematical society**, JSTOR, v. 7, n. 1, p. 48–50, 1956. Citation on page 47.

LEYS, C.; KLEIN, O.; DOMINICY, Y.; LEY, C. Detecting multivariate outliers: Use a robust variant of the mahalanobis distance. **Journal of Experimental Social Psychology**, Elsevier, v. 74, p. 150–156, 2018. Citation on page 36.

LI, S.; LEE, R.; LANG, S.-D. Mining distance-based outliers from categorical data. In: IEEE. **Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)**. [S.l.], 2007. p. 225–230. Citation on page 38.

LIU, F. T.; TING, K. M.; ZHOU, Z.-H. Isolation forest. In: **ICDM**. [S.l.: s.n.], 2008. p. 413–422. Citations on pages 17, 33, 51, 53, 54, 55, and 78.

MALINI, N.; PUSHPA, M. Analysis on credit card fraud identification techniques based on knn and outlier detection. In: IEEE. **2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)**. [S.l.], 2017. p. 255–258. Citation on page 36.

MOONESINGHE, H.; TAN, P.-N. Outlier detection using random walks. In: IEEE. **ICTAI**. [S.l.], 2006. Citations on pages 27 and 52.

NIAN, K.; ZHANG, H.; TAYAL, A.; COLEMAN, T.; LI, Y. Auto insurance fraud detection using unsupervised spectral ranking for anomaly. **The Journal of Finance and Data Science**, v. 2, n. 1, p. 58–75, 2016. Citations on pages 17, 28, 31, 42, 51, 63, 64, and 84.

OMOHUNDRO, S. M. **Five balltree construction algorithms**. [S.l.]: International Computer Science Institute Berkeley, 1989. Citation on page 57.

PAPADIMITRIOU, S.; KITAGAWA, H.; GIBBONS, P. B.; FALOUTSOS, C. LOCI: Fast outlier detection using the local correlation integral. In: IEEE. **ICDE**. [S.l.], 2003. Citations on pages 27, 38, and 52.

PARK, J.; SANDBERG, I. W. Universal approximation using radial-basis-function networks. **Neural computation**, MITP, v. 3, n. 2, p. 246–257, 1991. Citation on page 53.

PATRICK, E. A.; III, F. P. F. A generalized k-nearest neighbor rule. **Information and control**, Elsevier, v. 16, n. 2, p. 128–152, 1970. Citation on page 38.

PHAM, N.; PAGH, R. A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data. In: ACM. **Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2012. p. 877–885. Citation on page 41.

RAMASWAMY, S.; RASTOGI, R.; SHIM, K. Efficient algorithms for mining outliers from large data sets. In: **Proceedings of the 2000 ACM SIGMOD international conference on Management of data**. New York, NY, USA: Association for Computing Machinery, 2000. v. 29, n. 2, p. 427–438. ISSN 0163-5808. Citations on pages 29, 38, 51, 57, and 78.

ROSENBERG, A.; HIRSCHBERG, J. V-measure: A conditional entropy-based external cluster evaluation measure. In: **Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)**. [S.l.: s.n.], 2007. p. 410–420. Citation on page 52.

SATHE, S.; AGGARWAL, C. C. Similarity forests. In: **KDD**. [S.l.: s.n.], 2017. p. 395–403. Citations on pages 55 and 78.

SATOPAA, V.; ALBRECHT, J.; IRWIN, D.; RAGHAVAN, B. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In: **ICDCS Workshops**. [S.l.: s.n.], 2011. Citation on page 73.

SCHöLKOPF, B.; WILLIAMSON, R. C.; SMOLA, A. J.; SHAWE-TAYLOR, J.; PLATT, J. C. Support vector method for novelty detection. In: **NIPS**. [S.l.: s.n.], 1999. Citations on pages 51, 60, and 78.

SHAWE-TAYLOR, J.; ŽLIČAR, B. Novelty detection with one-class support vector machines. In: **Advances in Statistical Models for Data Analysis**. [S.l.]: Springer, 2015. p. 231–257. Citation on page 36.

SHI, K.; LI, L. High performance genetic algorithm based text clustering using parts of speech and outlier elimination. **Applied intelligence**, Springer, v. 38, n. 4, p. 511–519, 2013. Citation on page 36.

STEINBUSS, G.; BÖHM, K. Benchmarking unsupervised outlier detection with realistic synthetic data. **ACM TKDD**, v. 15, n. 4, p. 1–20, 2021. Citations on pages 82 and 83.

STEINLEY, D. K-means clustering: a half-century synthesis. **British Journal of Mathematical and Statistical Psychology**, Wiley Online Library, v. 59, n. 1, p. 1–34, 2006. Citations on pages 29 and 39.

TRAINA, C.; TRAINA, A.; FALOUTSOS, C.; CORDEIRO, R. **Arboretum**. 2000. Available: <https://bitbucket.org/gbdi/arboretum/src/master/>. Citation on page 45.

TRAINA, C.; TRAINA, A.; FALOUTSOS, C.; SEEGER, B. Fast indexing and visualization of metric data sets using slim-trees. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 14, n. 2, p. 244–260, 2002. Citations on pages 15, 17, 43, 48, and 49.

TRAINA, C.; TRAINA, A.; SEEGER, B.; FALOUTSOS, C. Slim-trees: High performance metric trees minimizing overlap between nodes. In: SPRINGER. **International Conference on Extending Database Technology**. [S.l.], 2000. p. 51–65. Citations on pages 29, 36, 43, 47, 48, and 62.

TRIPATHI, D.; LONE, T.; SHARMA, Y.; DWIVEDI, S. Credit card fraud detection using local outlier factor. **International Journal of Pure and Applied Mathematics**, v. 118, n. 7, p. 229–234, 2018. Citations on pages 33 and 34.

WALKOWIAK, T.; DATKO, S.; MACIEJEWSKI, H. Utilizing local outlier factor for open-set classification in high-dimensional data-case study applied for text documents. In: SPRINGER. **Proceedings of SAI Intelligent Systems Conference**. [S.l.], 2019. p. 408–418. Citation on page 58.

WEI, L.; QIAN, W.; ZHOU, A.; JIN, W.; YU, J. X. Hot: Hypergraph-based outlier test for categorical data. In: SPRINGER. **Pacific-Asia Conference on Knowledge Discovery and Data Mining**. [S.l.], 2003. p. 399–410. Citation on page 38.

WU, S.; WANG, S. Information-theoretic outlier detection for large-scale categorical data. **IEEE transactions on knowledge and data engineering**, IEEE, v. 25, n. 3, p. 589–602, 2011. Citation on page 36.

YIN, J.; WANG, J. A model-based approach for text clustering with outlier detection. In: IEEE. **2016 IEEE 32nd International Conference on Data Engineering (ICDE)**. [S.l.], 2016. p. 625–636. Citation on page 52.

ZHANG, K. Z.; BENYOUCEF, M. Consumer behavior in social commerce: A literature review. **Decision Support Systems**, Elsevier, v. 86, p. 95–108, 2016. Citation on page 28.

ZHOU, C.; PAFFENROTH, R. C. Anomaly detection with robust deep autoencoders. In: ACM. **Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. [S.l.], 2017. p. 665–674. Citations on pages 61, 78, and 79.


ICMC
USP
SÃO CARLOS