POLITECNICO DI TORINO

POLITECNICO DI TORINO, EURECOM

Master degree course in COMMUNICATIONS AND COMPUTER NETWORKS ENGINEERING

Master Degree Thesis

Clustering of Categorical Data for Anonymization and Anomaly Detection

Supervisors

prof. Elena Baralis

prof. Paolo Papotti

Candidate

Riccardo Cappuzzo

matricola: 231643

Internship Tutor

Anderson Santana De Oliveira

Anno accademico 2017-2018

Contents

Abstract

The field of data analysis has exploded in recent years thanks to the huge wealth of information that can be collected through the Internet and other similar ways of gathering data. Machine learning is often employed to study this data, as well as for modeling and predicting future behaviors in many different fields. One of the techniques employed when performing machine learning is clustering: grouping objects such that objects that belong to a group are closer to members of that group compared to members of other groups. To perform clustering, it is necessary to use a distance function able to assign points to clusters: this is especially problematic when categorical data is involved. Algorithms suitable to perform clustering of categorical data are available, but most implementations are not efficient when working with large datasets. ROCK (RObust Clustering using linKs) avoids the need for a global distance matrix by introducing a step called "labeling", this allows achieving a good clustering accuracy by using only a sample of the complete data set, which in turns makes tackling larger datasets possible. ROCK, however, has some drawbacks of its own: it performs poorly when records in a dataset are very homogeneous or most features are numeric, it is fairly slow and estimating of the number of clusters is hard to do. With my contributions, I fixed most of these issues: to help the algorithm with handling homogeneous data I designed new versions of $f(\theta)$, a function that is used as a criterion for choosing the best clusters to merge; this allowed me to prevent ROCK from generating very large clusters and improved the execution time in some cases, while improving the results in general. To estimate the number of clusters I introduced a new termination condition, GDSTOP, that stops the execution once further merge operations would worsen the result (even if they could still be performed). Parallelization can be implemented in some regions of the code to remove some of the constraints on memory and time. I also implemented a hierarchical distance, trying to preserve more information when features have some semantic similarity that can be accounted for. I tested the algorithm's performances when clustering different datasets and under two more use cases: anomaly detection and anonymization. For what concerns the first, the results were unsatisfactory for various reasons, the datasets I used being one of them: finding suitable data is a very hard task. For the latter, instead, I used ROCK as a preprocessing step in an algorithm that applies k-anonymity to categorical data. This was done with the objective of achieving a higher utility than traditional algorithms: anonymization is performed on a cluster-by-cluster basis (rather than considering the entire dataset) to reduce the number of transformations needed to achieve anonymity.Here the algorithm performed quite well: the utility loss was not as pronounced as with other algorithms and the anonymity property was preserved despite that.

Chapter 1

Introduction

1.1 Motivation

The huge growth of data analysis recent years, due to the immense wealth of information that can be collected thanks to the diffusion of the Internet and other similar ways of gathering data, made accessing personal information much easier than before. Businesses are becoming more and more interested in the field as it becomes evident how having some insight into customer trends allows having a better idea of the future course of action. Predictive analysis and machine learning are widely used for studying this data, generating models and predicting future behaviors.

Machine learning has become increasingly prevalent in recent days thanks to the development of platforms able to handle Big Data, and to its efficiency in extracting patterns from the huge "data lakes" that companies and agencies generate. One of the techniques employed when performing machine learning is *clustering*: grouping objects such that objects that belong to a group are more similar to the members of that group compared to members of other groups. Clustering is employed in a wide range of disciplines and applications, thanks to the fact that it can be performed without supervision, so that results may be produced without the need for an analyst to train the algorithm beforehand. In clustering, a core concept in common among many algorithms is the idea of distance between items: indeed, to assign a point to a cluster rather than another, it is necessary to find what are the closest points to it according to some kind of measure. Be it hierarchical, density-based or centroid-based, most clustering algorithms will employ some kind of distance function to discriminate between clusters. This is a major issue when it is necessary to perform analysis of categorical data, i.e. data that assumes a finite number of possible values (such as names): finding the distance between point $(0, 3)$ and point $(2, 4)$ is different from finding the distance between the term "married" and the term "divorced". Unfortunately, datasets that contain this kind of *mixed data* are extremely widespread and cannot be handled trivially by using traditional distances such as Euclidean distance or Manhattan distance: this prevents most "standard" clustering algorithms from being used in such cases.

To cluster categorical data, two different approaches may be used: either categorical features are encoded (e.g. using one-hot encoding) or suppressed, or algorithms able

to handle generalized metrics must be employed. Intuitively, the first approach is quite impractical since dropping features causes an irreversible and maybe very significant loss of information. For what concerns categorical data, a critical problem is caused by the presence of IDs or similar features that are mostly unique (but may nonetheless assume a wide range of values) and that may be very important during the analysis: these features cannot be encoded because this would increase exponentially the number of features to consider, nor can they be treated as integers since two consecutive IDs should in principle be as far from each other as any two other IDs in the set. Many algorithms able to handle categorical data, thus solving the issues described above, were suggested. Unfortunately, their public implementations fail at clustering some datasets because they often require a complete distance matrix: when the number of records is too large, this approach becomes too costly from the memory and time point of view.

ROCK [15] mitigates this issue by introducing a step called "labeling", instead of using a global distance matrix. This allows achieving a good clustering accuracy by using only a sample of the complete dataset, thus allowing to tackle much bigger datasets than what can be handled by comparable algorithms of public use, such as Affinity Propagation [14].

My search for an algorithm able to cluster categorical data was initially motivated by the idea of employing such a clustering algorithm to improve the utility of data anonymized using k-anonymity. *k-anonymity* is a very widely employed method for performing personal data anonymization, but has some very important drawbacks: firstly, it is not secure since records can easily be de-anonymized (this point will be described more thoroughly in later sections); secondly, to enforce the k-anonymity property (each record must be indistinguishable from at least $k-1$ other records in the dataset) irreversible transformations must be performed and this worsens noticeably the utility of the dataset for data mining purposes. My idea was employing a clustering algorithm to reduce this loss in utility: anonymization is applied on a cluster basis since records in each cluster should be close enough to each other that the number of transformations needed to reach k-anonymity should not be so large that the quality would degrade too much.

Another idea that I considered was performing *anomaly detection* using ROCK: categorical data remains a major issue in this field since strings and nominal attributes are present in most log files. This requires data exploration to understand whether such fields can be dropped or, if this is not possible, they must be encoded in a suitable way for algorithms that rely on numerical features to work. This approach is often impractical due to the fact that encoding (e.g. one-hot) of categorical attributes leads to an explosion in the number of dimensions of the dataset.

Finally, to improve clustering performances in an environment that can be generalized I tested a *Hierarchical distance*, i.e. a distance function able to take into consideration the semantic distance present between two records. This produces good results and is especially useful when considering the anonymization case, in which generalization, together with suppression, is one of the two possible transformations.

1.2 State of the art

Clustering The subject of clustering has been widely studied in the literature, and the large number of proposed clustering algorithms sparked the need for surveys such as [13], or [41]. Their aim was understanding the criteria that should be taken into consideration when choosing the correct algorithm for an application. Clustering is an important technique to perform machine learning and has the advantage of being completely unsupervised: a clustering algorithm does not require intervention from the user to perform its task, and can be used as a primer for further machine learning application, or may produce results that can be analyzed immediately.

Clustering methods can be categorized depending on the cluster model, i.e. the "idea" that an algorithm has of how a cluster should be, and how points should be assigned to a cluster. Depending on the cluster model, clustering algorithms may roughly be split into four categories: *connectivity-based clustering*, where it is assumed that the closer two points are to each other, the more related they are; *centroid-based clustering*, where each cluster is represented by a centroid and each point in the dataset will be assigned to the cluster with the closest centroid (k-means is one such algorithm); *distribution-based clustering* where clusters are defined as objects that belong to the same probability distribution; *density-based clustering* where clusters are defined as regions where the density of points is larger than that in the remainder of the dataset [12].

In my work, I started by focusing on plain clustering of categorical data, so I searched for publicly available libraries able to perform this operation: I tried to employ k-means by using one-hot encoding, but this approach was not practical in most cases. I found some implementations of PAM [3], that didn't behave as well as I hoped. Affinity Propagation [14] was also available online. All these implementations were not successful for different reasons: some accepted only numerical data, even though the algorithm is in principle able to employ any kind of data; most implementations required a complete distance matrix, which is impractical for very large sets.

Since one of my objectives was performing clustering of large datasets, I initially chose ROCK [15] because it was the only algorithm that provided the concept of *labeling*, and that was thus able to handle much larger datasets than what could be possible by using only the distance matrix. Some improvements to ROCK have been developed after the original algorithm was proposed, such as QROCK [10] and IROCK [33]. The first one, QROCK (from Quick ROCK) was developed to be a much quicker version of ROCK (hence the name), and it achieves this result by removing one of the phases of the algorithm (this point will be described more thoroughly in Section 2.2.3), but has the very important drawback of producing worse clusters when records are not well distinct. This case, unfortunately, happens fairly frequently so I decided to go back to the original version of ROCK. For what concerns IROCK, instead, the only change that the paper applies to ROCK is employing a link matrix (see Section 2.3) that contains real numbers instead of integers: this was necessary because the paper employed the algorithm to study documents and having a real matrix was necessary for the novel distance function that was introduced there. For my purposes, switching from an integer-based link matrix to a real-based one didn't yield noticeable results so, in this case, I decided to use the original approach simply because

this would allow me to compare the results to what was produced by the original ROCK.

Hierarchical distance Very often, categorical data has some kind of semantics intrinsic in some or all its features: this means that different records may share some degree of similarity that goes beyond the "same value/different value" approach that is often taken with measures such as the Cosine Similarity or the Jaccard Coefficient. When considering personal data, a good example would be the Education level: two people that have a Bachelor's and a Master's degree are intuitively closer to each other than to someone who didn't go beyond a High School Diploma. The idea of taking this information into consideration by means of a *hierarchical distance* has been considered in other papers (e.g. [25], [32]); its application in this environment, however, should be quite novel (particularly for what concerns anonymization). The obvious drawback of this approach is that it clearly does not apply to all possible categorical attributes: city names, for example, cannot be generalized by means of a simple semantic classification; while IDs and names cannot be generalized at all.

Anonymization Anonymization is another field that is becoming more and more pervasive in recent days, due to the huge amount of personal information available online. Companies very often have a huge wealth of information that, if mined, would yield even more valuable data that may help driving business (e.g. advertisements aimed at a certain category of customers, depending on personal information [17]). Just as often, however, companies do not have the means (or the competences) to perform meaningful data mining on data and have thus to publicly release the data they have in competitions or provide it to specialized agencies that can perform data analysis. Given that data is strictly personal, and that sensitive attributes may be contained in the data (e.g. race, religion, healthcare information), legislative bodies moved to produce regulations to protect this data (such as the EU GDPR [30]). Data protection implies performing anonymization of records so that they cannot be re-identified after being released. [9] summarizes well what the different techniques used are, their advantages and drawbacks. Nowadays, k-anonymity [35] is, together with differential privacy [11], one of the main methods used to anonymize data so that records cannot be re-identified by an attacker. k-anonymity is heavily flawed for multiple reasons both from the security [20], [36] and the utility [5] points of view. Due to lack of time, I wasn't able to devise a completely novel anonymization algorithm, so what I attempted to do was improving the second problem by following the idea suggested in [34], where k-anonymity was implemented after a clustering operation was performed: this allows applying anonymization on a cluster-by-cluster basis instead of considering the entire dataset to build equivalence classes.

Anomaly detection Finally, for what concerns anomaly detection, [8] is a very good survey of different anomaly detection techniques and the measures that can be employed in this field. According to it, multiple different techniques can be used to perform anomaly detection: classification, nearest neighbor, statistical analysis, information theoretic analysis, spectral analysis are all, together with clustering, valid techniques to perform anomaly detection. The latter is advantageous because it is unsupervised and it doesn't require

a long time to perform observations. It has, however some important drawbacks: clustering algorithms often are not optimized to perform anomaly detection, in some cases anomalies may be merged with normal data and more importantly, anomaly detection using clustering fails if anomalies form significant clusters by themselves (this happened in some situation, as I will describe briefly in Chapter 5). Very often, before performing anomaly detection, Exploratory Data Analysis (EDA) is carried out on a dataset of interest to extract additional useful features and to drop unimportant ones (see [26], [18]). I attempted to use ROCK after performing only a lightweight EDA phase during which I dropped useless fields because I was interested in finding out if it would be able to produce good results with an almost "stock" dataset (further discussion will be reported in Section 4.3). The reason why I tried to use ROCK to perform anomaly detection is that it has the interesting property of leaving points out of clusters if they're too dissimilar from all formed groups to be merged with one of them.

Accuracy measures To test my results, I employed different quality measures depending on what was the purpose of the trial and on the application. When performing general clustering (with the aim of classifying records) I employed Homogeneity, Completeness and V-measure [31], as well as Adjusted Mutual Information [37]. When possible, I did the same when performing anomaly detection, and when the measures were not applicable I reported other results. Finally, for the anonymization part, the literature contains a plethora of measures for assessing the risk [38]. Among them, I chose two measures that would be able to describe the re-identification risk as well as the utility loss caused by the transformation: they are, respectively, the Estimated number of Correct Matches [7] and the Global Certainty Penalty [40].

1.3 Table of contents

In this document, Chapter 1 contains the motivations behind my study and a description of the state of the art in the field of clustering and data mining. Chapter 2 describes the algorithm I chose to perform my studies, how it is prepared, how it executes and the observations I made on its original implementation. Chapter 3 contains my contributions: a method for designing $f(\theta)$, a new termination condition that allows the algorithm to stop the execution once the cluster quality is too low, new labeling techniques, studies on parallelization and a hierarchical distance that allows taking into consideration the semantic properties of data that can be generalized in a hierarchy. Chapter 4 describes the applications I considered for the algorithm, i.e. standard clustering, anonymization and anomaly detection. Finally, Chapter 5 reports the results I've achieved and Chapter 6 wraps them up together with my conclusions and suggestions for future work.

Chapter 2

ROCK

Here, I will describe how the ROCK (RObust Clustering using linKs) [15] algorithm proceeds, its execution phases and the functions it employs. Observations on complexity and choice of parameters will also be made in this section.

2.1 Overview of ROCK

Typically, clustering can be classified in *partitional* or *hierarchical*: algorithms that perform partitional clustering subdivide the entire space of points in a fixed number of subsets, points are then assigned to one according to their distance from the centroid of the subset; hierarchical algorithms execute clustering by partitioning clusters or merging points until a certain number of groups has been created (i.e. when the desired height on the dendrogram has been reached) Partitional algorithms have the major drawback of requiring the number of clusters before executing the clustering operation: finding this number is a problem on its own, which is usually solved by using the *elbow method*, or similar procedures. They are, however, very fast and have a much lower complexity than hierarchical algorithms. In turn, hierarchical algorithms do not require the number of clusters to execute, but are much more computationally expensive (agglomerative algorithms have a complexity in the general case of $\mathcal{O}(n^2\ log(n))$, while divisive algorithms have complexity $\mathcal{O}(2^n)$ in the worst case).

ROCK is an *agglomerative hierarchical* clustering algorithm that relies on splitting the computation in *clustering* and *labeling* to handle large numbers of records. More precisely, the clustering phase is a traditional agglomerative algorithm whose novelty lies in the method used to choose the best clusters to merge at each iteration; only a sample of the complete data set can be used when performing clustering, so that the overall complexity of the algorithm is reduced without a major loss in the quality of final clusters. Figure 2.1 shows the operations performed by ROCK, together with the parameters that can be tweaked at each step.

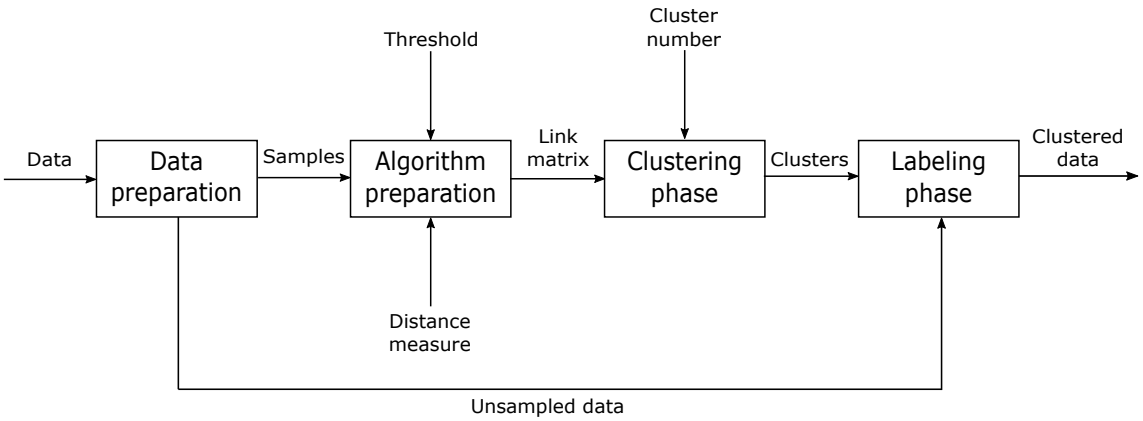


Figure 2.1: The general workflow of ROCK is presented here.

2.2 Clustering paradigm

As introduced before, ROCK is agglomerative and hierarchical, which means that clusters are built by merging together smaller clusters. To choose the best clusters to merge at each iteration, the paper [15] introduces two new concepts: *neighbors* and *links*.

Given a point p_i, a point p_j is said to be a *neighbor* of p_i if the following holds:

$$sim(p_i, p_j) \geq \theta \tag{2.1}$$

where $sim(p_i, p_j)$ is any similarity function that assumes values between 0 and 1, with equal points having similarity 1.

A link l_{ij} is a path of length 2 between points p_i and p_j such that every pair of consecutive points on the path are neighbors. $link(p_i, p_j)$ describes the number of common neighbors between p_i and p_j.

Intuitively, if $link(p_i, p_j)$ is large, the points are well connected to each other and should belong to the same cluster. This is advantageous compared to traditional approaches, that rely on the pairwise similarity between points to decide which clusters should be merged: it may happen that clusters contain outliers that are pairwise very close to each other, while not representing the actual general behavior of their respective cluster. Using the number of links instead of the pairwise similarity allows for a higher degree of generality and gives the algorithm a more global view of relationships between points, thanks to the knowledge of paths of length 2. In [15], length 2 was chosen for three reasons: firstly, computing paths of length 2 is much more efficient than computing paths of higher lengths; secondly, this length produces a tighter representation of how points are distributed compared to higher degrees and, lastly, using higher degrees does not noticeably increase the amount of information obtained.

2.2.1 Similarity function

ROCK uses a "similarity" function rather than a distance one, so that two items that are equal to each other have similarity 1 and distance 0; since both are normalized, $d = 1 - s$ where d is the distance and s is the similarity. It's important to note that the function $sim(p_i, p_j)$ introduced in Eqn. (2.1) can be any kind of distance function, even a non-metric one: this allows ROCK to be used when it is not possible to define a purely numeric distance function such as the Euclidean distance or the Manhattan distance. In order to handle mixed datasets (i.e. datasets whose records both numerical and categorical features), I chose *Gower's similarity coefficient* [1] (shown below, in Eqn. (2.2)) as a distance function for general datasets where it was not possible to use numeric distance functions.

$$d(i,j) = \frac{\sum_{k=1}^{N_{attr}} \delta_{ij}^{(k)} \cdot d_{ij}^{(k)}}{\sum_{k=1}^{N_{attr}} \delta_{ij}^{(k)}} \tag{2.2}$$

The similarity measure of two attributes is computed in different ways depending on whether they are numeric or nominal, as shown here:

$$d_{ij}^{(k)} = \begin{cases} 1 - \frac{\left|x_i^{(k)} - x_j^{(k)}\right|}{\max_h x_h^{(k)} - \min_h x_h^{(k)}} & \textit{in numeric case} \\ \left.\begin{array}{ll} 0 & if\ x_i^{(k)} = x_j^{(k)} \\ 1 & otherwise \end{array}\right\} & \textit{in nominal case} \end{cases} \tag{2.3}$$

δ_{ij} is used to decide whether an attribute is valid or not, so that missing points are ignored:

$$\delta_{ij}^{(k)} = \begin{cases} 0 & if\ x_i^{(k)}\ or\ x_j^{(k)} is\ missing \\ w^{(k)} & otherwise \end{cases} \tag{2.4}$$

$w^{(k)}$ is an arbitrary weight given to attribute k, such that $\sum_{k=1}^{n_{attr}} w^{(k)} = n_{attr}$ (in the simplest case, $w^{(k)} = 1 \forall k$). This allows the user to tweak how each attribute will influence the clustering result: for example, given a set of personal data, it may be important to highlight the marital status over the age. When all attributes are treated like nominal values and weights are the same for all features, Gower's similarity coefficient is equivalent to Jaccard's similarity.

Observation on the effect of missing values Eqn. (2.2) works well in a general case, but has some issues when multiple attributes are missing: since features are skipped altogether when they contain missing values, the similarity between two items may be inflated in case the surviving attributes are equal. Examples 2.2.1 and 2.2.2 explain this issue more thoroughly:

Example 2.2.1. Consider Table 2.1, where each A_i is a feature and "?" indicates a missing value. Applying Eqn. (2.2) to records 1 and 2 will result in similarity 1, since all attributes but one are missing, and the surviving one is the same for both tuples (the same will happen for records 2 and 3). Compare now this to what happens if we apply

Eqn. (2.2) to records 1 and 3: the corresponding similarity will be 4/5 instead, lower than the previous case. The similarity function will, therefore, consider the first two lines closer to each other than the first and the last, when in reality it should not be possible to tell given the large number of missing features.

Record	A_1	A_2	A_3	A_4	A_5
r_1	a	b	c	d	e
r_2	?	?	c	?	?
r_3	b	b	c	d	e

Table 2.1: Example of problematic case with distance function.

Indeed, the solution to this issue depends on the use case: the analyst will have to take into account this problem during the data preparation phase and choose how to approach it. In my opinion, however, Eqn. (2.2) is still valid and is still behaving as it is supposed to be. Moreover, trying to ignore (2.4) altogether has the opposite result:

Example 2.2.2. Consider the same situation as in Example 2.2.1. Here, using Eqn. (2.2) will yield similarity 1 for records 1 and 2. Dividing by the total number of records instead of the number of valid records would result in similarity 4/5 instead of 1.

Record	A_1	A_2	A_3	A_4	A_5
r_1	?	b	c	d	e
r_2	?	b	c	d	e

Table 2.2: Further example of problematic case.

Concluding, this is a drawback of Gower's similarity that should be kept into consideration when the dataset under study contains a large number of missing values. A possible solution may be removing problematic records (maybe records that contain more than a certain fraction of missing attributes), or by adding "dummy values" to missing values, so that they will still be counted as "different" for the purposes of distance measurement. An easy implementation of the latter approach for categorical attributes is setting missing values to an alphanumeric ID equal to the ID of the record: in this way, the value will for sure be different from any other feature. Missing numeric values may be filled with the average, the median, or more advanced techniques that can avoid introducing bias.

2.2.2 Criterion function

Characterizing "good clusters" is of paramount importance because an accurate model allows developing algorithms that can, given a suitable criterion function, maximize said function to produce the best possible outcome. In this case, the objective is generating clusters that have a high degree of connectivity (i.e. members of each cluster should have

many links among each other), so the idea would be maximizing the sum of $link(p_q, p_r)$ for all points p_q, p_r belonging to a single cluster, while minimizing the sum when p_q and p_r belong to different clusters. The resulting criterion function is the following:

$$E_l = \sum_{i=1}^{k} n_i \cdot \sum_{p_q, p_r \in C_i} \frac{link(p_q, p_r)}{n_i^{1+2f(\theta)}} \tag{2.5}$$

where C_i denotes cluster i of size n_i. Although it may seem that maximizing the number of links by using a function like $E_l = \sum_{i=1}^{k} n_i \cdot \sum_{p_q, p_r \in C_i} link(p_q, p_r)$ would be enough to find the best clusters, this is a naïve approach that leads to favoring large clusters over small ones. Example 2.2.3 shows such a situation.

Example 2.2.3. Assume an iteration where it is necessary to find the best candidates among three clusters: C_1 contains 400 points, C_2 contains 30 points and C_3 contains 20 points. Each cell (i, j) in Table 2.3 contains the number of links between cluster C_i and cluster C_j.

	C_1	C_2	C_3
C_1	380	20	10
C_2	20	28	15
C_3	10	15	18

Table 2.3: Example: table of links between clusters

In this case, the naïve approach would choose clusters C_1 and C_2, even though the number of links present between the two clusters is, proportionally, much smaller than the number of links between C_2 and C_3 (respectively, 20/410 and 15/50).

This is the reason why the denominator in Eqn. (2.5) is introduced: $n_i^{1+2f(\theta)}$ is the estimated number of links present in cluster C_i with size n_i. $f(\theta)$ is a function dependent on the data set and on the kind of clusters desired by the user, and such that each point belonging to C_i has approximately $n_i^{f(\theta)}$ neighbors in C_i. Assuming that points outside C_i have very few links in C_i (an assumption that improves as the algorithm progresses), each point in cluster C_i generates $n_i^{f(\theta)} \cdot n_i^{f(\theta)} = n_i^{2f(\theta)}$ links, one for each pair of its neighbors. Given that cluster C_i contains n_i such points, the overall *expected number of links* will be $n_i \cdot n_i^{2f(\theta)} = n_i^{1+2f(\theta)}$, i.e. the denominator of Eqn. (2.5).

Estimating $f(\theta)$ is not easy, and in [15] the choice fell on Eqn. 2.6: this decision was made because the paper focused mostly on basket case data, and this function reflected well that particular application.

$$f(\theta) = \frac{1-\theta}{1+\theta} \tag{2.6}$$

In [15], it is affirmed that that finding a good $f(\theta)$ is not easy; however, in case clusters are well-defined, an inaccurate but reasonable estimate for $f(\theta)$ may work well in practice. Moreover, the fact that every cluster is normalized by $n^{1+2f(\theta)}$ should mean that an error in the estimation of $f(\theta)$ should influence all clusters, thus reducing the bias.

In my tests, I observed that when records are not well distinguishable this is not true and that performances drop. In Chapter 3 I'll introduce methods for defining a valid $f(\theta)$, while in Chapter 5 I'll show how poor choices for $f(\theta)$ may lead to underwhelming clustering results when data is not well defined.

2.2.3 Goodness function

To characterize the best clusters a *criterion function* was introduced in [15]: the objective of this function is maximizing the number of links between points belonging to the same cluster while keeping the number of cross-cluster links as low as possible. As already said, choosing a function that simply maximizes the number of links is a naïve approach: there would be a very strong bias towards large clusters, which become too attractive and start "absorbing" smaller clusters. Thus, it is necessary to normalize the total number of links between two clusters by their size. This is achieved through the *goodness measure*, a score given to every pair of clusters and that represents how close they are to each other: the pair with the largest goodness during each iteration will be merged. In [15], the goodness measure is defined as follows:

$$g(C_i, C_j) = \frac{links[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - (n_i)^{1+2f(\theta)} - (n_j)^{1+2f(\theta)}} \tag{2.7}$$

The threshold θ is, together with the desired number of clusters k, one of the two parameters that determine how the clustering algorithm will behave. The choice of θ should be made by the user: choosing a value very close to 0 will often lead to a very large cluster that contains the majority of records, with some outliers left aside depending on the data set. Choosing a value of θ that is too close to 1, instead, will lead to an early stop in the clustering operation, which often will terminate before reaching the desired number of clusters: this happens because at a certain point there will be no more cross-links (i.e. links that connect two clusters), thus records will be contained by connected components only.

This is the principle upon which QROCK [10] works: this algorithm skips completely the computation of the goodness function, relying on the threshold only to perform clustering. This results in a much shorter execution time, but, at the same time, the accuracy of the final result degrades noticeably for some data sets. This issue is caused by the fact that looking for perfectly disjointed connected components is not always the right approach: indeed, very often clustering produces better results when cross-links are still present, since perfectly disjoint components are reached when the only remaining clusters are outliers. In short, looser clustering may lead in some cases to better results than what can be achieved considering connected components only.

In the following sections, I'll show the different phases of ROCK and how it adapts hierarchical clustering to employ Eqn. (2.5) as a criterion for choosing the best clusters to merge at each iteration.

2.3 Data preparation

The data preparation phase is not strictly a part of ROCK: indeed, it simply consists in parsing files in such a way that the ROCK algorithm can handle them. Specifically, each feature is classified as either *categorical* or *numerical* and then sampling is performed: a subset of the entire data set is used during the clustering phase, and the remaining portion will be handled by the labeling step. The number of samples to be chosen has a large influence on the quality of the final result, but it is a bottleneck due to memory constraints: if n is the number of chosen samples, to perform clustering it is necessary to build a matrix of size n^2, which is extremely costly. This subject will be analyzed more thoroughly in section 2.4.1.

Once samples have been extracted, the *link matrix* M_L used during the clustering phase is built: firstly, the *similarity matrix* M_S that contains the pairwise similarity of each couple of points is created using Eqn. (2.2); the *neighbor matrix* M_N is then built by setting all the entries of M_S so that $(i, j) = 1 \; if \; sim(i, j) \geq \theta$ and 0 otherwise. Finally, M_L is obtained by squaring M_N: this will produce a matrix that contains the number of undirected paths of length 2 that have points i and j as their ends, and such that all the points belonging to a path are neighbors. The need for building matrices that contain all possible pairs means that the complexity in memory is $\mathcal{O}(n^2)$, where n is the size of the sample set; this explains why it is not possible in general to perform clustering on the entire data set, and why labeling is needed.

Example 2.3.1. To better explain this phase, consider a very simple dataset S that contains the points A, B, C, D. The pairwise similarity between each of these points is shown in Figure 2.2a and in Table 2.4. The value of the threshold is $\theta = 0.5$. During the data preparation step, the algorithm will first build M_S by applying the distance function defined in Section 2.2.1, to obtain Table 2.4. Once M_S has been built, entries with similarity lower than θ are suppressed, while the remaining ones are set to 1 to obtain M_N shown in Table 2.5 (also see Figure 2.2b). Finally, M_N is squared to generate M_L, shown in Table 2.6. This last matrix will then be fed to the clustering algorithm, which will then use it to compute the goodness function.

	A	B	C	D
A	1	0.6	0.1	0
B	0.6	1	0.7	0.2
C	0.1	0.7	1	0.8
D	0	0.2	0.8	1

Table 2.4: $M_S(S)$.

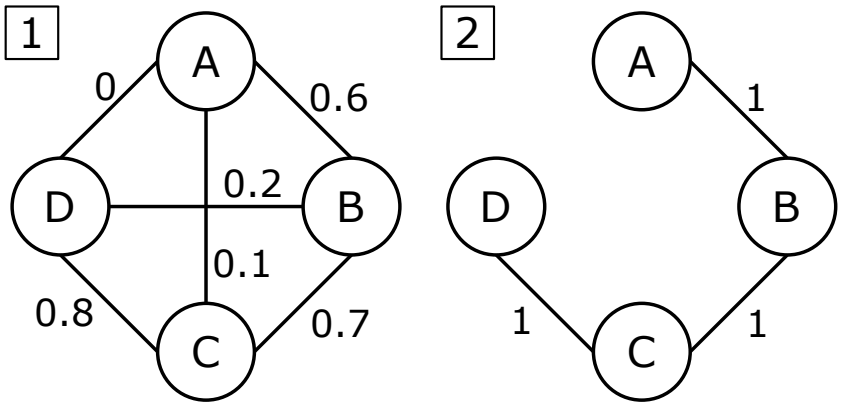
	A	B	C	D
A	1	1	0	0
B	1	1	1	0
C	0	1	1	1
D	0	0	1	1

Table 2.5: $M_N(S)$.

	A	B	C	D
A	2	2	1	0
B	2	3	2	1
C	1	2	3	2
D	0	0	2	2

Table 2.6: $M_L(S)$.

Due to the limitations of my machine, it was not possible to employ more than 5000 samples for my tests: depending on the size of the data set, I would very often get memory errors. In most cases, the time required to build the link matrix was still much shorter than both clustering and labeling (if necessary): this means that – given a distributed matrix – it would be possible to scale better and achieve a better accuracy in clustering.



(a) Starting condition.

(b) Arcs with similarity $< \theta$ are removed, surviving arcs are set to 1.

The size of the sample set, in fact, influences the final clustering result heavily, mostly by reducing the number of outliers produced by the algorithm.

In IROCK [33], the neighbor matrix is not built at all: similarities smaller than θ are still suppressed, but the link matrix is built by squaring the similarity matrix itself; this means that the resulting matrix will not contain integers anymore: this is done to include the similarity measure proposed by the paper. I didn't employ that similarity because it was specific to text analysis on documents; when using the standard ROCK procedure, however, results present negligible differences. I decided to keep the original approach to better compare my results with the original implementation.

2.4 The algorithm

After the first processing step, an additional data structure must be built before the execution starts: since this is an aggregative hierarchical algorithm, clusters initially contain only a single element and during each iteration two of them are merged together. To this end, the goodness (2.7) of each pair of clusters is computed and a heap that contains the pair $(goodness, other\_cluster)$ together with all other pairs is created; this list is accompanied by another list that contains the points that belong to the cluster. After this last preliminary step is executed, the actual algorithm described in Section 2.4.1 begins. Figure 2.3 shows all execution steps.

2.4.1 Clustering phase

The clustering phase is the core of ROCK and the only mandatory part, given an implementation that can bypass the bottleneck caused by the memory occupancy. The operations performed by the algorithm are shown in Figure 2.3 and in Algorithm 2.4.1:

Let S be a sample of the original data set

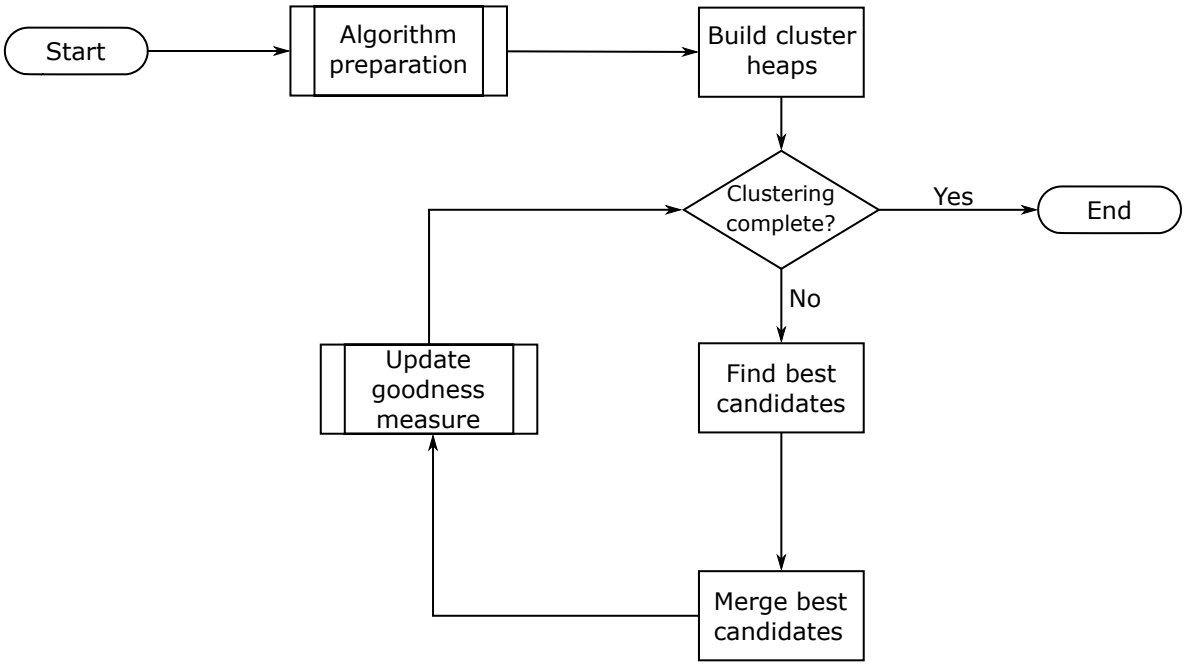


Figure 2.3: A flowchart of ROCK is presented here, showing the data preparation phase and the construction of data structures.

Let k be the minimum number of clusters
$link \coloneqq$ compute_links(S)
for all $s \in S$ **do**
  $q[s] \coloneqq$ build_local_heap($link$, s)
end for
$Q \coloneqq$ build_global_heap(S, q)
while size(Q) $\geq k$ **do**
  $u \coloneqq$ extract_max(Q)
  $v \coloneqq \max(q[u])$
  delete(Q, v)
  $w \coloneqq$ merge(u, v)
  for all $x \in q[u] \cup q[v]$ **do**
    $link[x, w] \coloneqq link[x, u] + link[x, v]$
    delete($q[x]$, u); delete($q[x]$, v)
    insert($q[x]$, w, $g(x, w)$); insert($q[w]$, x, $g(x, w)$)
    update(Q, x, $q[x]$)
  end for
  insert(Q, w, $q[w]$)
  deallocate($q[u]$); deallocate($q[v]$)
end while

`compute_links`(s) produces the link matrix defined in Section 2.3. `build_local_heap`

and `build_global_heap` represent the preliminary step described at the start of this section. Heaps are used for improving performance: there is a local heap for each cluster and a global heap that contains the top of each local heap. Each local heap contains all the clusters whose goodness measure computed with respect to the "heap owner" is larger than 0.

The *merge* step is what characterizes ROCK as a hierarchical algorithm: indeed, each iteration can be seen as a level in the dendrogram and, in principle, the algorithm may end at any step (this is the reason for using k). During each iteration, the best clusters to merge are found looking at the top of the heap; a new cluster that combines the elements of the two best clusters is formed and the goodness measure for every involved cluster is updated. Finally, heaps are updated with the inclusion of the newest cluster. The other possible termination condition is reached when mergers would not increase the number of links inside a cluster, i.e. all clusters are already well distinct and correspond to connected components [10]: no goodness measures are larger than 0. Very importantly, this does not mean that the optimal result has been reached: in fact, this point may be reached after a single cluster absorbed most other clusters. This situation usually occurs when the threshold is close to 1 and, while choosing a suitable number of clusters may help prevent it, it may still occur when it's not as high.

The `update(Q, x, q[x])` step consists in computing the value of the goodness measure of clusters w and x: this is done for each cluster involved in the merge, i.e. for each cluster present in the similarity matrix of either cluster u or cluster v. This is done by computing the sum

$$links(C_i, C_j) = \sum_{i \in C_i} \sum_{j \in C_2} link(i, j) \tag{2.8}$$

After the clustering phase terminates labeling may be executed in case a sample was taken. If the entire dataset was clustered, no further action is performed.

Example 2.4.1. To better explain the goodness update phase performed by the step `update(Q, x, q[x])`, consider the Link matrix obtained in 2.3.1 and reported here in Table 2.7 for simplicity. Suppose the iteration step requires to compute the goodness of two clusters C_1 and C_2, such that $C_1 = \{A, B\}$ and $C_2 = \{C\}$. Given Eqn. (2.7), to obtain $g(C_1, C_2)$ it's necessary to first compute the number of links between C_1 and C_2: this can be done by finding the intersection of the rows corresponding to points A and B, and the column corresponding to point C. In this case, the cells are $(A, C) = 1$ and $(B, C) = 2$, so the total number of links is 3. With $f(\theta) = \frac{1-\theta}{1+\theta} = 1/3$, the final value of the goodness is obtained by inserting the values in Eqn. (2.7), whose result is $1.452468\ldots$. This is the value that will be assigned to both cluster C_1 and cluster C_2.

2.4.2 Labeling phase

The labeling phase attempts to assign each unlabeled point to a cluster or marks the point as an outlier in case this were not possible. To choose the best cluster for each point, a subset of size $L_i = min(|C_i|, L)$, where $|C_i|$ is the cardinality of C_i, is extracted from cluster C_i and the number of neighbors of the unlabeled point present in the L points is computed. Chapter 5 contains in-depth observations on the result of choosing one of

	A	B	C	D
A	2	2	1	0
B	2	3	2	1
C	1	2	1	2
D	0	0	1	2

Table 2.7: Link matrix of S with, highlighted, the rows and column affected in Example 2.4.1.

two different methods, sequential sampling or random sampling. This number N_i is then normalized by the number of points in the subset, to once again normalize the score by the expected number of neighbors:

$$r_i = \frac{N_i}{(|L_i| + 1)^{f(\theta)}} \tag{2.9}$$

where $|L_i|$ is the number of points in subset L_i of cluster C_i. $f(\theta)$ is used to estimate the number of links present in a cluster of size $|L_i|$. Once the value r_i has been computed for all clusters, the highest value is chosen and the point is assigned to the corresponding cluster. The sequence of operations performed for every unlabeled point is shown in Figure 2.4.

In [15], labeling is not described thoroughly, so I implemented three different versions: L_0, where only the points belonging to the cluster at the start of the labeling operation (L_0 for simplicity) are considered; L_1, where any points that are added to the cluster may be employed in later iterations and L_3, where outliers are considered as valid clusters and may be used during further labeling iterations.

Going back to the sampling phase, it's important to note that the size of the sample set influences the result of the labeling phase: indeed, if the sample set is very large, there will be more potential "cluster seeds" that can be used by unlabeled points to get assigned to. Ideally, the labeling phase shouldn't be necessary and clustering should be done on the entire dataset. As already said, however, this is not possible due to the large complexity in time and memory of the clustering phase. The size of L has a large effect on both execution time and accuracy, since using 40 points instead of 100 results in shorter execution time but, at the same time, more outliers. This is caused by the fact that taking more points from each cluster gives more "chances" to each unlabeled point to find at least a single neighbors in at least a cluster and when L is small, this chance is much smaller. The value of L is another parameter that should be chosen on a use case basis: it's a tradeoff between the cost of handling more points for each cluster (which results in a longer execution time) and obtaining a smaller number of outliers. Further observations on the effect of labeling will be made in Section 5.2.

Labeling is less accurate than clustering because each point is assigned considering only a subset of each cluster, and it does not use links at all: only pairwise similarity is employed in this phase. However, the clusters produced in the hierarchical phase are

usually very homogeneous. This means that taking only a sample from each cluster does not result in a big loss in accuracy, so the gain in performance largely offsets this drawback. When studying the literature, ROCK was considered to be a fully hierarchical clustering algorithm with complexity n^2 because of the matrix cost, so the labeling phase was very often forgotten. In my opinion, labeling is a very interesting feature of ROCK since it allows achieving good performances without the need for very large distance matrices.

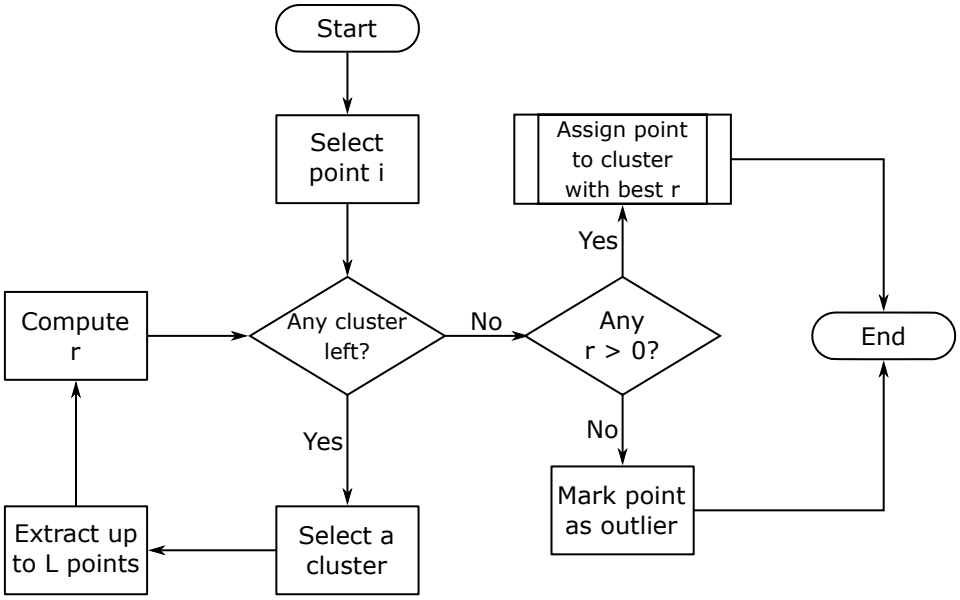


Figure 2.4: Here is shown the inner loop in the labeling phase, where it is decided whether the unlabeled point will be assigned to a cluster or not.

2.4.3 Complexity

Consider the following nomenclature:

- N: number of elements in the complete dataset
- n: number of elements in the sample dataset
- k: minimum number of clusters
- l: number of points considered from each cluster to perform labeling
- t: iteration counter
- z: size of the largest cluster

The most expensive operations performed by ROCK are generating the link matrix, updating the similarity lists and performing labeling.

Building the similarity matrix is an operation with complexity $\mathcal{O}(n^2)$, while the multiplication needed to generate the link matrix has complexity $\mathcal{O}(n^3)$ (or lower, depending on

the algorithm used to perform the operation). Updating the similarity has a complexity that varies greatly depending on the size of the largest cluster. The snippet of code below has complexity $\mathcal{O}((n-t)\cdot z + t + z)$ (where `cluster_y` is the cluster that has been built during the current iteration). In the worst case, all points will be assigned to a single cluster so that, at iteration t, its size will be $n - t = z$. The final complexity will then become $\mathcal{O}((n-t)\cdot(n-t)+(n-t)))$, which is close to quadratic in case t is small enough. Furthermore, this step must be repeated once for each iteration, therefore another factor $n-k$ needs to be added to the equation to obtain

$$\mathcal{O}(((n-t)\cdot z + t + z)\cdot(n-k)) \approx \mathcal{O}((n\cdot n + n)\cdot(n)) \approx \mathcal{O}(n^3) \tag{2.10}$$

Finally, labeling is performed on the entire dataset, so the number of iterations is fixed and equal to $N - n$, and each iteration contains the computation of neighbors present in each cluster: in the worst case, there will be n clusters (this would be a pathological case where each point is completely different from all other points). To compute the number of neighbors in a cluster, the number of iterations is l (in case there are s clusters, $l = 1 \quad \forall s_i \in s$). Here, the worst case complexity is reached when all clusters have size l, which means that on average n/l clusters must be considered in each iteration. In this situation, the complexity is $\mathcal{O}(N \cdot l)$. This gives an overall complexity of

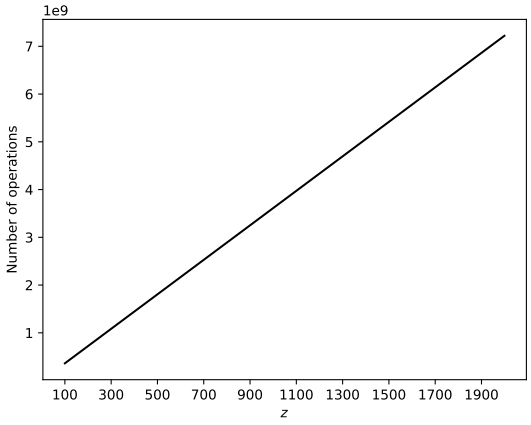


Figure 2.5: Complexity as function of z (size of the largest cluster) for 2000 samples.

$$\mathcal{O}(N \cdot l + n^3) \tag{2.11}$$

This means that, while on average labeling dominates the cost of clustering, it may happen that in the worst case and with a large sample the clustering operation may be more expensive enough to be comparable to the labeling step. The complexity as a function of the size of the largest cluster is shown in Figure 2.5.

```
1 for cluster_x in sim(cluster_y):
2   for ii in cluster_x.records:
3     for jj in cluster_y:
4       n_links = sum(link_matrix[ii][jj])
5       goodness = compute_goodness(n_links,
6                                   len(cluster_x),
7                                   len(cluster_y))
```

In the next section, I will explain the improvements I made to ROCK and how they fix some of the issues described so far.

Chapter 3

My contributions

In this section, I will describe the reasoning behind my improvements to ROCK, what they do and how they can improve the behavior of the original algorithm in various applications. Firstly, in Section 3.2.1 I present an empirical method for designing a function of θ able to handle datasets whose records are not well distinct from each other. After doing so, in Section 2.2.3 I introduce a completely new termination condition for the clustering phase, that allows the algorithm to stop once merging clusters becomes counterproductive, providing an estimate of the number of clusters that will be needed. Section 3.2.3 reports the studies I made to understand whether a parallel execution in the code could be introduced, Section 3.2.4 describes the different versions of labeling I developed and Section 3.2.5 describes the use of weights in the algorithm. Hierarchical distance is described in Section 3.3. It is a stand-alone concept: it can be used the standard ROCK implementation, or in any other clustering algorithm that can handle arbitrary distance measures, so it is not part of eROCK.

3.1 Motivation

While I was performing tests with ROCK I noticed that at times clusters were extremely asymmetric: a single cluster contained most points in the dataset, while few other outliers were left out in clusters that contained only a single point and results didn't improve by changing the threshold. This was especially noticeable with datasets such as the *adult* or the *IHIS* one, while it didn't happen when I was testing the *agaricus-lepiota* dataset. This was a very large issue when the purpose of clustering was performing anonymization: having a very large cluster to anonymize reduces the utility of the result since more records will be taken into consideration to perform generalization and suppression. To solve this issue, I focused on how the algorithm handles its parameters, trying to introduce further ways to tweak the computation of parameters. GDSTOP modifies the execution means of the algorithm so it is a somewhat more radical addition to the original program.

3.2 eROCK (enhanced ROCK)

This section describes the studies I made on ROCK, i.e. improving $f(\theta)$, introducing GDSTOP, studying parallelization, modifying labeling and weighing parameters.

3.2.1 Design of $f(\theta)$

As explained in Section 2.2.2, $f(\theta)$ has the purpose of estimating in a heuristic way the number of links contained by cluster C_i (a quantity needed to compute the value of the goodness) without the overhead required in case the number of links had to be evaluated exhaustively. In the original paper, [15], Eqn. 3.1 was chosen to perform this task, with the reasoning that it can describe well the market basket case.

$$f_0(\theta) = \frac{1-\theta}{1+\theta} \tag{3.1}$$

During my experimentation with ROCK, I noticed that, while the basic algorithm worked well on some datasets (generating homogeneous and pure clusters), it did not perform quite as well in other cases; this was especially evident with datasets whose records were not well distinct from each other (mostly datasets that contained personal data, such as the Adult data set from [21], or the IHIS dataset [29]). In these cases, a single cluster contained most items in the data set, with some outliers left out: this may be the desired behavior for anomaly detection, but it is a concern if the objective is performing anonymization. In the latter case, to achieve a high utility clusters must be homogeneous; a large cluster that contains the majority of records is too heterogeneous and the resulting equivalence classes will need more transformations to achieve k-anonymity. Moreover, whenever this imbalanced situation occurred, many points were left out without a cluster.

My intuition was that large clusters were overly favored when they were considered as merge candidates because the normalization function wasn't able to score clusters independently of their size, which is the idea behind the goodness measure. Indeed, since large clusters tend to have many links between them, their score should be normalized by the number of links expected to be found between the clusters. Therefore, if two clusters have more links between each other than all other clusters, but have fewer links than what is expected to be found in clusters of their size, they should be penalized. Often, this did not happen with Eqn. (3.1): the largest cluster remained the most attractive one even when further mergers reduced its quality. To solve this problem, the idea I had was changing the normalization performed by the denominator in such a way that it would be able to perform better in these more demanding cases. To my knowledge, this approach was not considered in other variations of ROCK: the ones I found still use (3.1).

Initially, I approached the issue of choosing a new $f(\theta)$ thinking that what I needed to obtain a larger denominator was simply a larger exponent: my first attempts suggested functions such as $f(\theta) = n \cdot \theta \quad n > 0$ as a simple (and incorrect) solution to the problem. The issue with this kind of functions is the fact that I didn't consider that two constraints must be satisfied by any valid $f(\theta)$: by definition (see Section 2.2.2), $f(\theta)$ is a function such that each point belonging to a cluster C_i has $n^{f(\theta)}$ neighbors in C_i; thus, when $\theta = 1$, the only neighbor of a point p_i is the point itself (since $n^{f(\theta)} = 1$). Conversely, when $\theta = 0$

every point in C_i will have n_i neighbors and $n^{f(\theta)} = n$. This means that $f(\theta)$ must be defined in [0,1] and should be decreasing in the range [0,1].

My first solution did not satisfy either constraint: given a n large enough, a point in a cluster of size n could have more than n neighbors in the cluster; other than that, using a function that increases in the range [0,1] leads to inconsistent results, since for large values of θ the estimated number of neighbors would be, would have, impossibly, larger than the estimate for low values of θ.

In the second iteration, I designed a series of functions of θ that satisfied the two constraints and were able to handle homogeneous data sets more effectively: all of them share the same trait of remaining close to 1 much longer than Eqn. (3.1). They also all depend on an additional parameter $t \in [0,1]$ that allows the user to tweak the slope according to the use case. Functions $f_1(\theta)$, $f_2(\theta)$, $f_3(\theta)$ will be referenced throughout my report as I performed most of my tests with all of them.

$$f_1(\theta) = \frac{\frac{1}{t} \cdot (1-\theta)}{\frac{1}{t} - \theta} \qquad t \in (0,1] \tag{3.2}$$

$$f_2(\theta) = cos(\theta \cdot \frac{\pi}{2})^{1-t} \qquad t \in [0,1] \tag{3.3}$$

$$f_3(\theta) = (log_2(2-\theta))^{1-t} \qquad t \in [0,1] \tag{3.4}$$

Figure 3.1 shows the behavior all the functions introduced above, given $t = 0.85$, while Figure 3.2 shows the behavior of the same functions with $t = 0.5$. Table 3.1 presents an excerpt of the values of $f(\theta)$ with $t = 0.85$. It's important to note that $f(\theta)$ is an exponent in Eqn. (2.7), so if it increases its effect on the denominator will be very large (see Example 3.2.1). This proved to be very effective at preventing large clusters from absorbing small ones and improved the results when clustering data sets such as the UCI *adult* one. Yet another example of the huge difference caused by changing $f(\theta)$ is shown in Figure 3.3, where the behavior of the goodness function itself is shown. In the figure, on the x and y axes lie n_1 and n_2 (both in the range [10,20]), while on the z-axis the value of the goodness function when the algorithm is using $f_0(\theta)$ is shown. Interestingly, the value of the goodness when using $f_0(\theta)$ is so much larger than the other cases that the function looks like a 2D "sheet". Figure 3.4 shows that this is due to the scaling of the plot.

Example 3.2.1. When computing the value of the goodness measure, a cluster C_i will have size n_i. n_i will be raised to power $g = 1+2f(\theta)$ as per Eqn. (2.7). Given $\theta = 0.70$, the value of the exponent g will be $1 + 2f_0(0.70) = 1.1622$ for $f_0(\theta)$ and $1 + 2f_1(0.70) = 2.573$ for $f_1(\theta)$. This is a huge change: the exponent more than doubles so, instead of a slight increase, the denominator is more than squared as a consequence.

It is important to note that the choice of $f(\theta)$ should be made while keeping the intended use case in mind: for example, when performing fraud and anomaly detection, obtaining balanced clusters has a much lower importance than being able to identify and isolate outliers; when studying how records are grouped or when performing anonymization, however, obtaining homogeneous clusters is, instead, more significant. From practical experience, however, using stronger functions produced better results in general, and has

the additional advantage of reducing the execution time and making the worst case less likely: since large clusters are heavily penalized, the value of z remains "manageable", thus reducing the complexity.

θ	$f_1(\theta)$	$f_2(\theta)$	$f_3(\theta)$	$f_4(\theta)$
0.70	0.0811	0.5405	0.7865	0.8039
0.85	0.1765	0.7407	0.8644	0.8883

Table 3.1: Value of the exponent with different $f(\theta)$ and values of θ.

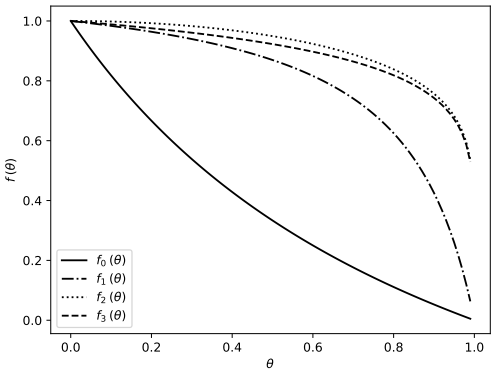


Figure 3.1: Some different behaviors of $f(\theta)$ are shown here, given $t = 0.85$.

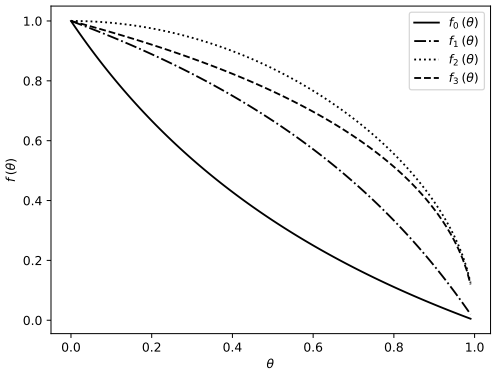


Figure 3.2: A different example of the different behaviors of $f(\theta)$, here with $t = 0.5$.

3.2.2 GDSTOP

In the original paper [15], there were two possible conditions for the algorithm to stop iterating: either the number of clusters specified by the user was reached (ROCK is a hierarchical clustering algorithm), or no cross-links could be found between any two clusters (so there was not a single connected component that spanned the entire similarity graph). There are two major problems with this approach:

- It is very hard to decide the number of clusters before starting the clustering operation. This is a common issue among clustering algorithms.
- When clustering some data sets, cross-links may remain present for the entire duration of the clustering phase, thus leading to cluster mergers that worsen the clustering accuracy.

The first point can be described by the question "at what height in the dendrogram should I stop to obtain the best clusters?" for hierarchical algorithms, or "how many centroids should I generate before starting the execution?" for partitional algorithms: to answer to

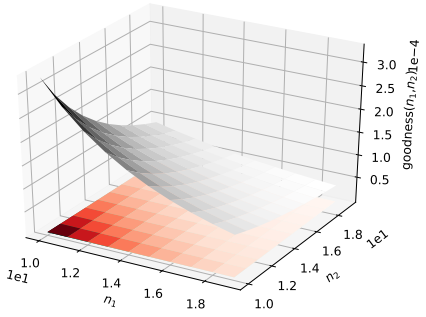


Figure 3.3: Behavior of the goodness function with different $f(\theta)$. Here, $f_0(\theta)$ is in grey, $f_1(\theta)$ is in red.

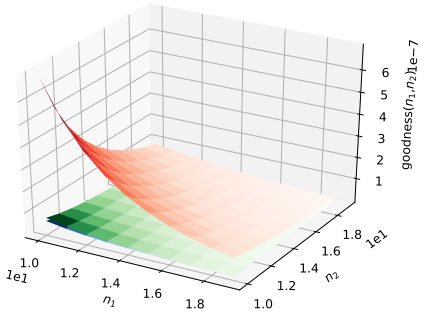


Figure 3.4: Close in of the goodness function with $f_1(\theta)$, $f_2(\theta)$, $f_3(\theta)$ respectively in red, green and blue.

either question, the elbow method or some equivalents is often used. This means observing the variance presented by the clusters as a function of the number of clusters: the point at which the increase is variance is not as large as in the previous will be the "elbow" and will define the number of clusters that should be used.

In the following section, I introduce a new termination condition to add to the two above, called *GDSTOP*. Using GDSTOP, the clustering algorithm terminates once the maximum goodness value is lower than a certain threshold: this means that the algorithm has a mean for understanding whether further merges would worsen the result or not. This is especially helpful in a preliminary study of the data set: since this operation is confined to the clustering phase, it is possible to repeat this operation multiple times and analyze the clustering result, without performing labeling. The results obtained in this way may give a good insight into how many clusters should be chosen when the labeling phase is executed. It can be seen as a "soft" version of the termination reached when all clusters are distinct: in this case, although links between clusters may still be present, their number is low and mergers may be counterproductive.

Choosing the termination value

The choice of the termination value influences heavily the moment when the algorithm stops clustering, and consequently the quality of the final clusters. Initially, my approach was to terminate the execution once the goodness was lower than the threshold:

$$f_g(\theta) = \theta \tag{3.5}$$

However, I noticed that this approach failed when the threshold was close to 1 since, depending on what $f(\theta)$ is, the optimal result may require further merging operations. Even with lower values of θ, however, clustering would often stop too early to reach a "good" number of clusters, so I looked for a better definition. Once again, this happened with

problematic clusters. In a similar fashion to what I did with $f(\theta)$, I identified some constraints that $f_g(\theta)$ must satisfy: it must depend on θ, it must be defined in the range [0,1] and it should decrease monotonically in the range [0,1]. Additionally, it should result in a lower bound for g_{max} than Eqn. (3.5). I needed the first constraint to avoid increasing the number of parameters required by the algorithm. Having a decreasing function is needed to balance the larger number of links that will be present when the value of θ is low: if the threshold decreases, the number of links will increase and to counteract this behavior the execution should stop earlier.

Functions similar in shape to $1 - \theta$ satisfied the requirements, but the lower bound they produced was too large. Eventually, I decided to use the following equation to produce the termination value:

$$f_g = \frac{1-\theta}{\frac{1}{1-\theta} + f(\theta)} \tag{3.6}$$

where $f(\theta)$ is the same equation used to compute the goodness (e.g. those defined in section 3.2.1). This function was able to produce good results for most data sets and, unlike what happened with previous functions, without the issue of stopping too early; in some cases, execution stopped when it reached the optimal number of clusters (*agaricus-lepiota*, *dummy*). Figure 3.5 shows the behavior of Eqn. (3.6).

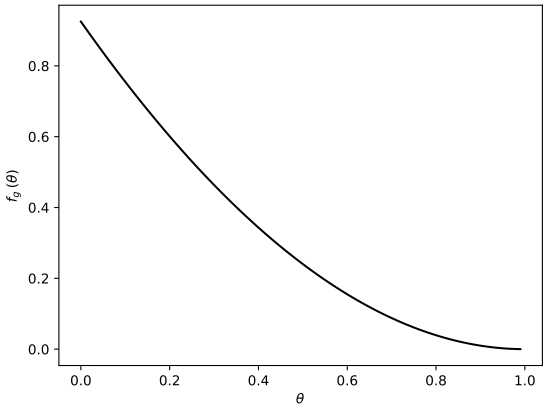


Figure 3.5: The behavior of $f_g(\theta)$ is shown in the picture above.

When performing tests on this approach, however, I noticed that GDSTOP does not always yield optimal results. It may happen that the clustering phase will end before reaching the optimal number of clusters: this means that Eqn. (3.6) is not appropriate for any dataset. When this happens, a good approach is observing the number of clusters produced by GDSTOP and then force the algorithm to use the standard termination with k equal to this value, or lower than it. The advantage of GDSTOP is having a good idea of the number of clusters that should be used by the clustering algorithm on a certain dataset: indeed, in some cases datasets with similar sizes required wide range of different values of k (e.g. around 100 for the adult dataset, 30 for the IHIS dataset, 250 for the clickfraud dataset, 9 for the dummy dataset). Regardless of the quality of the result, GDSTOP is a simple indication of the number of clusters that can be used, but it does not replace the termination with k: after all, the exact same result may be achieved by setting k to be the same number of clusters produced using GDSTOP. Still, it remains a useful tool for estimating the cluster number before more in-depth analysis and a finer tuning of the number of clusters.

In my opinion, the best procedure consists in running the clustering algorithm once

with $k = 1$ to understand the minimum number of clusters that may be formed given the parameters chosen in the run and then running again the algorithm using GDSTOP: in this way it is possible to obtain a *range* of possible values of k, with the number of clusters produced using GDSTOP being the upper bound and the number of clusters present when $k = 1$ as the lower bound. In some cases, the result may not be optimal, however the need for only two runs to obtain a good estimate of the number of clusters is very encouraging given how many more runs are often needed by techniques such as the elbow method.

Observations on the termination condition

As explained in Section 2.4.1, there are two possible termination conditions in the original algorithm. GDSTOP acts as a "fuzzy" version of the termination caused by the threshold: depending on the value of the maximum goodness during the last iteration it is possible to understand what caused the algorithm to stop. In fact, if GDSTOP is not in use and the goodness is 0 then clusters are completely distinct, with no links connecting any two clusters; when the number of clusters reaches k, the value of the goodness is usually larger than 1 and further merging operations could be performed (this does not mean that they would necessarily improve the result). When GDSTOP is used and the number of clusters is small enough, the goodness will be very small but still larger than 0. The value of the goodness depends heavily on what θ was.

3.2.3 Introducing parallelization

While trying to improve the performances of the algorithm, I studied how parallel execution could be introduced. Unfortunately, ROCK is intrinsically sequential: each iteration depends on the result produced by the previous step both during clustering the clustering phase and during the labeling phase. I identified four phases that may be executed in parallel: performing matrix multiplication, searching the best cluster, updating the goodness measure and labeling.

For what concerns the first, the computation of the link matrix can easily be parallelized by distributing operations on different nodes.

For what concerns clustering, it is still possible to introduce some degree of parallelization within each iteration, although to a limited degree. The research of the maximum goodness can be parallelized by splitting the collection of heaps and distributing it on different nodes; similarly, once the merger has been completed, the goodness update for all involved clusters can be performed in parallel since operations on each pair of clusters are independent of each other. The code that performs the search is shown below: the first snippet shows the standard operation, where each cluster in the heap is studied by the processor to find what the maximum goodness is over the entire set of clusters, while the second presents the parallel version.

```
1 def find_best_clusters(cluster_heap):
2   # Initialize variables
3   best_key = -1
4   best_similar_key = -1
```

```
5    best_goodness = 0.0
6    for index in cluster_heap:
7      cluster = cluster_heap[index]
8      # Find the top of the heap
9          max_gd = cluster.sim_map.goodness
10     best_candidate = cluster.sim_map.index
11     # Update best goodness if needed
12         if best_goodness < max_gd:
13       best_goodness = max_gd
14       best_key = index
15       best_similar_key = best_candidate
16   # Return IDs of the best clusters
17   return (best_key, best_similar_key)
```

In the parallel case, the cluster heap is distributed over a pool of nodes, each of which executes the code described in the fully sequential case on a fraction of the heap. Here, the pool function returns a list that contains the best cluster according to a node and the corresponding goodness. After the computation has been completed, the actual best goodness is found by calling the max function and extracting the corresponding keys.

```
1  def find_best(cluster_heap):
2    # Initialize variables
3    best_key = -1
4    best_similar_key = -1
5    best_goodness = 0.0
6    for index in cluster_heap:
7      # Select current cluster
8      cluster = cluster_heap[index]
9      # Find the top of the heap
10     max_gd = cluster.sim_map.goodness
11     best_candidate = cluster.sim_map.index
12     # Update best goodness if needed
13     if best_goodness < max_gd:
14       best_goodness = max_gd
15       best_key = index
16       best_similar_key = best_candidate
17     # Return IDs of the best clusters
18   return (best_key, best_similar_key), best_goodness
19
20 def parallel_find_best_clusters(cluster_heap):
21   # Distribute clusters among nodes
22   cluster_groups = split_clusters(cluster_heap, number_nodes)
23   max_gd = 0
24   # Find best pair of clusters in a subgroup
25   (bk, bsk), gd = pool(find_best, group)
```

```
26    # Find best clusters overall
27    max_gd = max(gd)
28    best_key = bk[gd.index]
29    best_similar_key = bsk[gd.index]
30    # Return IDs of the best clusters
31    return (best_key, best_similar_key)
```

Labeling is more problematic: since the cluster set is updated at the end of each iteration by adding a point to an already present cluster, or by creating a new cluster that contains an outlier, introducing parallelization cannot be done in a simple way: in fact, choosing to process unlabeled points in batches of a generic size n would lead to an unacceptable drop in precision, since a large part of the batch would not see the "current" situation, thus leading to results that are biased towards what was the situation before the start of the execution. The following snippet shows how the labeling operation is performed during the sequential case.

```
1  def label_data(clusters, complete_data, th, l_points):
2    for point in unlabeled_points:
3      best_c = -1
4      max_ratio = 0.0
5      for index in original_clusters: # For L_2
6      # for index in clusters:  # For L_3
7        n_neighbors = 0.0
8        cluster = clusters[index].records
9            # Find L
10       chosen_points = min([len(cluster), l_points])
11       for point in cluster[:chosen_points]: # Sequential sampling
12         n_neighbors += find_neighbors(
13                          complete_data[point],
14                          unlabeled_point, th)
15         r = normalization(n_neighbors,
16                             chosen_points, th)
17         if r > max_ratio:
18           max_ratio = r
19           best_c = index
20     if best_c in clusters.keys():
21         # Add point to cluster
22       clusters[best_c].records.append(idx)
23     else:
24         # Create a new cluster that contains the point
25       next_key = max(clusters.keys()) + 1
26       clusters[next_key] = RockCluster()
27       clusters[next_key].records = [idx]
```

It is possible to take a "milder" approach by using small batches whose size is equal

to the number of nodes and then splitting this smaller batch. Unfortunately, the accuracy of the result will decrease in both cases, although an argument may be made on whether an increase in performances warrants a possible drop in quality. This idea is proposed in the following snippet of code.

```
def assign_point()
  best_c = -1
  max_ratio = 0.0
  for index in original_clusters: # For L_2
  # for index in clusters:  # For L_3
    n_neighbors = 0.0
    cluster = clusters[index].records
    # Find and select L points
    chosen_points = min([len(cluster), l_points])
    for point in cluster[:chosen_points]: # Sequential sampling
      n_neighbors += find_neighbors(
                complete_data[point],
                 unlabeled_point, th)
      r = normalization(n_neighbors,
                        chosen_points, th)
      if r > max_ratio:
        max_ratio = r
        best_c = index
    if best_c in clusters.keys():
      # Add point to cluster
      clusters[best_c].records.append(idx)
    else:
      # Create a new cluster that contains the point
      next_key = max(clusters.keys()) + 1
      clusters[next_key] = RockCluster()
      clusters[next_key].records = [idx]


def parallel_label_data(clusters, complete_data, th, l_points):
  for idx in unlabeled_points:
    # Extract a batch of points, then
        # assign each point to a different node
    batch = unlabeled_points[idx:idx+n_nodes]
    pool(assign_point, batch)
```

Due to a lack of time, I couldn't perform tests on this subject and so I don't have any conclusive result that can favor a side rather than the other. Moreover, since my implementation was realized in Python I could not use a multithreaded approach because of the Global Interpreter Lock (GIL). Attempting to use multiprocessing did not improve the situation due to the slowdown caused by the need for synchronization. Moving the

code to a different language and finding a solution for this problem will be future work.

3.2.4 Improvements to labeling

As briefly described in Section 2.4.2, labeling was described very concisely in [15], so I realized three versions of it. The first one (L_0 for simplicity) can actually be parallelized with no issue at all: any unlabeled point that has been added to a cluster will be ignored, so only the initial condition will be considered during the entire operation. Intuitively, this method is deeply flawed and the results proved it: the number of resulting outliers was very large since the algorithm cannot take advantage of new points once they have been added to a cluster. To solve this issue, I implemented two incremental variations: in the first (L_1), when a point is added to a cluster, it will be considered in the successive iterations (as long as it falls within the L chosen points), while outliers become clusters that contain only a single point; in the last version L_3, these outliers are considered as valid clusters by any yet-to-label point. Both second and third versions reduced noticeably the number of outliers resulting from the labeling phase compared to the previous version.

From the performance point of view, the first two approaches differ slightly (the execution of L_1 will take longer because clusters may increase in size and thus more points may be considered as the labeling operation progresses), while the third may heavily encumber the execution: since every outlier becomes a valid cluster, in a worst case scenario every single point may become an outlier and an unlabeled point n will have to browse through the previous $n - 1$ points. This leads to a complexity which is quadratic with respect to the size of the complete dataset: this is problematic with very large sets that produce many outliers, so this approach should not be used in every case. In those situations where the number of outliers remained manageable, L_2 resulted in a lower number of outliers; observing the number of outliers produced by L_1 is a good indication of whether L_2 can be applied to the use case or not.

A somewhat surprising result is the fact that, although labeling is intrinsically less accurate than clustering (since it considers only a fraction of each cluster and considers pairwise distances between points only, instead of employing links), the quality of clusters didn't decline noticeably when I started considering outliers as valid clusters.

3.2.5 Weighing features

A very simple addition that was described in Section 2.2 consists in introducing weights to features: this allows to favor certain attributes over others so that two records that are similar (or dissimilar) in that particular attribute will "feel" it more than they would in a different case. Doing so has the result of obtaining clusters that combine records with the same values for certain features. Therefore, it may be possible to produce clusters that are particularly homogeneous: for example, when clustering census data it may be important to preserve the marital status over the ZIP code; this can be done by increasing the weight of the marital status so that records that differ in that are seen as "more distant" from each other than what would happen with no weights. Being able to tweak the importance of a given feature, moreover, allows achieving better results when clustering a dataset with an objective in mind.

3.3 Hierarchical distance

The final concept I considered during my studies is a way for preserving some of the information contained by the attributes that can be generalized hierarchically or that show some semantic similarity (e.g. professions, or education level): an example of such attribute is shown in Figure 3.6. Taking the hierarchy into consideration allows doing so, unlike what is achieved through a simple string comparison. This field has already been considered in the literature [32] [25]; I introduce a slight variation that lets the user the distance depending on the set, and more interestingly I use it to perform anonymization. For this reason, the discussion relative to its effectiveness is in Section 4.2.

Still, while it's definitely an interesting approach for some applications, this cannot be applied to just any categorical attribute, or attribute in general. Some attributes, although categorical, cannot be placed in a hierarchy in a trivial way: for example, if an attribute contains a city name, the criterion with which it should be placed on a hierarchy is not obvious. Similarly, ID's, proper nouns and similar features cannot be generalized at all.

For those cases where this method is viable, it is necessary to devise the correct hierarchical tree for each of the available attributes: this is not a trivial task, as I noticed when performing anonymization. Having a deeper tree allows for a finer-grained scoring of similarities so that it is possible to better classify different records; in the anonymization case, this means that the size of the transformations that must be performed is smaller than in a coarser hierarchy.

The similarity measure I employed and slightly tweaked is the one introduced in [32]. It uses a *log* function to produce a score that decreases sharply as two nodes move on further branches, and is based on the height of the *Lowest Common Ancestor* (LCA) between any two values, where the LCA of nodes p_i and p_j is the lowest (i.e. deepest) node that has both p_i and p_j among its descendants. The function is the following:

$$sim(p_i, p_j) \;=\; 1 - log_2\left(1 + \frac{h-d}{h}\right) \tag{3.7}$$

Here, I am measuring the similarity between points p_i and p_j, hence the score increases as the value of the logarithm decreases. h is the height of the tree (i.e. the distance between root and leaves), while d is the height of the LCA. This function behaves in such a way that, if the attribute has the same value for two records the similarity is 1; if the attribute is different, the similarity decreases accordingly.

A possible way for tweaking the similarity is adding an exponent to the logarithm as follows:

$$sim(p_i, p_j) \;=\; 1 - log_2\left(1 + \frac{h-d}{h}\right)^{1-t} \tag{3.8}$$

Adding the $1-t$ parameter allows changing the similarity value given to points p_i and p_j depending on their distance. Figure 3.7 shows some examples of the behavior of Eqn. (3.7) as t increases. Having a lower similarity value may be important for some applications, although I couldn't find a criterion for choosing an optimal value of t. This is a part of future work.

Example 3.3.1. To measure the similarity between two records that have the feature *marital status*, it's necessary to find the LCA for that attribute in relation to the two records. Then, Eqn. (3.7) can be applied to compute the similarity between them. The *MARSTAT* LCA of records 1 and 2 is *Spouse present*, and its depth is 1. Therefore, $sim(r_1, r_2) = 1 - log_2\left(1 + \frac{2-1}{2}\right) = 0.4150$. If, instead, we consider records 1 and 3, the LCA is the root (for which $d = 0$), so the similarity becomes $sim(r_1, r_3) = 1 - log_2\left(1 + \frac{2-0}{2}\right) = 0$ as expected.

3.4 Notes on the implementation

All the functions and algorithms included so far (and in Chapter 4) were implemented using Python 3 and the libraries included in it. Some additional open source libraries were used for more advanced tasks. Numpy [39] was used for matrix multiplications and most of the "heavier" operations, and is needed by many other libraries; I used Matplotlib [16] for building all plots in this report; the AMI, V-Measure, Precision, AUPRC metrics (when used) were taken from Scikit-learn [28]; the Pandas [24] library was employed to perform data analysis and to modify larger datasets; Cython [6] was used to recompile eROCK to achieve better performances by using a C implementation.

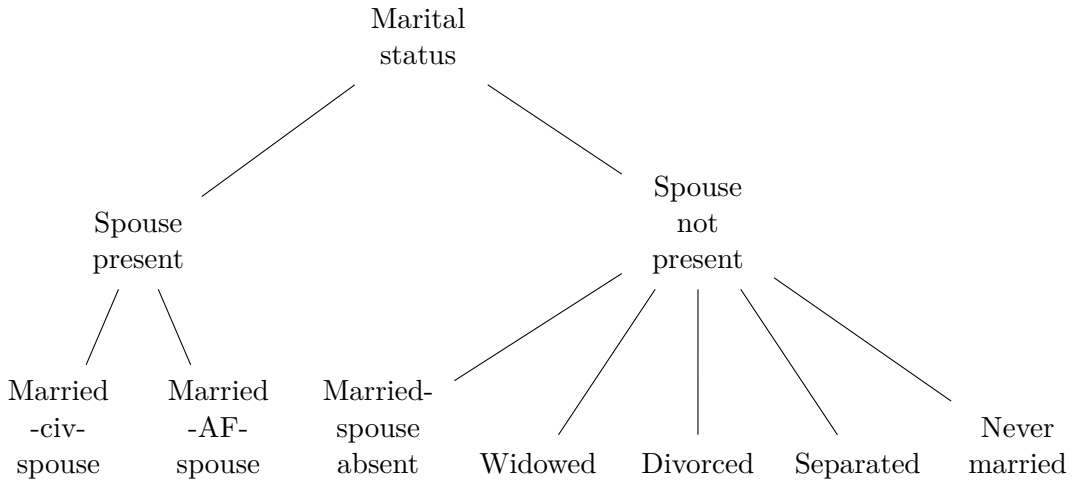


Figure 3.6: Example of hierarchical attribute representing the marital status hierarchy in the adult dataset.

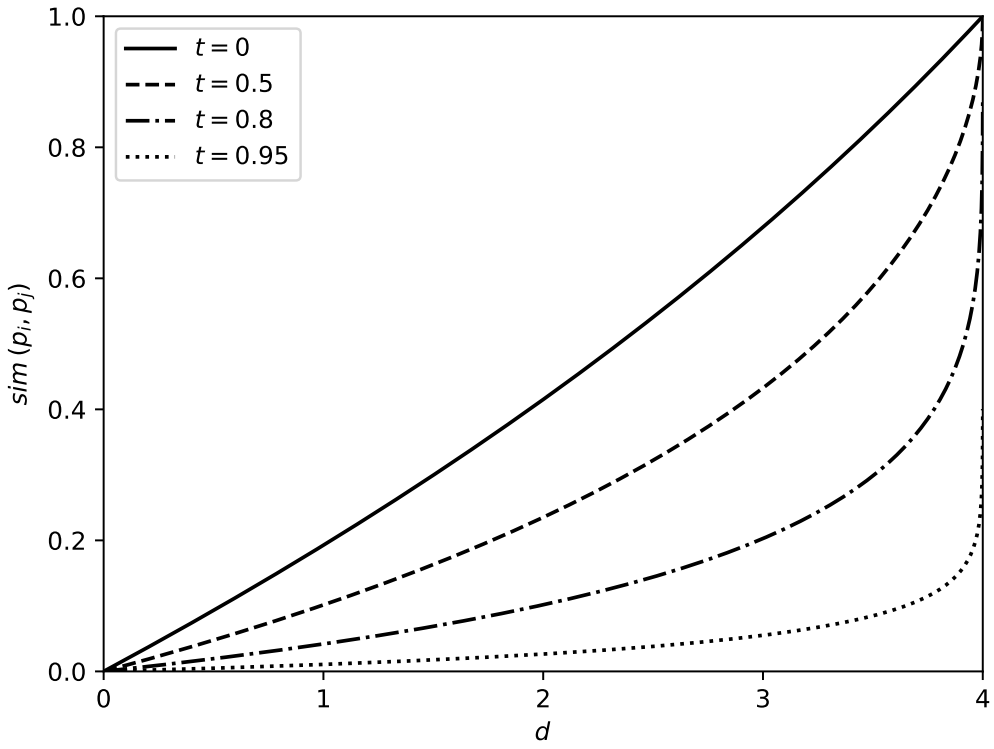


Figure 3.7: Behavior of Eqn: (3.8) as a function of t. $h = 4$.

Chapter 4

Applications

The main motivation behind the development of eROCK and its improvements was *anonymization*. Indeed, the paper [34] was what led me to considering clustering as a preliminary step in an anonymization procedure that should be able to preserve utility better than traditional techniques. The second application that was taken into consideration is *anomaly detection*, mostly in two fields: classification of anomalies and fraud detection.

The two applications have vastly different requirements for what an "ideal" result may look like: anonymization requires clusters whose records are very homogeneous, so that records that belong to the same cluster do not have a large variance in their attributes; for anomaly detection, on the other hand, the homogeneity of clusters is not as important, while as pinpointing which records behave differently from "normal" has a much larger significance.

I also performed simple cluster analysis to understand what were the results of my improvements, so a description of what I've done will be reported here, while the actual results will be in Chapter 5.

4.1 Generic clustering

While debugging the implementation and tweaking the results I performed multiple tests and observed how both eROCK and eeROCK behaved with different parameters and when applied to different datasets. Besides using accuracy measures to have a quantitative measure of the clustering quality, I focused on the number of outliers and on the size of the largest clusters produced by the algorithm. This helped with shedding some light on what some of the drawbacks of the original eROCK implementation are, and how to fix them. In particular, I noticed that, depending on the dataset, the performances may drop noticeably in quality: more specifically, datasets whose entries tend to be "homogeneous" (i.e. whose records tend to have links towards potentially every other entry in the set) are harder to cluster than other, more "well-defined" ones. "Hard" datasets tend to produce a large number of outliers and, in some cases, a very large cluster. With these datasets, tweaking the value of the threshold does not help much, and setting too high a threshold results in a single connected component and a large number of outliers that could not be added to it.

Other observations were relative to the effect of different labeling methods, and to the choice of the number of samples. Some of the results I observed were somewhat counterintuitive, while others confirmed my original thoughts. More importantly, using different $f(\theta)$ showed how the functions I designed produced the same, if not noticeably better results in most conditions.

4.2 Anonymization

4.2.1 Background

As described in Chapter 1, anonymization has become a major field of interest for companies that work with Big Data due to the need for protecting the privacy of personal information. Another necessity is being able to perform data mining on the same personal information. Multiple techniques have been devised to this end (a good survey is [9]), and many of them are variants of either *k-anonymity* or *differential privacy*.

k-anonymity

Definition 4.2.1 (k-anonymity)**.** Given a dataset X with attributes $A_1, A_2, \ldots, A_k$ and a subset of *Quasi-Identifiers* QI, X is said to satisfy the k-anonymity property for the subset of attributes QI if all records belonging to X share with $k-1$ other records the same value for each attribute belonging to QI. *Quasi-identifiers* are all attributes that can be used to re-identify an individual. Different anonymized versions of the same dataset may be produced depending on what subset of QI has been chosen; the choice of QI depends on the dataset and on the target use case.

k-anonymity is widely diffused because of its simplicity: to apply k-anonymity it is sufficient to transform the dataset in such a way that the property is satisfied by means of *generalizations* and *suppressions*. The first kind of transformation consists in "climbing" a hierarchical tree akin to what is shown in Figure 3.6 until the LCA has been reached (this procedure suggested me the idea of using the hierarchical distance); with the latter, an attribute is simply removed. Generalization and suppression are always used when an attribute is categorical; when it is numerical, instead, the value is substituted by the range of the in that class: in every equivalence class, each value of numeric attribute a will be substituted by the range $[min(a_1 \cdots a_k),\ max(a_1 \cdots a_k)]$ (e.g. if an equivalence class contains age values that range from 25 to 33, all ages will be substituted by the string [25,33]). What can be considered a Quasi-Identifier depends on the dataset, on the desired level of privacy and on what the background knowledge of an attacker may be: estimating the latter is very hard and represents an active field of research [20]. In my tests, I considered all attributes except the sensitive one as Quasi-Identifiers.

Example 4.2.1. Table 4.1 is an example of personal data that must be anonymized, where all identifying attributes have been removed (e.g. proper nouns, address, SSN, unique identifier etc.). The **RACE** attribute is *sensitive* and so it shouldn't be anonymized, while all other attributes are treated as *Quasi-Identifiers* so they must undergo transformations

to satisfy the k-anonymity property. For each *equivalence class* of size 3, a generalization is performed for each QI, and a "*" is used to denote a local suppression of the attribute. The resulting dataset is shown in Table 4.2.

Record	**RACE**	EDUCATION	MARSTAT	SEX	AGE
r_1	**White**	Doctorate	Married	M	32
r_2	**White**	Doctorate	Married	M	45
r_3	**Black**	Masters	Married	M	24
r_4	**Asian**	College	Married	F	55
r_5	**Other**	Prof-school	Never married	M	16
r_6	**White**	Prof-school	Separated	F	40
r_7	**Black**	College	Divorced	M	33
r_8	**Black**	Bachelors	Separated	M	55
r_9	**Black**	Bachelors	Widowed	F	84

Table 4.1: This table contains another example of personal records that must be anonymized with 3-anonymity.

Eq. Class	Record	**RACE**	EDUCATION	MARSTAT	SEX	AGE
EQ_1	r_1	**White**	Graduate	Married	M	24-45
	r_2	**White**	Graduate	Married	M	24-45
	r_3	**Black**	Graduate	Married	M	24-45
EQ_2	r_4	**Asian**	Higher education	*	*	16-55
	r_5	**Other**	Higher education	*	*	16-55
	r_6	**White**	Higher education	*	*	16-55
EQ_3	r_7	**Black**	Undergraduate	Spouse not present	*	33-84
	r_8	**Black**	Undergraduate	Spouse not present	*	33-84
	r_9	**Black**	Undergraduate	Spouse not present	*	33-84

Table 4.2: This is the 3-anonymized version of Table 4.1.

k-anonymity is flawed both from the security point of view and from the utility point of view. Multiple attacks can be performed on a k-anonymized dataset; two examples are the *homogeneity attack* and the *background knowledge attack*. In the first case, the sensitive attribute is shared among all the elements in the equivalence class (see EQ_3 in Table 4.1, where the RACE attribute is the same for all records). This makes it impossible to protect the result from information disclosure since an attacker able to identify the equivalence class to which a victim belongs to would immediately be able to learn the value of the confidential value. The second attack is based on the idea that an attacker may have some kind of background knowledge relative to individuals included in the dataset: for example, they may know that their target is 32 years old and has a Doctorate. This

allows pinpointing EQ_1 as the best candidate for containing the victim. A further issue of k-anonymity is that, once an attacker is able to identify the equivalence class a victim belongs to, they would obtain further information about the subject (in this case, that the victim is male and married).

Variations of k-anonymity were proposed to solve some of these issues, such as l-diversity [23] or t-closeness [19]. l-diversity forces each sensitive attribute in an equivalence class to be "well represented": the meaning of "well represented" is vague, and different interpretations as possible (a strict definition requires each class to contain at least l different attributes, while entropy-based definitions require the content of the equivalence class to reflect the distribution of attributes in that class). t-closeness was designed as an improvement over l-diversity, and it requires sensitive attributes in an equivalence class to be distributed in such a way that the distance between the distribution of the attribute in the class and the distribution of the attribute in the entire dataset is smaller than a certain value t.

In my studies, I didn't consider either version and focused on plain k-anonymity, trying to optimize the utility that can be achieved by reducing as much as possible the number of transformations that must be performed to satisfy the k-anonymity property. In [34], it is suggested how to achieve better utility by applying k-anonymity to a data set that has undergone clustering: instead of building equivalence classes from randomly picked records, they are built starting from clusters. This reduces the distortion thanks to how the content of clusters tends to be more homogeneous and this results in a smaller number of generalizations/suppressions to be performed. This choice was motivated by the fact that, despite all its flaws, k-anonymity is still widely used; it was also chosen to understand whether using clustering as a preprocessing step was worth the effort, so that in future work better algorithms than k-anonymity may be employed. I used my eeROCK implementation coupled with a simpler algorithm to test if there were improvements: this is explained in Section 4.2.2.

Differential privacy

Definition 4.2.2 ((ε, δ)-differential privacy)**.** A function κ is said to be (ε, δ)-differentially private with S being the image of κ if, for all data sets X_1 and X_2 that differ in one record, the following is satisfied:

$$Pr(\kappa(X_1) \in S) \leq exp(\varepsilon) \times Pr(\kappa(X_2) \in S) + \delta$$

In simple terms, Def. 4.2.2 can be explained as, given the question "Does record R have attribute A?", the probability that the answer is truthful depends on an exponential distribution with factor ε. Differential privacy has been proven to be secure, although it can only be applied to queries: function κ is used to fetch records from an unmodified database, and it does not perform any kind of transformation on it. Like with k-anonymity, multiple variations of differential privacy have been proposed and implemented, but all of them have the same problem of being applicable to queries only. Because of this, I didn't consider differential privacy for my studies and it won't be a matter for future works since it cannot benefit from the clustering performed by eROCK.

4.2.2 Algorithm

The algorithm I developed applies k-anonymity by performing two passes of the eROCK algorithm (the second uses lowered threshold) and then applying a weaker algorithm that enforces the k-anonymity property on the resulting outliers. The reason why I chose two passes is that, often, eROCK was unable to produce clusters larger than k after the second execution: further use of the algorithm would then result in a waste of resources. Here, eROCK is used to perform pre-processing on the data set. Figure 4.1 shows the operations performed by Algorithm 4.2.2.

Let $S^{(0)}$ be the original data set
Let k be the minimum number size of an equivalence class
$C^{(1)} \coloneqq \text{eROCK}(S^{(0)})$
for all $c \in C^{(1)}$ **do**
  if size(c) $<$ k **then**
    add(c, $S^{(1)}$)
    remove(c, $S^{(0)}$)
  end if
end for
$\theta \coloneqq \theta - \theta/10$
$C^{(2)} \coloneqq \text{eROCK}(S^{(1)})$
for all $c \in C^{(2)}$ **do**
  if size(c) $<$ k **then**
    add(c, $S^{(2)}$)
    remove(c, $S^{(1)}$)
  end if
end for
$C^{(3)} \coloneqq \text{SHORT\_CLUSTERING}(S^{(1)})$
$C^{(4)} \coloneqq C^{(1)} + C^{(2)} + C^{(3)}$
$S_A \coloneqq \text{ANONYMIZE}(C^{(4)}, k)$

The `short_clustering` function calls a lighter algorithm than eROCK, which forces the "leftovers" from the previous two clustering operations into equivalence classes of size k. The algorithm is highly simplified to lower execution time to the detriment of accuracy: the rationale behind its use rests on the assumption that records that couldn't be assigned to a cluster during the previous two steps are very peculiar, so having a very high clustering accuracy would not help because of the large number of generalizations and suppressions required to guarantee the k-anonymity property. The short algorithm Algo. 4.2.2 is described below:

Let S be the set of records to assign to a cluster
Let C be the set of clusters
Let k be the anonymization degree
Let $n = 0$ be the iteration counter
while $n < |S| - k + 1$ **do**
  c_n = build_cluster(S)
  remove(S, c_n)

$n = n + 1$
end while
$c_0 = c_0 + S$

When the last iteration is complete, less than k elements will be left in set S, so elements are arbitrarily assigned to cluster c_0. In function `build_cluster(S)` the distance between a record and all other records is computed and put in an array; the array is then sorted in descending order and the first k records are extracted to form an equivalence class. Since the equivalence class is removed from S, this results in skipping k items at each iteration. The final result is obtaining equivalence classes of size k (with the exception of c_0, which may contain more than k elements). Afterwards, anonymization is achieved by taking subsets of size k or more from each cluster and performing generalization and suppression.

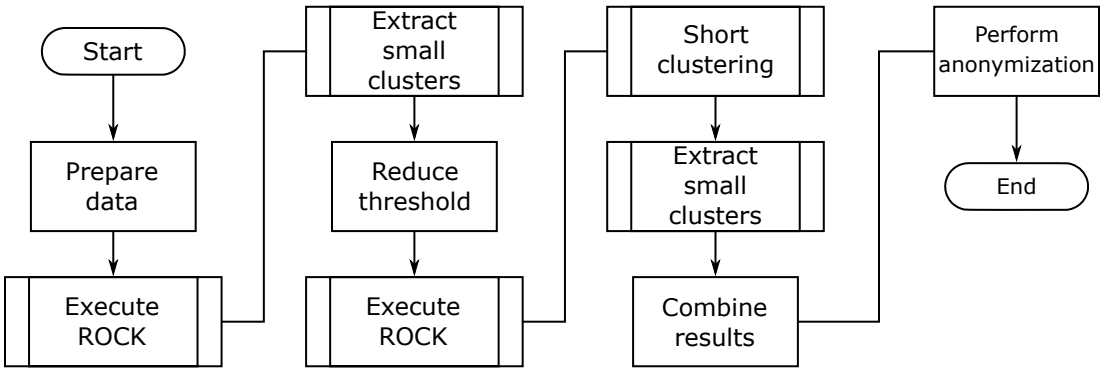


Figure 4.1: The workflow of the anonymization algorithm is presented here.

The results of this algorithm will be presented in Section 5.3. In short, while the quality of the anonymized result was good, performances remain the main issue: the number of outliers may be very large and this results in having to perform clustering on a large number of items in the second and maybe third step. For this application, outliers were kept as such and were not considered as valid clusters when labeling was performed.

4.3 Anomaly detection

The second major application that was considered for clustering is anomaly detection, both in system security and fraud detection. For what concerns system security I was considering intrusion detection, i.e. identifying anomalous traffic that may be caused by an attacker or spotting unauthorized access to functions; fraud detection consisted in flagging fraudulent behaviors in credit card transactions and online advertising. The reason why I used clustering as an anomaly detection tool was in part motivated by papers such as [8], in part due to the behavior of eROCK when outliers were found: the fact it flags outliers as such, instead of forcibly merging them with other clusters appeared to be a very strong point of the algorithm. Unfortunately, this encouraging behavior occurred only in some very specific (and fairly simple) cases, such as a synthetic dataset I used to do some preliminary tests; when I tried to apply my approach to datasets that required a

deeper analysis, the results I obtained showed the limitations of my method.

4.3.1 Clustering as an anomaly detection tool

As already explained, clustering is done to group similar records in clusters: as already explained in Section sec:stateart, clustering algorithms are characterized by the criterion used to assign points to a cluster. Depending on the behavior of the algorithm, there may be three different categories of clustering algorithms that rely on three different assumptions:

- Normal data instances belong to a cluster, while anomalies do not belong to any cluster.
- Normal data instances are close to cluster centroids, while anomalies lie far from centroids.
- Normal data belong to large and dense clusters, while anomalies belong to small or disperse clusters.

eROCK belongs to the first category: when a record is too far from any other cluster to be merged, it is simply skipped. It is possible to do so because, from the start, records are treated as fully-fledged clusters regardless of their size.

4.3.2 My approach

One of the main advantages of clustering as an anomaly detection tool is the fact that clustering is an unsupervised technique, so it can proceed even without preliminary observations, nor training: this is the assumption I was working on during my analyses. In particular, I was trying to understand what results could be achieved by eROCK with a minimal Exploratory Data Analysis (EDA). In general, I did not look for additional features through EDA: I simply studied the dataset and tried to understand whether keeping some features would cause a deterioration of the results, or whether an attribute was simply useless. Unfortunately, I overestimated the performances of the algorithm and expected it to achieve better results than what could be done through such a shallow study.

I observed that categorical attributes that may assume a very large number of different values (such as IDs) tend to harm the result of the clustering operations because the similarity score of two records will be reduced. For this reason, before performing the clustering operation I dropped columns that contained features that showed this behavior. Other problematic features are those that contain a very large number of missing values: these can be handled by inserting “dummy values”, or by removing the feature altogether. In my studies, I never attempted to add more features for the algorithm to work on (differently from what is done in the literature) to understand what the performances of the algorithm were when clustering was done on the “bare” dataset.

A major issue of eROCK is that it performs quite poorly when a large fraction of the features is numerical, or when there are no categorical attributes at all. In these cases,

eROCK does not behave nearly as well as traditional algorithms: this is one of the reasons why I attempted to perform clustering without additional features. In fact, attempting to add new numeric features to the dataset does not help the algorithm. I did not attempt to build additional categorical features (this may be a part of future work).

The main reason why eROCK performs poorly is that anomalies in real datasets require a higher level of analysis for which a training step is necessary: while eROCK can identify records whose features differ heavily from others, anomalous records may need to be recognized through patterns that eROCK does not notice. How eROCK behaves may still be of use in some cases where extracting records with strongly anomalous features may be beneficial; for example, during anonymization outliers are very problematic because they would force an algorithm to introduce stronger generalizations than what would be necessary if they were not present. By identifying those outliers, an analyst may decide to drop them altogether in order to reduce the drop in utility they would cause.

Chapter 5

Results

This chapter will be divided into three parts: firstly I will report the results and the observations I made while clustering data without considering either anomaly detection or anonymization; in the second part I will talk about the results I obtained using my k-anonymity algorithm, comparing it with what can be achieved using an open source tool (ARX [29]) and with a different anonymization algorithm called 3PHA [22]; finally, I will present the results of my attempts at performing anomaly detection. A further introductory section explains the quality measures I used and introduces the datasets I employed to perform my studies. Quality measures were already introduced briefly in Section 1.2; datasets will be described more thoroughly here.

5.1 Datasets and accuracy measures

5.1.1 Datasets

I studied multiple different datasets with the aim of understanding where ROCK can perform well and whether my changes added some improvement or not. Since the main issue was clustering categorical data, I focused on datasets that contained mostly this type of attributes. The tests I performed depended on the dataset I was considering: for example, some of them could not be used to perform anonymization (in fact, only the *IHIS* and *Adult* ones were suitable for this), while others could not be used to perform anomaly detection. Other datasets were specifically designed for performing anomaly and fraud detection: these datasets were mostly generated for ML challenges and for this reason they should undergo an extensive Exploratory Data Analysis, while I simply ran eROCK on them ignoring this step (with the exception of dropping problematic and/or useless columns). It's important to note that, before testing datasets larger than 30000 records, I sampled them to keep the execution times manageable when running the algorithm. Most datasets contain mixed features (i.e. that include both categorical and numerical attributes).

Smaller datasets were used mostly as benchmarks to understand the performances of the algorithm and for troubleshooting purposes. Although there is not much information to gather from them, they still can provide an idea of the performances and help with

tweaking parameters for more meaningful trials.

In Table 5.1 I report the main datasets I studied, together with their source and some data relative to them:

Name	Source	Samples	Classes	Use
agaricus-lepiota	UCI [21]	8124	2	Classification
adult	ARX [29]	30161	2	All
ihis	ARX [29]	1193505	2	Anonymization
KDD Cup 1999	KDD UCI Archive [2]	494020	23	Anomaly detection
clickfraud	SMU LARC [26]	2598816	3	Anomaly detection
ETD simulation	SAP [4]	1466664	2	Anomaly detection

Table 5.1: List of datasets used to perform experiments.

Exploratory data analysis may help with the choice of parameters (especially the number of clusters and the value of the threshold) by finding how "homogeneous" records belonging to a dataset are: in fact, if records are very homogeneous and θ is close to 1 the number of clusters may be too large and the algorithm may fail at finding a suitable number of links between records. In the opposite case, having easily distinguishable clusters means that a larger value for the threshold should be chosen. The number of clusters to be used is similarly difficult to infer and similarly dependent on how homogeneous records are: if records are easily distinguishable the number of clusters needed to produce good classification results may be very small, while in other cases the "sensible" number of clusters may be larger than a few hundreds depending on the actual dataset.

5.1.2 Quality measures

Clustering quality measures

I measured the quality of the final result with different quality measures provided by the scikit-learn Python library [27]: *homogeneity score* (H), *completeness score* (C), *V-M measure* ($V - M$) [31], and *Adjusted Mutual Information score* [37]. These measures are used when classes are binary and they all require ground truth to produce a result. As a consequence, they cannot be employed when a data set does not provide classes, or when multiple valid classes are present. Homogeneity scores a clustering result depending on how homogeneous clusters are, i.e. observing whether clusters contain only members of a certain class. Completeness, conversely, gives a score that depends on whether all points belonging to a class belong to the same cluster. The V-Measure is the harmonic mean between completeness and homogeneity:

$$V = 2 \cdot \frac{(H \cdot C)}{(H + C)} \tag{5.1}$$

Finally, the Adjusted Mutual Information score is an adjustment of the Mutual Information score made to account for chance, needed because the MI score is in general larger for

a clustering result that contains a very large number of clusters even if the actual amount of information is smaller. For two clusterings U and V, the AMI score (like other Mutual Information based scores) indicates the agreement between two assignments ignoring permutations and is defined as follows:

$$AMI(U,V) = \frac{MI(U,V) - E(MI(U,V))}{max(H(U),H(V)) - E(MI(U,V))} \tag{5.2}$$

All the previous measures are normalized so that the maximum value is 1 (perfect clustering). For the AMI score negative scores may be possible in case the classification is random, while for H, C and $V - M$ the lower bound is 0.

I did not use traditional quality measures such as Precision, Recall and Accuracy because they are strictly binary measures: for a binary classification, they will require a perfectly binary classification (i.e. only 2 clusters). Both ROCK and eROCK do not work well with this: since they are hierarchical algorithms, climbing the dendrogram until there are only two clusters produces very poor results in general. In fact, in some cases this may not even be possible, with both algorithms stopping way before that because all clusters will be disjoint.

Quality measures for anonymization

For what concerns anonymization all the scores described so far become meaningless since they all require a ground truth to produce a result, thus they cannot be applied to unclassified data sets. Moreover, since anonymization does not involve classification, such measures cannot be used to judge the quality of the result produced by an anonymization algorithm. To solve this problem, I had to use quality measures able to measure the two quantities I was trying to improve with my approach, namely the loss in utility caused by the application of k-anonymity and the disclosure risk records undergo after anonymization. In the rich landscape of measures that were proposed to measure the quality of an anonymization result [38], I chose the following two: the Global Certainty Penalty described in [40] and the Estimated number of Correct Matches used in [7]. GCP was chosen because, besides the fact that it provides a very good indication of the amount of information loss, it can also account for attributes with different weights. ECM was chosen because of its simplicity and because it quantifies the disclosure risk independently of the amount of background information an attacker may have.

Global Certainty Penalty The Global Certainty Penalty describes the amount of information that was lost because of anonymization, by producing a number between 0 and 1, with 0 meaning *no information loss* and 1 *complete information loss*. It is based on the concept of Normalized Certainty Penalty (NCP), which allows measuring the level of generalization of an attribute while taking into consideration its weight. The NCP has two different formulations depending on whether a feature is categorical or numeric. Given a table T with Quasi-Identifiers $(A_1, A_2, \ldots, A_n)$ and a tuple $t = \{x_1, x_2, \ldots, x_n\}$ that is generalized to a tuple $t' = \{[y_1, z_1], [y_2, z_2], \ldots, [y_n, z_n]\}$, such that $y_i \leq x_i \leq z_i \quad \forall 1 \leq i \leq$

n. The Normalized Certainty Penalty of attribute A_i will be

$$NCP_{A_i}\{t\} = \frac{z_i - y_i}{|A_i|} \tag{5.3}$$

where $|A_i| = max_{t \in T} A_i^{(t)} - min_{t \in T} A_i^{(t)}$.

When an attribute A_i is categorical, instead, the NCP is computed in a different way. Consider the following example:

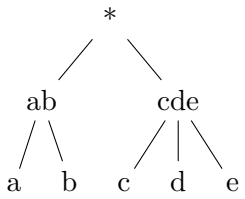


Figure 5.1: Sample hierarchy for NCP with hierarchical values

In this case, the value of the NCP will be computed as follows:

$$NCP_{A_i}\{t\} = \frac{|u|}{|A_i|} \tag{5.4}$$

where $|u|$ denotes the number of leaves of subtree u, while $|A_i|$ indicates the number of values that can be assumed by attribute A_i. Here, subtree *cde* will have $|u| = 1$, while $A_i = 5$. This means that a generalization that puts together a and b yields a penalty of $2/5$, while grouping a and e results in a penalty of 1.

Given an equivalence class E of size k, the NCP of class E will be defined as follows:

$$NCP_E = \sum_{t \in T} \sum_{i=1}^{n} NCP_{A_i}(t) \tag{5.5}$$

A weighted version of NCP_E is the following:

$$WNCP_E = \sum_{t \in T} \sum_{i=1}^{n} w_i \cdot NCP_{A_i}(t) \tag{5.6}$$

NCP_E measures the information loss a single equivalence class experiences, so to measure the total loss that a table is subject to after anonymization the Global Certainty Penalty (GCP) is introduced:

$$GCP_T = \frac{\sum_{E \in T} NCP(E) \cdot |E|}{|T| \cdot n} \tag{5.7}$$

Here, $|E|$ denotes the size of class E, $|T|$ is the size of table T (i.e. the total number of records present in the dataset) and n is the number of attributes of each record in the table. Because of how it is defined, the GCP is 0 when no utility has been lost (i.e. no generalization has been performed at all) and 1 means that there is only one equivalence class that covers the entire dataset, which in turn means that all records have been completely generalized and no information was preserved.

Estimated number of Correct Matches The second measure is a risk measure rather than a utility one: it quantifies the *average number* of correct matches an attacker may gather, considering the case where they are randomly guessing according to $P(r|s)$. Here $P(r|s)$ is the probability of re-identifying a record $r \in \mathcal{R}$ given its anonymized version $\hat{s}$. Given the conditional probability $P(r|s)$, the conditional entropy of the dataset represents the average number of binary questions that must be asked to identify r given s and is defined as follows:

$$H(\mathcal{R}|s) = -\sum_{r \in \mathcal{R}} P(r|s) log_2 P(r|s) \tag{5.8}$$

The Expected number of Correct Matches is then defined as

$$E_{CM} = \sum_{s \in \mathcal{S}} \frac{1}{2^{H(\mathcal{R}|s)}} \tag{5.9}$$

This value should be kept as low as possible and may take any positive value: its value depends on the size of the dataset, so I used the nECM (normalized ECM) by normalizing its value relative to the size of the dataset. The nECM allows having a better idea of the effect of the algorithm when it is applied to different datasets.

Quality measures for anomaly detection

Regarding anomaly detection, I initially planned to mark as anomalous all clusters whose size was smaller than a certain percentage of the total number of records (this percentage depended on the size of the dataset). *Anomalous* and *legitimate* clusters would then be merged with their respective class to obtain a binary classification so that it would be possible to score results using *precision score*, *F-measure* and *AUC-PR*. After performing my tests, however, I noticed that this approach would not work because small clusters may still contain legitimate transactions or operations. For this reason, I employed again the quality measures used for regular clustering.

5.2 General results

To test the algorithm I repeated the clustering operation multiple times varying different factors: the value of the threshold θ, the number of clusters to use, the size of the sample set, what $f(\theta)$ should be used and the number of points to use during labeling.

5.2.1 Comparing eROCK and original ROCK

Here I'll describe the improvements I obtained by using the improvements I described in Chapter 3. To this end, I used the *IHIS*, *adult* and *agaricus-lepiota* datasets. For each trial, I recorded the quality scores described in Section 5.1.2, the number of clusters, the function of θ I used and the variable I was observing (θ, L, number of samples); for some trials I also reported the size of the biggest cluster, the number of outliers and the number of clusters. The most interesting results are reported in the following plots and tables.

Clustering results as a function of θ and number of clusters

Firstly, it's important to describe how do results vary as a function of the parameters that can be changed in the original ROCK algorithm, i.e. the number of clusters to be used and the value of the threshold θ. To this end, I decided to use IHIS, Adult and agaricus-lepiota to show how results may change depending on whether clusters are well defined or not; I chose the first two because they proved to be a hard match for the algorithm since their records are quite similar to each other. The agaricus-lepiota dataset is, conversely, easier to cluster because records are more distinct.

Figures 5.2, 5.3 and 5.4 show the quality measures for a dataset whose classes are easy to spot. For these tests, I chose $k = 22$ as the number of clusters (as was done in [15]). Firstly, it can be noted how all measures are much larger than the corresponding ones for the *adult* dataset, and how curves are much closer to each other compared to other cases. The effectiveness of the new $f(\theta)$ at reducing the number of outliers is also reflected by Figure 5.5. For the *agaricus-lepiota* dataset, the optimal value of θ is around 0.80. Figures 5.6 and 5.7 show how the final clustering quality depends on θ and that there is a maximum around $\theta = 0.74$: this value depends on the dataset, and shouldn't be taken as optimal in general. Here, it is especially evident how the original function $f_0(\theta)$ has worse scores than other functions in most trials; the somewhat surprising increase in the V-Measure score it experiences as θ becomes larger than 0.74 is explained by Figure 5.10: the number of outliers generated by the algorithm when using $f_0(\theta)$ increases dramatically with $\theta = 0.74$ and, since the V-Measure is an average of Homonegeity and Completeness, it is heavily influenced by the increase in Homogeneity caused by the fact that outliers are considered as clusters containing only a single point and are thus perfectly homogeneous.

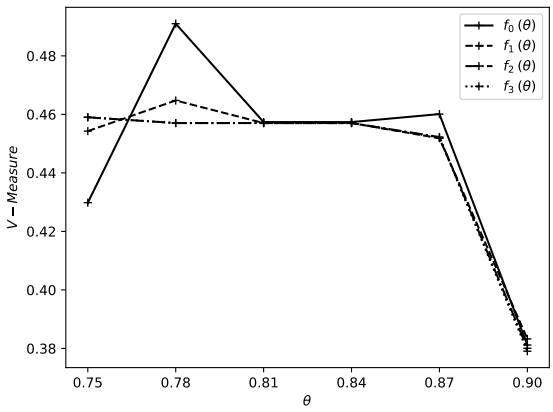


Figure 5.2: V-Measure as a function of θ. *agaricus-lepiota*.

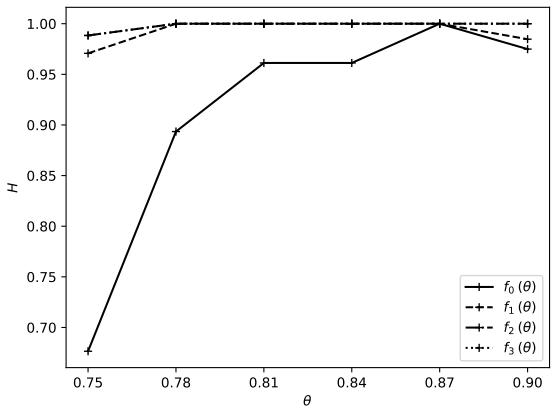


Figure 5.3: Homogeneity as a function of θ. *agaricus-lepiota*.

Figures 5.8 and 5.9 are very important because they show the negative behavior of

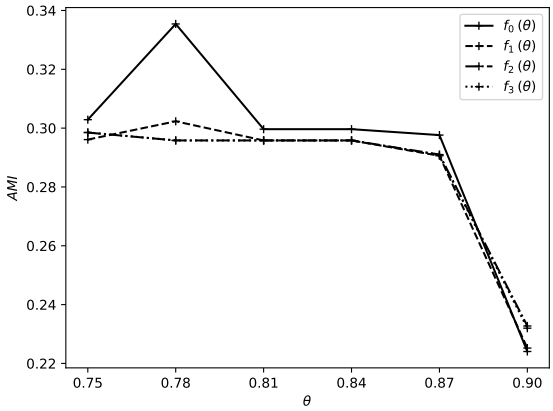


Figure 5.4: AMI as a function of θ. *agaricus-lepiota*.

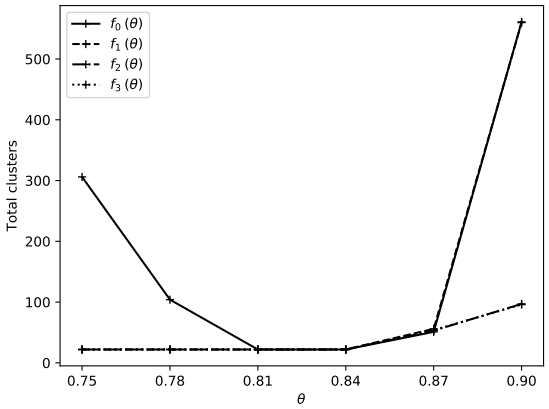


Figure 5.5: Number of clusters as a function of θ. *agaricus-lepiota*.

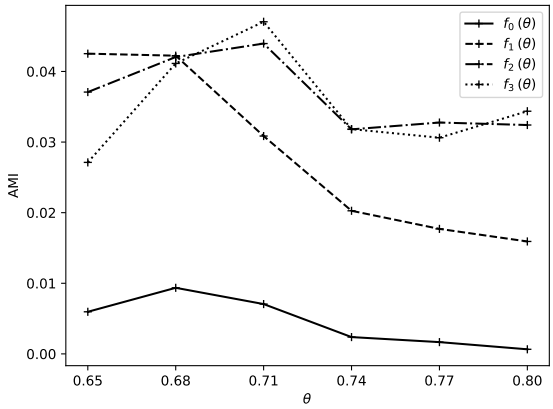


Figure 5.6: AMI score of results as a function of θ. *adult dataset*

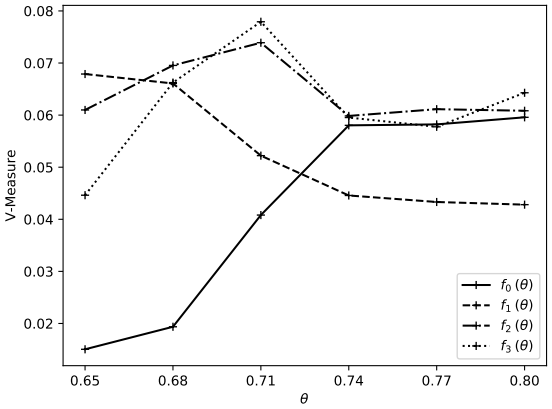


Figure 5.7: VM score of results as a function of θ. *adult dataset*

ROCK when clustering difficult datasets, and the consequences of this behavior. The first figure shows the size of the largest cluster as a function of θ for different $f(\theta)$: it is evident how $f_0(\theta)$ produces a cluster whose size is too large to be of any use, and that contains around 80% of the records in the dataset. Such a result is, clearly, not the desired one; other functions show a more "meaningful" behavior (although $f_1(\theta)$ does approach $f_0(\theta)$ for large values of θ). Producing very large clusters has additional, harmful consequences; one of them is shown in the second figure, 5.9, where the execution time (reported in seconds on the y-axis) is shown: the time necessary to complete the execution with $f_0(\theta)$ is always about double the time necessary to other functions. This is a consequence of the worst case complexity described in Section 2.4.3: having a very large cluster that contains

most points leads to a much larger complexity than what is the average case.

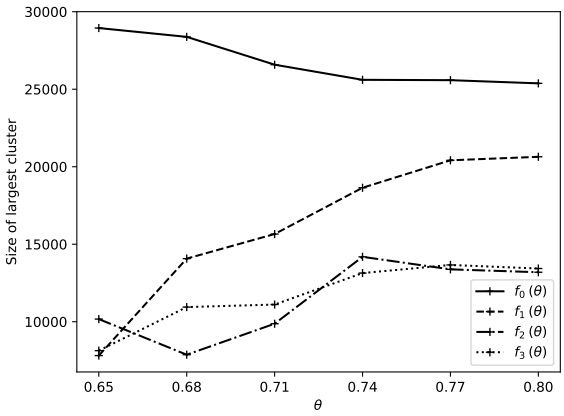


Figure 5.8: Size of largest cluster as a function of θ.

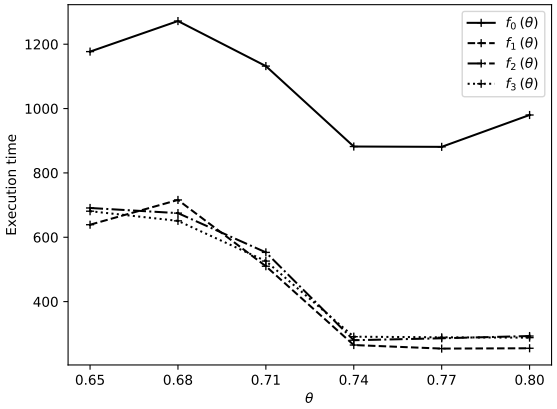


Figure 5.9: Execution time as a function of θ.

Observations on the number of outliers

The behavior I chose for my implementations of ROCK and eROCK makes it so that any point that cannot be labeled is marked as an outlier; for my purposes, I consider as outliers any cluster of size 1 generated during the clustering phase. I've identified 3 possible ways for reducing the number of outliers: changing the number of samples for the clustering phase, changing the number of points to use during the labeling phase and changing $f(\theta)$. These behaviors can be observed in Figures 5.10, 5.11 and 5.12.

In the first figure, the effect of threshold and $f(\theta)$ is evident: when the threshold increases, the number of outliers increases extremely quickly for $f_0(\theta)$, while it is not as fast for other functions. The increase due to θ is caused by the fact that having a higher threshold means that neighbor pairs are harder to form, and this is especially noticeable in the labeling phase. For what concerns the actual function of θ, the cause of the behavior is the fact that large clusters are favored too much over small ones: small clusters will not increase in size and most points will be assigned to large clusters. As a consequence, when the labeling phase starts, the chance of assigning a point to a large cluster will not improve (since, in general, even before labeling starts they already contain more than L points) and it will be very unlikely that a point will be assigned to a cluster that contains only 1 item. Having balanced clusters helps with labeling because a larger fraction of the sample set is represented; moreover, having many very small clusters means that increasing the value of L will not improve the result since large clusters are still not fully represented, while small ones do not experience an improvement in coverage.

In Figure 5.11, the effect of changing the number of samples is shown: somewhat surprisingly, there isn't such a large change in the number of outliers after the first few data points. This is partially due to the fact that the number of outliers is quite small

to start with; it is also noticeable how after the sample size has been increased beyond a certain value the number of outliers tends to remain constant. The latter effect is likely due to the fact that the number of outliers is, by then, simply the number of points that cannot be clustered because they're actual outliers. It must be noted that this should not be considered a general behavior: depending on the size of the complete dataset and on the chosen threshold, 2000 samples may well be far from what is necessary to achieve good results; the choice of how many samples should be used must be done on a use case basis while considering the complexity issue.

How L affects the result will be explained more thoroughly in the following subsection.

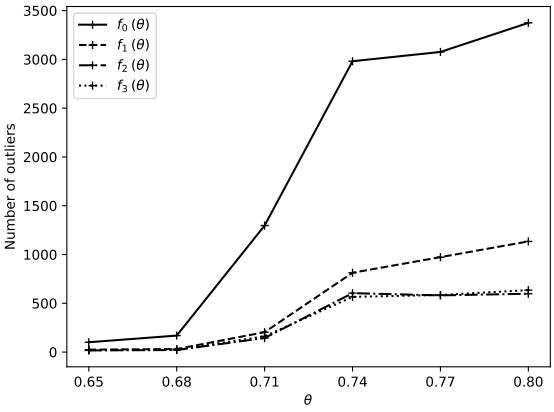


Figure 5.10: Number of outliers as a function of θ. *adult*.

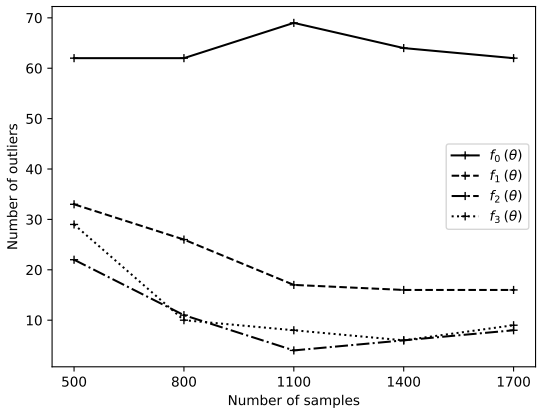


Figure 5.11: Number of outliers as a function of the number of samples. *IHIS*.

Observations on the effects of labeling

As already described in Section 3.2.4, I implemented and tested three different versions of the labeling phase, each one behaving differently regarding outliers and how to consider them. Another possible variation is the method used to decide how the L points are chosen during the labeling step: either the first L points from each cluster are taken for computing the number of neighbors, or L random points are drawn from each cluster. All versions operate differently, but some general behaviors arose. For all following plots, the dataset I used was the *adult* one and the threshold was $\theta = 0.68$, with 2000 samples and $k = 100$. The variable of interest was L, so I tested the behavior for values of L in the range $[40, 140]$ to understand how this would influence the final result.

Firstly, I used L_2 to test the effect of L on the result. It is evident how the number of outliers decreases as L increases, and how much larger the number of outliers for $f_0(\theta)$ is compared to the other functions; $f_1(\theta)$, $f_2(\theta)$ and $f_3(\theta)$ are much closer to each other and overall they produce a much smaller number of outliers. In this case as well, the clustering

result for $f_0(\theta)$ has the issue introduced in Figure 5.8 of containing a single huge cluster. As a consequence, since the majority of points are contained by one cluster, fewer clusters will have a size larger than L, fewer points overall will be considered when processing each unlabeled point and the number of unlabeled points that were successfully assigned to a cluster will decrease (Example 5.2.1 briefly illustrates the issue). This results in having less potential neighbors and more outliers. The downwards trend shown as L increases is explained simply by the fact that, by increasing L, more points are taken from each cluster.

Example 5.2.1. Consider the distribution of points in clusters described by Table 5.2. Assuming $L = 100$, in the first case only 103 points will be considered when evaluating the number of neighbors of an unlabeled point; in the second case all 400 points would be considered, so the chance of finding at least one valid neighbor is much higher. The fact that in the first table the majority of points belongs to the first cluster means that all of them would be alike; consequently, if the unlabeled point is very different from those belonging to C_1, having the full set of 100 points will not help with the assignment. It is still important to note that, even in the perfect case, points may still be marked as outliers: neither ROCK nor eROCK force points into clusters (unlike other algorithms such as k-means).

Cluster	Cluster size
1	397
2	1
3	1
4	1

Cluster	Cluster size
1	100
2	100
3	100
4	100

Table 5.2: Examples of worst case (left) and perfect case (right) scenarios for the labeling phase.

The second observation to be made is relative to the sampling method: random sampling showed a slightly better behavior for $f_0(\theta)$ and little to no change for other functions; this is most likely due to the fact that random sampling allows covering a large fraction of the single cluster compared to other approaches. It should still be noted that this improvement is present only for the first trials; these were made with a value of L that, in my opinion, is too small to produce good results in a real case; for $L \leq 100$ the result is almost the same for both approaches. Something to note when considering labeling with random sampling is the fact that this approach is more likely to consider points that were added during the labeling phase itself: this may lead to a worse performance than what could be achieved by considering only points assigned during clustering. For reasons of complexity during the development phase, I did not implement a more complex check on the cluster size that would allow to, for example, perform labeling only on points belonging to the cluster that was generated during the clustering phase.

The second set of tests was performed using L_3 so that outliers would be considered as valid clusters to study. This approach produced a much smaller number of outliers, as can

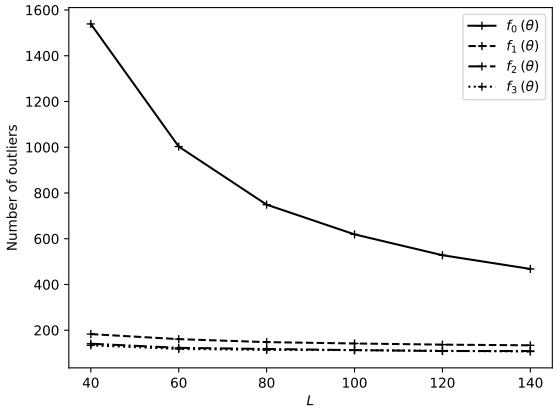


Figure 5.12: Number of outliers as a function of L, sequential sampling. *adult*, L_2.

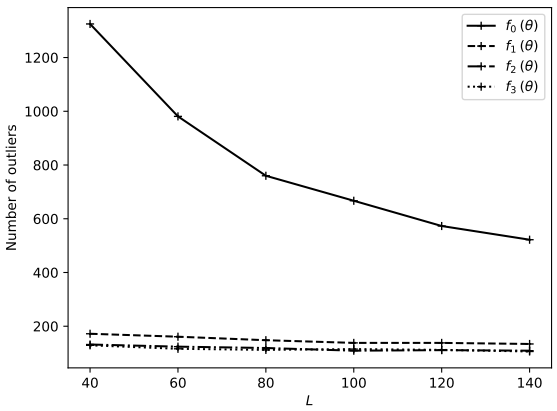


Figure 5.13: Number of outliers as a function of L, random sampling. *adult*, L_2.

be seen from Figures 5.14 and 5.15. In this case, all functions returned a number of outliers close to the average of around 65, and labeling with random sampling produced a somewhat inconsistent behavior with a number of outliers that didn't decrease monotonically and that, instead, increased or decreased depending on the trial. Overall, the Homogeneity score was better using L_3 compared to L_2 (Figures 5.16 and 5.17) while the execution time was, unexpectedly, shorter for L_3 than for L_2 (Figures 5.18 and 5.19): I do not have an explanation for this latter result, especially considering that in many situations L_3 fell in the worst case scenario and required impractically long execution times. Indeed, caution is needed when employing L_3: if the size of the dataset contains many records and the potential number of outliers is potentially very large the use of L_3 may lead to a quadratic complexity in the number of elements present in the complete dataset (this happened in the complete IHIS dataset, for example). When this does not happen, however, L_3 produces the best results out of all the variations I tested.

5.3 Anonymization

For testing anonymization, I considered the adult and IHIS datasets because they are good examples of microdata (i.e. data relative to individuals). For the adult dataset I performed generalization using only the hierarchies that were provided with the dataset, while for IHIS I used both "official" hierarchies and a different one that changed some of the classes: it is important to note this because, as can be seen from Table 5.3, scores change depending on the generalization hierarchy.

I tested the adult dataset with the standard distance function (Section 2.2.1) and with the hierarchical distance (Section 3.3). I also changed $f(\theta)$ and I used the ARX anonymization tool [29] to see how my implementation would compare to it. Performances were a large issue for this dataset (even more for the larger IHIS set), and the average execution time for all trials was around 30 minutes. For what concerns the scores, anonymization

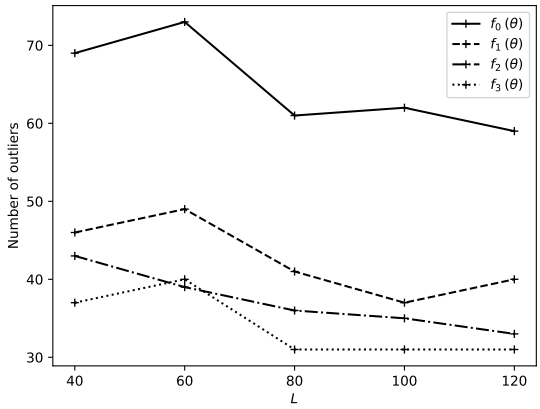


Figure 5.14: Number of outliers as a function of L, sequential sampling. *adult*, L_3.

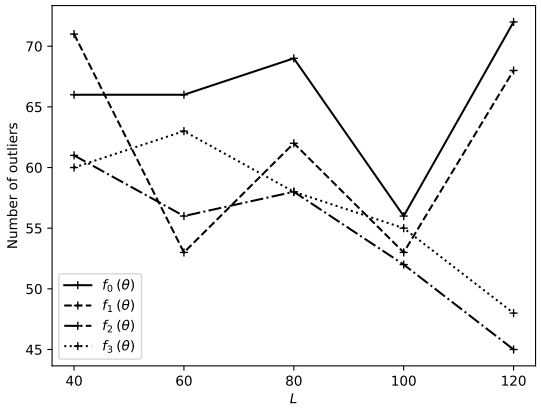


Figure 5.15: Number of outliers as a function of L, random sampling. *adult*, L_3.

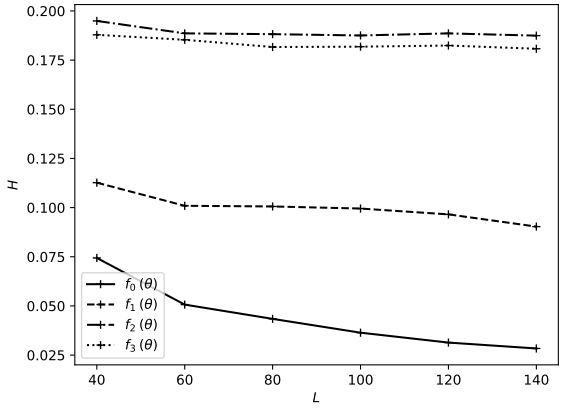


Figure 5.16: Homogeneity measure for L_2. *adult*.

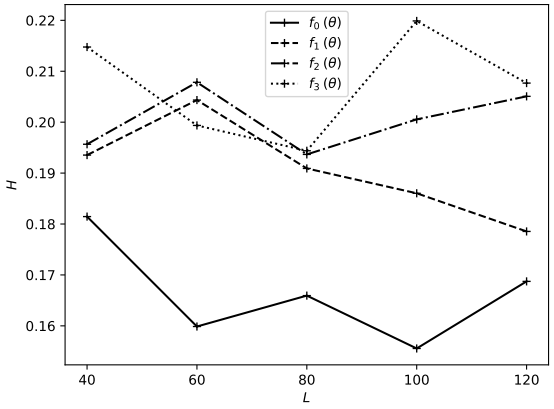


Figure 5.17: Homogeneity measure for L_3. *adult*.

performed using the hierarchical distance had worse performances than what was achieved using the standard Gower distance (this was a fairly disappointing result). The value of GCP for ARX is especially high because the generalization is much more severe than with other methods. The ECM is, unexpectedly, larger for the version anonymized by ARX: my guess for this is that, since the ECM is based on the number of equivalence classes with size exactly equal to k, if there is a large number of such classes records are more in danger because the random choice considers a smaller pool of possible values. Once again, $f_0(\theta)$ performed worse than $f_3(\theta)$: in this case, the reason was probably the fact that it produces a larger number of outliers.

IHIS was tested using the 3PHA algorithm [22], ARX and my algorithm. In this case,

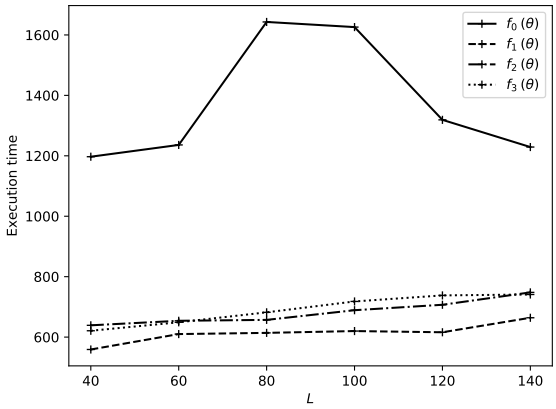


Figure 5.18: Execution time in seconds while using L_2. *adult*.

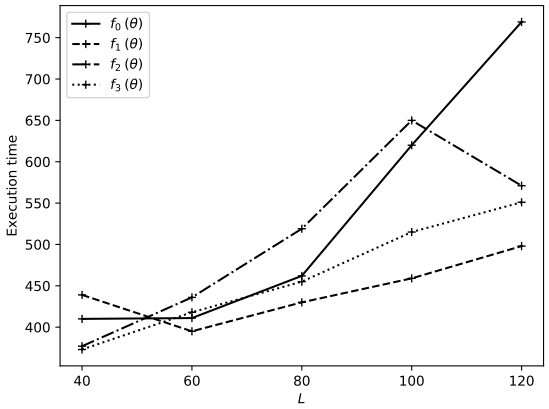


Figure 5.19: Execution time in seconds while using L_3. *adult*.

the execution took an impractically long time for both 3PHA and my algorithm (around 19 hours for the first, 13 for the second), while ARX completed the operation in a matter of minutes. For what concerns the quality scores, ARX has an incredibly small ECM and a very large GCP, 3PHA behaves better than my algorithm from both GCP and ECM points of view. Changing the generalization hierarchy yields a way higher ECM and a lower GCP, so this should be taken into consideration when the hierarchy must be built. I could not draw any conclusion on a "rule of thumb" for creating the hierarchy.

Finally, when using official hierarchies, both GCP and nECM decrease as the number of samples increases: this means that the amount of disruption caused by the anonymization step is less problematic with larger datasets. Overall, while the utility results are good, this remains a simple k-anonymity implementation, which means that all the vulnerabilities and issues presented in Section 4.2 are still present. Future work will involve understanding the impact of different hierarchies on the result and on improving the performances. Further studies will also be needed to understand how to implement the clustering step in the variants of k-anonymity introduced in its section.

Tables 5.4 and 5.5 show the difference between the final result obtained eROCK and the one produced by ARX on the adult dataset. In both tables, the *Country of origin* column was suppressed for space reasons[1] (in the first table, the value was *US* for all rows, while it was suppressed in all rows in the second table). It is especially evident how much more information is lost by ARX compared to eROCK, especially considering that the anonymization coefficient was the same, $k = 5$.

[1]For the same reason, instead of Amer-Indian-Eskimo the value of the Ethnicity column was substituted by A.-I.-E.

Dataset	Records	Notes	ECM	nECM	GCP	$f(\theta)$
Adult	30161	ARX	6806.79	0.225682	0.593076	NA
		Hierarchical d.	1583.05	0.0524868	0.380202	$f_3(\theta)$
		$\theta = 0.75$, hierarchical d.	4121.19	0.13664	0.373554	$f_3(\theta)$
		Gower d.	2204.57	0.0730935	0.332064	$f_3(\theta)$
		Gower d.	3364.34	0.111546	0.370152	$f_0(\theta)$
		Hierarchical d.	4524.61	0.150015	0.443811	$f_0(\theta)$
IHIS variant	1193505	3PHA	46814.7	0.0392246	0.153527	NA
		Hierarchical d.	47373.5	0.0396928	0.225684	$f_3(\theta)$
		ARX	84.2484	7.059e-05	0.47619	NA
IHIS official	1193505	Gower d.	47194.6	0.0395429	0.323456	$f_3(\theta)$
		Hierarchical d.	47498.2	0.0397973	0.412694	$f_3(\theta)$

Table 5.3: Quality results of k-anonymity algorithm on Adult dataset. All cases where a $f(\theta)$ is reported were realized with eROCK.

Sex	Age	Ethnicity	MARSTAT	Education	Work class	Occupation	Salary
F	21-47	A.-I.-E.	No spouse	High School	Private	Nontech	<=50K
F	21-47	A.-I.-E.	No spouse	High School	Private	Nontech	<=50K
F	21-47	A.-I.-E.	No spouse	High School	Private	Nontech	<=50K
F	21-47	A.-I.-E.	No spouse	High School	Private	Nontech	<=50K
F	21-47	A.-I.-E.	No spouse	High School	Private	Nontech	<=50K

Table 5.4: Result of anonymization with $k = 5$ as performed by eROCK.

5.4 Anomaly detection

For the anomaly detection part, I analyzed 3 different datasets: two of them focusing on intrusion detection (KDD and ETD simulation), while the last contained records for a challenge on fraud detection in online advertisements. During the testing phase, I realized I had unreasonably high expectations of what ROCK (and eROCK) could do, and the results I obtained when clustering both KDD and clickfraud showed this. The ETD simulation dataset was too simple and the reason why ROCK worked so well was due to the fact that its classes are extremely easy to distinguish.

Still, some tests were made on the KDD dataset and an acceptable result was obtained (shown in Table 5.6. Before performing the clustering operation, most columns were dropped and the weights of the surviving ones were tweaked to favor some over others. Most clusters were not pure; some of the attacks were correctly placed in a single cluster; some of the very small clusters contained only very rare attacks. There is a major issue with this dataset: in this case, normal connections are the minority, with since only 20% of the records being normal traffic; for this reason, the assumption of "anomalies being fewer than normal behavior" breaks down and the result is that the algorithm is performing

Sex	Age	Ethnicity	MARSTAT	Education	Work class	Occupation	Salary
F	*	A.-I.-E.	*	*	*	Nontech	<=50K
F	*	A.-I.-E.	*	*	*	Nontech	<=50K
F	*	A.-I.-E.	*	*	*	Nontech	<=50K
F	*	A.-I.-E.	*	*	*	Nontech	<=50K
F	*	A.-I.-E.	*	*	*	Nontech	<=50K

Table 5.5: Result of anonymization with $k = 5$ as performed by ARX.

classification rather than anomaly detection. What makes this task even harder is the fact that normal traffic contains all possible protocols, while attacks tend to use only one avenue of attack (ICMP, UDP etc.). For these reasons, eROCK did not perform particularly well. The results obtained here, however, should not be discarded since they represent a fairly successful preprocessing step, that could lead to later, more accurate studies.

Cluster ID	Element count	Unique classes	Most represented	Frequency
0	9	2	teardrop	5
1	48	3	normal	39
2	57	2	neptune	55
3	19	3	portsweep	11
4	8	1	normal	8
5	2865	6	neptune	1127
6	10	1	normal	10
7	76634	19	normal	72992
8	17462	9	normal	16206
9	106950	8	neptune	105853
10	8	2	neptune	4
11	7192	15	normal	6929
12	285	4	neptune	138
13	282443	7	smurf	280758
14	31	3	portsweep	19

Table 5.6: Result of clustering applied to the KDD dataset. $\theta = 0.65$, $k = 15$.

I then tested the ETD simulation dataset employed in [4]; this dataset was synthetically generated to simulate a log file that reports the transactions performed by different users with different applications. Most transactions are legitimate, some are malicious (precisely, one of the users uses a protected application); only 4 meaningful fields are present: user ID, applications, transmitted bytes and received bytes. In this case, clustering results were perfect simply because of how the distance function works: by choosing a suitable threshold, the algorithm was easily able to find classes by grouping those records that had the same user id and transaction. Changing the weights so that those two attributes had more importance than the others made this behavior even more obvious. In this case, I

clustered the complete dataset and was able to isolate all 11 malicious transactions (out of about 1.4M records) as outliers: since the sample used during the clustering phase did not contain any of them, they were all marked as outliers during the labeling phase. Moreover, clusters were perfectly homogeneous, each of them containing all elements of a class and nothing else. This led me to overly optimistic hopes for eROCK and should be considered as an example of a dataset that is simply "too easy" to cluster. Table 5.7 shows the clustering results; the synthetic nature of the dataset is especially evident here, from how perfectly partitioned clusters are.

Cluster number	Legitimate transactions	Anomalies
0	293334	0
1	293323	0
2	293334	0
3	293334	0
4	73332	0
5	73332	0
6	73332	0
7	73332	0
[8 – 18]	0	11
Total	1466653	11

Table 5.7: Result of clustering on ETD simulation dataset.

The final dataset I considered was the clickfraud [26] one. This was provided as a challenge dataset by an actual online advertisement company, that was looking for a ML-based approach to anomaly detection that would be able to identify frauds and mark them as such. The dataset contained information about user IP, device, advertiser ID, country, time, category. It also contained a huge amount of missing data for some features. Before performing clustering I dropped some useless features and changed missing values so that they would be considered as valid categorical entries. In this case, the clustering result not very good: indeed, while some small clusters were perfectly pure, no clusters were complete and, in general, most clusters were fully mixed (the largest ones in particular). In this dataset, the class feature may contain three different values, OK, Fraud and Observation. To apply the quality measures I considered the case in which all observations should be considered as "OK" and the case in which they are considered as "Fraud", and reported the results in Table 5.8. Quality measures are not encouraging and, in this case, it is not possible to apply other measures such as AUPRC or Precision because there are more than 2 clusters; since the size of clusters varies a lot, and clusters that contain "OK" entries may be very small, filtering the clusters by size is not the correct approach.

Overall, the approach that I used for performing anomaly detection did not produce good results: usually, when anomaly detection is performed (and, in general, when it is necessary to perform data mining) additional features are added to the dataset (this happens in [26] and [18], for example). In my case, I wanted to see if eROCK was able to return good results well without additional features, and with minimal pre-processing:

Case	H	C	V-Measure	AMI
Observations are OK	0.614046	0.022228	0.042903	0.006303
Observations are Fraud	0.615210	0.047518	0.088222	0.015794

Table 5.8: Quality results for clickfraud dataset.

in the end, this did not work out at all. Furthermore, one of the drawbacks of eROCK is how its performances degrade when most features are numerical rather than categorical; it is doubtful that adding new numerical features would cause an improvement in the results. I didn't try to add categorical features, and this will be a matter for future work: "grouping" items by adding categorical classes based on some observations made on the original dataset may improve the final result, especially if weights are used in a suitable way: weights and feature reduction allowed to produce the result shown in 5.6, so accurate studies that take them into account and introduce new features may be able to produce better results than what I have been to achieve so far.

Chapter 6

Conclusions

In this report, I described the requirements I had for a clustering algorithm able to handle categorical data, with the final objective of performing anonymization and anomaly detection. The algorithm I chose to do so was ROCK, designed in [15], because of the promising results it showed when used for categorical data. After completing the implementation of the original algorithm, I looked for ways to reduce its drawbacks and to handle a wider range of situations: $f(\theta)$, GDSTOP; changes to the labeling phase and the hierarchical distance. I tested the algorithm on multiple datasets, then I applied it in both anomaly detection and anonymization. Anonymization results were encouraging, while more work must be done to obtain good results for anomaly detection.

Out of all the improvements I introduced, changing $f(\theta)$ 3.2.1 was the most successful one: the intuition according to which the goodness function wasn't able to penalize larger clusters enough proved to be correct, and the idea of changing $f(\theta)$ so that the computation would produce more accurate scores massively improved the results (particularly in those cases when records are very homogeneous). Developing a $f(\theta)$ that depends on the application may be worth the effort in some cases, although I think that the three different functions I proposed here should, together with the original one, be able to cover most use cases.

GDSTOP 3.2.2 is an interesting addition when the structure of the dataset is unknown: it adds a new, milder termination condition compared to the default two, allowing to have an indication of the number of clusters that can be found. In its current iteration, however, it is not optimal: while it does help having an indication of the number of clusters to be used, an analyst should not blindly rely on it. Indeed, to obtain better clustering results it may be necessary to reach a much smaller number of clusters than what GDSTOP suggests. It is still a useful tool for the first steps of the analysis and for understanding how changing $f(\theta)$ or θ may influence the final result. Once the optimal number of clusters has been identified, however, using GDSTOP does not have an effect on the result.

The hierarchical distance 3.3 was a disappointment: its use is very limited both in scope and effectiveness, since hierarchies are not universally applicable and, when they can actually be employed, the result is often the same or worse than what could be achieved with the "default" distance. Given how influential the hierarchy is when performing the generalization step, designing deeper trees may improve the result and be worth the effort.

This, however, is part of future work.

Somewhat unexpectedly, the changes I made to the labeling phase 3.2.4 were more effective than I expected, particularly the version that allows outliers to be valid clusters. While not taking into consideration new points when they are added to clusters may allow for an easy parallelization, the loss in accuracy does not warrant the improvement in performances; when comparing the latter two versions, instead, there may be two reasons for considering only clusters built during the actual clustering phase: either it may be of interest to keep all outliers as such, so that in a later step it may be possible to analyze the entire set of outliers, or the number of unassigned points may be so large that the increase in complexity makes the last version impractical. The latter issue was strikingly evident when the complete dataset was very large: in this case, the main problem is the size of the sample set since by having a sample that represents poorly the complete dataset, a large number of outliers will be produced.

eROCK showed very good clustering results when compared to ROCK (represented in most trials by the use of $f_0(\theta)$), especially in those cases where records were highly homogeneous. What was reported in Section 5.2 gives examples of the behavior of the algorithm under different conditions and what the parameters that change its performance are.

Results in the anonymization part are interesting but there is a caveat: the algorithm is still very slow. While both utility and risk measures are quite good, the time required by the execution may be impractical. For this reason, the choice of this method over others must be made keeping the balance between time and utility in mind: any improvement to the implementation of ROCK will, in turn, reduce the execution time of this method while the performances remain the same.

Unfortunately, this was not the case for anomaly detection: while improving the performances of eROCK will reduce the execution time, the quality of the result will not change without a radical modification of the actual algorithm. I was too optimistic when I first attempted to use ROCK as an anomaly detection tool: since any record that cannot be assigned to a cluster will either be marked as an outlier or will remain in small clusters, anomaly detection seemed a logical application of the algorithm. However, this approach has two major drawbacks: firstly, outliers depend heavily on the size of the sample set, and on how well representative of the complete dataset it is; secondly, the distance function is the largest factor in defining whether a point is an anomaly or not and, in its current iteration (Gower's distance), performances are not good enough. The distance function I am currently using is, in fact, unable to find correlations between different values that would allow marking a record as "anomalous" when compared to others. ROCK (and eROCK) lacks this capability: it works by finding the number of attributes in common between different records, without "asking itself questions" about what does the difference mean. This is once again a drawback of the distance function, similar to the issue with missing values described in Example 2.2.1. Given a sufficient preparatory analysis, the use of weights and the introduction of additional features to the dataset (e.g. combinations of other features that give rise to arbitrary classes) it may be possible to "instruct" the algorithm and improve the final result. Developing a distance function that can perform analysis at a higher level is another possible solution. This is, however, material for future

work.

Many topics still require more in-depth analysis: parallelization is still missing, and it is of paramount importance for real world applications and to reduce the accuracy penalty caused by using a sample set that is unrepresentative of the complete dataset; data analysis with the objective of introducing additional features could help achieve better results in some applications; k-anonymity remains a very insecure method for performing anonymization, and applying other techniques requires changes to the algorithm so that their requirements are satisfied; results produced using the hierarchical distance may be improved by designing better generalization trees (this step, however, should not be required in the first place if the technique were as effective as it seemed in the beginning).

Overall, the work I conducted allowed me to test the performances of an interesting algorithm, studying its advantages and drawbacks and apply it to real life datasets. While some of the drawbacks were fixed, performance remains the major issue and this prevents the algorithm from being employed in a Big Data environment. Some of the improvements I made are quite interesting, others were less useful. Clustering and anonymization produced satisfying results, while anomaly detection was not as encouraging. A large amount of future work is still left to do, and fixing some of the remaining issues would make ROCK and eROCK competitive algorithms on the market.

Bibliography

[1] Gower's similarity coefficient. http://www.clustan.talktalk.net/gower_similarity.html. [Online; accessed 14-December-2017].

[2] KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999. [Online; accessed 1-December-2017].

[3] PAM (partitioning around medoids). In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, page 2012. Springer US, 2009.

[4] Claudio Agostino Ardagna, Valerio Bellandi, Paolo Ceravolo, Ernesto Damiani, Michele Bezzi, and Cédric Hébert. A model-driven methodology for big data analytics-as-a-service. In George Karypis and Jia Zhang, editors, *2017 IEEE International Congress on Big Data, BigData Congress 2017, Honolulu, HI, USA, June 25-30, 2017*, pages 105–112. IEEE Computer Society, 2017.

[5] Korra Sathya Babu and Sanjay Kumar Jena. Balancing between utility and privacy for k-anonymity. In *International Conference on Advances in Computing and Communications*, pages 1–8. Springer, 2011.

[6] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[7] Michele Bezzi. An entropy based method for measuring anonymity. In *Third International Conference on Security and Privacy in Communication Networks and the Workshops, SecureComm 2007, Nice, France, 17-21 September, 2007*, pages 28–32. IEEE, 2007.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[9] Josep Domingo-Ferrer, David Sánchez, and Jordi Soria-Comas. *Database Anonymization: Privacy Models, Data Utility, and Microaggregation-based Inter-model Connections*. Synthesis Lectures on Information Security, Privacy, & Trust. Morgan & Claypool Publishers, 2016.

[10] Mala Dutta, Anjana Kakoti Mahanta, and Arun K. Pujari. QROCK: A quick version of the ROCK algorithm for clustering of categorical data. *Pattern Recognition Letters*, 26(15):2364–2373, 2005.

[11] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in*

Computer Science, pages 265–284. Springer, 2006.

[12] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996.

[13] Adil Fahad, Najlaa Alshatri, Zahir Tari, Abdullah Alamri, Ibrahim Khalil, Albert Y. Zomaya, Sebti Foufou, and Abdelaziz Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE Trans. Emerging Topics Comput.*, 2(3):267–279, 2014.

[14] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[15] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. *Inf. Syst.*, 25(5):345–366, 2000.

[16] John D Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[17] Bernard J. Jansen, Kathleen A. Moore, and Stephen Carman. Evaluating the performance of demographic targeting using gender in sponsored search. *Inf. Process. Manage.*, 49(1):286–302, 2013.

[18] Arjun Joshua. Predicting Fraud in Financial Payment Services. https://www.kaggle.com/arjunjoshua/predicting-fraud-in-financial-payment-services, 2017. [Online; accessed 20-December-2017].

[19] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis, editors, *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 106–115. IEEE Computer Society, 2007.

[20] Tiancheng Li, Ninghui Li, and Jian Zhang. Modeling and integrating background knowledge in data anonymization. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 6–17. IEEE Computer Society, 2009.

[21] M. Lichman. UCI machine learning repository, 2013.

[22] Luigi Longo. Towards big data with k-anonymity. Master's thesis, EURECOM, January 2017.

[23] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. *L*-diversity: Privacy beyond *k*-anonymity. *TKDD*, 1(1):3, 2007.

[24] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. SciPy Austin, TX, 2010.

[25] Robert C McNamee. Canât see the forest for the leaves: Similarity and distance measures for hierarchical taxonomies with a patent classification example. *Research Policy*, 42(4):855–873, 2013.

[26] Richard Jayadi Oentaryo, Ee-Peng Lim, Michael Finegold, David Lo, Feida Zhu, Clifton Phua, Eng-Yeow Cheu, Ghim-Eng Yap, Kelvin Sim, Minh Nhut Nguyen, Kasun S. Perera, Bijay Neupane, Mustafa Amir Faisal, Zeyar Aung, Wei Lee Woon, Wei Chen, Dhaval Patel, and Daniel Berrar. Detecting click fraud in online advertising: a data mining approach. *Journal of Machine Learning Research*, 15(1):99–140, 2014.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[29] Fabian Prasser and Florian Kohlmayer. Putting statistical disclosure control into practice: The ARX data anonymization tool. In Aris Gkoulalas-Divanis and Grigorios Loukides, editors, *Medical Data Privacy Handbook*, pages 111–148. Springer, 2015.

[30] General Data Protection Regulation. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46. *Official Journal of the European Union (OJ)*, 59:1–88, 2016.

[31] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In Jason Eisner, editor, *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pages 410–420. ACL, 2007.

[32] David Sánchez, Montserrat Batet, David Isern, and Aïda Valls. Ontology-based semantic similarity: A new feature-based approach. *Expert Syst. Appl.*, 39(9):7718–7728, 2012.

[33] Shaoxu Song and Chunping Li. Improved ROCK for text clustering using asymmetric proximity. In Jirí Wiedermann, Gerard Tel, Jaroslav Pokorný, Mária Bieliková, and Julius Stuller, editors, *SOFSEM 2006: Theory and Practice of Computer Science, 32nd Conference on Current Trends in Theory and Practice of Computer Science, Merín, Czech Republic, January 21-27, 2006, Proceedings*, volume 3831 of *Lecture Notes in Computer Science*, pages 501–510. Springer, 2006.

[34] Jordi Soria-Comas, Josep Domingo-Ferrer, David Sánchez, and Sergio Martínez. Enhancing data utility in differential privacy via microaggregation-based k-anonymity. *VLDB J.*, 23(5):771–794, 2014.

[35] Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report.

[36] Latanya Sweeney. Weaving technology and policy together to maintain confidentiality. *The Journal of Law, Medicine & Ethics*, 25(2-3):98–110, 1997.

[37] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for

chance. *Journal of Machine Learning Research*, 11:2837–2854, 2010.

[38] Isabel Wagner and David Eckhoff. Technical privacy metrics: a systematic survey. *CoRR*, abs/1512.00327, 2015.

[39] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[40] Jian Xu, Wei Wang, Jian Pei, Xiaoyuan Wang, Baile Shi, and Ada Wai-Chee Fu. Utility-based anonymization for privacy preservation with less information loss. *SIGKDD Explorations*, 8(2):21–30, 2006.

[41] Ying Zhao and George Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002*, pages 515–524. ACM, 2002.