



APPLICATION OF CLUSTERING TECHNIQUES TO PRIVACY PROTECTION

MISTIC: JOINT UNIVERSITY MASTER IN SECURITY
OF INFORMATION AND COMMUNICATION TECHNOLOGIES

Master thesis report for the studies of Joint University Master in Security of Information and Communication Technologies presented by Julián Alarte Aleixandre and directed by Agustí Solanas Gómez.

Universitat Oberta de Catalunya, Barcelona, 2018

Abstract

Clustering techniques can be used over numerical datasets to create similar groups of elements. A well-known application of clustering techniques is the privacy protection in numerical datasets.

Microaggregation is a data protection method that consists of constructing homogeneous clusters from data. Once the clusters are created, the original data of each member of the cluster is replaced by its centroid. The typical size of the clusters is k and the maximum size is $2k-1$. The higher the k value is, the larger the information loss is and the less the disclosure risks.

Microaggregation ensures the k -anonymity property. In the case of a release of data, this property guarantees that the individuals who are subjects of the data will not be identified while the data continues being useful.

Typical microaggregation algorithms are *MD*, *MNAV* and *V-MNAV*. This work proposes to study the application of other clustering techniques to ensure the k -anonymity, concretely the hierarchical agglomerative clustering algorithm (*HAC*). The main goal of this clustering technique is to build a hierarchy of clusters. Particularly, the hierarchical agglomerative clustering is a bottom up technique. Hence, initially it supposes each observation as a cluster and then it merges pairs of clusters while the hierarchy increases.

*To my wife Ana and my son Julián who encourage and
support me.*

Acknowledgements

I would first like to thank my advisor, Agustí Solanas, for the guidance, encouragement and advice he has provided throughout my time as his student. Without his assistance and involvement in every step in the whole process, this work would have not been accomplished.

I would also express my gratitude to Josep Silva. Under his mentorship I have learnt invaluable lessons.

I must express my gratitude to Mariana Gómez, who helped me with the English correction.

Finally, I must express my gratitude to my wife, my son, my sister and my parents, for their continued support and encouragement during all my years as a student.

Contents

1	Introduction	1
1.1	Context and justification	1
1.2	Objectives	2
1.3	Approach and method	2
1.4	Work planning	3
2	State of the art	5
3	Theoretical concepts	7
3.1	k -anonymity and microaggregation	7
3.2	Microaggregation algorithms	9
3.2.1	MDAV	9
3.2.2	HAC	10
3.3	Similarity between the k -partition elements	11
4	Environment	13
4.0.1	Anaconda	13
4.0.2	Notepad++	15
4.0.3	R	15

5	MDAV	21
5.1	Algorithm	21
5.2	Test	22
5.3	Evaluation	24
6	HAC	27
6.1	Algorithm	27
6.2	Test	28
6.3	Evaluation	29
7	HAC improvements	31
7.1	Maximization of the number of partitions	31
7.2	Linkage criteria evaluation	33
7.3	Cluster creation priority	34
8	MDAV vs improved HAC	37
9	Conclusions	39
	Bibliography	41

List of Figures

1.1	Gantt chart of the project	4
3.1	Linking to re-identify data	8
4.1	Anaconda installation wizard	14
4.2	Anaconda directory selection	14
4.3	Anaconda installation	14
4.4	Notepad++ installation wizard	15
4.5	Notepad++ directory selection	15
4.6	Notepad++ components selection screen	16
4.7	Notepad++ configuration settings screen	16
4.8	R directory selection	17
4.9	R components selection	17
4.10	R configuration mode	17
4.11	R configuration settings screen	18
4.12	R modules mirror selection	18
4.13	R modules installation selection	19
5.1	MDAV test file k -partition	23
5.2	MDAV test file with R	25
6.1	Hierarchical agglomerative clustering algorithm	27
6.2	HAC test file k -partition	29

CHAPTER 1

Introduction

1.1 Context and justification

Nowadays, the amount of data collections containing person-specific information is growing exponentially. There is a great variety of devices that can obtain and store information about ourselves, such as portable electronic devices, smartphones, IoT devices, wearables, etc. This fact makes really difficult to control the available amount of data collections about ourselves, and consequently, it is also difficult to guarantee our privacy protection.

Data holders with limited knowledge need mechanisms that allow them to guarantee the data privacy and confidentiality. On the one hand, data holders have to be able to produce useful anonymous data. On the other hand, the data has to be appropriately protected in order to avoid the identification of the subjects.

For instance, consider a data holder such as a high school, that has private field structured information from its students, is asked to share that information with researchers that are writing a paper about school failure. Is there a way to release such information safely guaranteeing that the students can not be re-identified by comparing that information with other data collections?

Latanya Sweeney proposed a formal protection model named k -anonymity. It ensures that each person contained in the data collection released can not be distinguished from at least $k-1$ people also appearing in the data collection.

The k -anonymity property of a data collection can be achieved through microaggregation algorithms. Microaggregation is a clustering problem whose aim is to cluster a set of points into homogeneous groups with size between k and $2k$.

There are typical clustering algorithms, such as MD , $MNAV$, $V-MNAV$, etc., used to ensure the k -anonymity property of data collections. This work studies the application of other clustering

techniques in order to ensure the k -anonymity property. The application of these clustering techniques is compared with the traditional clustering algorithms used for microaggregation.

1.2 Objectives

The main objectives of this work are:

- Implementation and evaluation of a well-known microaggregation algorithm such as *MDAV*. Once implemented, the algorithm will be evaluated using a typical data collection called *Census*.
- Adaptation of a common clustering algorithm to use it for microaggregation. The selected algorithm is *hierarchical agglomerative clustering*. This algorithm will be implemented and evaluated using the *Census* data collection.
- Optimization of the *hierarchical agglomerative clustering* in order to improve the commonly used *linkage criteria*. These criteria influence the shape of the clusters. Thus, improving the *linkage criteria* will influence the homogeneity of the clusters.
- Development and evaluation of the restrictions needed in order to obtain clusters with minimum size k and maximum size $2k-1$.

1.3 Approach and method

This thesis starts studying a well-known microaggregation algorithm (*MDAV*). This algorithm is implemented and evaluated through a data collection called *Census*. Another well-known clustering algorithm (*HAC*) is introduced in order to study its application to microaggregation. Once this algorithm is implemented, it is also evaluated through the *Census* data collection.

Due to the evaluation of the two algorithms using the same data collection, both algorithms can be compared. Then, many changes are introduced to *HAC* algorithm in order to improve its efficiency.

In conclusion, this thesis mainly develops the following points:

1. **State of the art study:** This phase consists of studying the most important publications related to k -anonymity, *MDAV* algorithm and *hierarchical agglomerative clustering* algorithm.
2. **Algorithm implementation:** Both algorithms, *MDAV* and *hierarchical agglomerative clustering* will be implemented in Python.
3. **Algorithm evaluation:** This phase evaluates and compares both algorithms using the *Census* data collection for different k sizes.
4. **Hierarchical agglomerative clustering improvement:** The main goal of this work is to improve this algorithm in order to obtain a performance similar to the *MDAV* algorithm.

1.4 Work planning

This work has been done in a four-month period, so the time allocation for the development of the project has been optimized to achieve the goal avoiding risks. The project started in February and will end in June 2018.

The phases distributed in the Gantt chart are:

- Initial phase: it involved the topic selection, the contact with the advisor and the advisor confirmation. Once the topic was selected, the objectives and the planning were defined.
- State of the art review: after the topic selection, it was necessary to search and review the state of the art. Then, two algorithms from the bibliography were selected: a typical microaggregation algorithm (*MDAV*), and a clustering algorithm not often used for microaggregation (*HAC*).
- Environment configuration: this phase involved the installation of a distribution of the *Python* programming language, called *Anaconda*. Besides, it was installed a programming language called *R*, widely used for statistical computing and graphics.
- Algorithm implementation and test: at this stage both algorithms, *MDAV* and *HAC* were implemented. Several small test benchmarks were also created in order to test them.
- *HAC* improvements: they involved the development, coding and testing of several possible improvements to the *HAC* algorithm. These adjustments are based on the *HAC*'s linkage methods and the *HAC*'s cardinality methods.
- Documentation: once the adjustments were tested with the *Census* dataset, the next step was to write the master's thesis and document all the processes developed in the project.
- Project delivery: finally, the master thesis was sent to the University for the assessment process.

Figure 1.4 shows the Gantt chart for the work. It includes all the phases and the tasks done.

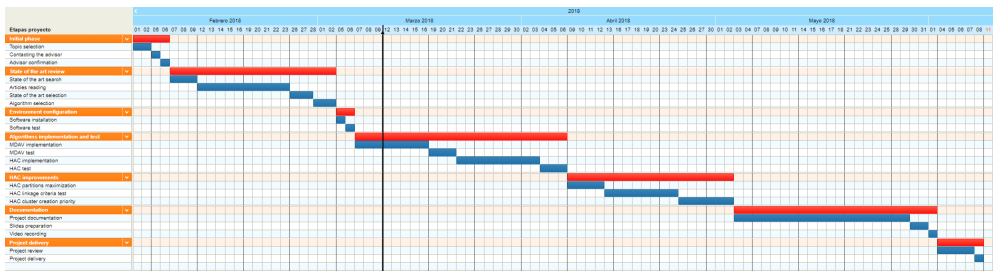


Figure 1.1: Gantt chart of the project

CHAPTER 2

State of the art

The k -anonymity protection model [17], proposed by Latanya Sweeney, ensures that a data transfer satisfies k -anonymity if the information about each person contained in the data transfer cannot be distinguished from at least $k - 1$ individuals whose information also appears in the data transfer.

Initially, k -anonymity property was implemented by removing/generating the values of the quasi-identifier attributes (see definition 3.2). Then, it was proven that it is possible to achieve the same goal by microaggregating the quasi-identifiers, without removing part of the data [5].

Since the 1990's, microaggregation has been used in several agencies and countries, such as the European agency Eurostat [3], the German government [14], and other countries [8].

Microaggregation also contributes to artificial intelligence [6] by increasing the knowledge level of a decision-making system. It is also used in data mining. In this field, it contributes by reducing a data set while minimizing the data loss. Moreover, microaggregation can contribute to protect web log data, for instance, in web-based e-commerce [13].

It should be kept in mind that microaggregation satisfies the k -anonymity condition by clustering a data set in groups of minimum size k and maximum size $2k - 1$. The records of the protected file are replaced by the centroids of each group. Consequently, for a minimum data loss, the homogeneity of the groups should be maximum.

The most common homogeneity measure is the *sum of squared errors* (SSE) [10][9][7], which is the sum of the squares of the distances between the centroid of a group and each record in the group.

The concept of clustering can be defined as a Machine Learning technique that implies the grouping of data points. Given a set of data points, a clustering algorithm can classify each point into a specific group. This classification is based on similar properties or features.

In literature, there are several clustering algorithms adapted and used for microaggregation, for instance:

- Maximum distance (*MD*): it was proposed in [4] as a multivariate microaggregation method. It has a $O(n^3/k)$ computational cost.
- Maximum distance to average vector (*MDAV*): this algorithm, proposed in [5], solves the high computational complexity presented by *MD*. It has a computational cost of $O(n^2)$.
- Variable-size maximum distance vector (*V-MDAV*): it was proposed in [15]. It tries to maximize the homogeneity of the groups by creating variable-size groups. It has a computational cost equal to *MDAV*, $O(n^2)$.
- K-means clustering microaggregation for statistical disclosure control: proposed in [11], this technique starts with one cluster and subsequently partitions the dataset into two or more clusters such that the total information loss across all clusters is the least. It has a computational cost of $O(n^{dk+1})$, where n is the number of records to be clustered.
- Two Fixed Reference Points (TFRP): proposed in [1], this algorithm is divided in two stages called TFRP-1 and TFRP-2. The algorithm has a computation time of $O(n^2lk)$.

This work studies the application of other clustering algorithms, concretely, the adaptation of the hierarchical agglomerative clustering (*HAC*) [2] [12] to microaggregation in order to satisfy the k -anonymity property.

Theoretical concepts

3.1 k -anonymity and microaggregation

In most cases, organizations try to keep the anonymity of the data collections by releasing and receiving personal data without all explicit identifiers such as name, address, postcode, telephone number, email, etc. They assume that the only way to re-identify individuals in the data collection is by using those explicit identifiers. Nevertheless, the remaining data in the data collection is enough to re-identify individuals. This re-identification can be conducted in several different ways. For example:

- Matching or linking the remaining data in the data collection with other data collections.
- Examining the data in order to find unique characteristics from the individuals.

The k -anonymity protection model [17] guarantees that a data transfer satisfies k -anonymity if the information for each individual contained in the data transfer cannot be distinguished from at least $k - 1$ individuals whose information also appears in the data transfer.

Sweeney introduces some definitions [3.1, 3.2, 3.3] in order to explain the k -anonymity protection model.

Definition 3.1 Attributes

Let $B(A_1, \dots, A_n)$ be a table with a finite number of tuples. The finite set of attributes of B are $\{A_1, \dots, A_n\}$.

This definition corresponds to the concept of *attribute* of the relational database model, where each row is called *tuple* and each column is called *attribute*.

Depending on their value as information, personal data can be classified in 3 main categories:

- Sensitive data: these attributes have a special value as information. For instance, medical information, salary, properties, etc.
- Identifiers: attributes that can identify an individual by themselves. For example, id number, phone number, email, etc.
- Quasi-identifiers: these are the remaining attributes. They are not particularly important, but the combination of some quasi-identifiers can be useful to re-identify an individual.

Definition 3.2 Quasi-identifier

Given a population of entities U , an entity-specific table $T(A_1, \dots, A_n)$, $f_c : U \rightarrow T$ and $f_g : T \rightarrow U'$, where $U \subseteq U'$. A quasi-identifier of T , written Q_T , is a set of attributes $\{A_i, \dots, A_j\} \subseteq \{A_1, \dots, A_n\}$ where: $\exists p_i \in U$ such that $f_g(f_c(p_i)[Q_T]) = p_i$.

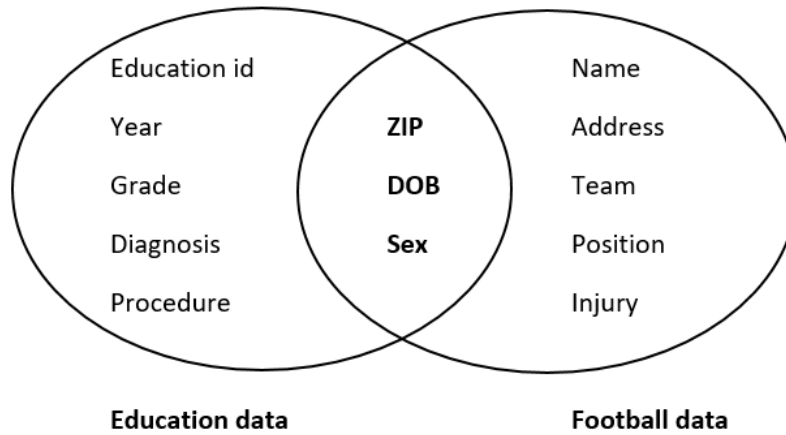


Figure 3.1: Linking to re-identify data

Figure 3.1 shows that it is possible the re-identification process by directly linking shared attributes by two tables. The table on the left contains data about students with special educational needs, while the table on the right contains data about children football teams. Note that there are three attributes contained by both tables.

A quasi-identifier for the table on the right, written Q_F is $\{name, address, ZIP, DOB, sex\}$. If the football data is linked to the education data, as $\{ZIP, DOB, sex\}$ are shared by both tables, and $\{name, address\}$ belong to the soccer data table, it is clear that there is a disclosure of educational data.

We can deduce that quasi-identifiers need to be altered in order to protect the privacy of the individuals. Thus, it will not be possible the re-identification process by using the quasi-identifier attributes.

Definition 3.3 k-anonymity

Let $RT(A_1, \dots, A_n)$ be a table and QI_{RT} be the quasi-identifier associated with it. RT is said to satisfy k -anonymity if and only if each sequence of values in $RT[QI_{RT}]$ appears with at least k occurrences in $RT[QI_{RT}]$.

Year	Grade	ZIP	Sex	Diagnosis
2015	5th	03258	Male	ADHD
2015	5th	03258	Male	School failure
2015	6th	03560	Female	High IQ
2015	6th	03560	Female	ADHD
2016	5th	03528	Female	ADHD
2016	5th	03528	Female	High IQ
2016	6th	03528	Male	School failure
2016	6th	03528	Male	ADHD
2016	6th	03528	Male	High IQ
2017	4th	03560	Female	School failure
2017	4th	03560	Female	ADHD

Table 3.1: k -anonymity where $k = 2$ and $QI = \{Year, Grade, ZIP, Sex\}$

Definition 3.3 implies that a quasi-identifier QI_{RT} must be repeated at least k times in a table in order to satisfy the k -anonymity protection model. Therefore k -anonymity ensures that the information for each individual contained in a data transfer cannot be distinguished from at least $k - 1$ individuals whose information also appears in the data transfer.

Table 3.1 provides an example of a table T that satisfies the k -anonymity property. The quasi-identifier is $QI_T = \{Year, Grade, ZIP, Sex\}$ and $k = 2$. Observe that, for each tuple in table T , the values of the tuple in QI_T appear at least twice.

K -anonymity property can be obtained by microaggregation, which is a clustering problem whose goal is to cluster a set of points into homogeneous groups with size between k and $2k - 1$. Once the groups have been created, each record is replaced by the centroid of its group. Thus, these groups with $size \geq k$ satisfy the k -anonymity property. Note that the within-group homogeneity should be maximized in order to minimize the information loss.

3.2 Microaggregation algorithms

Microaggregation belongs to a group of techniques called SDC (*Statistical Disclosure Control*). SDC are techniques developed in order to preserve privacy in databases. *Disruptive* methods belong to SDC techniques. These methods change the data to maintain the data privacy. k -anonymity is a property related to database privacy and microaggregation is a disruptive method useful to satisfy this property.

3.2.1 MDAV

MDAV (*Maximum distance to average vector*) was proposed in [5] as part of the multivariate microaggregation method implemented in μ -Argus software. It obtains the same efficiency as the *Maximum Distance* (MD) [4] algorithm, but it has a lower computational complexity.

The MDAV algorithm consists is described below:

1. First of all, it computes the centroid ¹ of all the records in the dataset. Then, it finds the furthest record from the centroid (called r) and the furthest record from r (called s).
2. Next, it builds two sets of records near r and s : one set includes r and the $k - 1$ records closest to it, and the other set includes s and the $k - 1$ records closest to it.
3. If there are more than $2k - 1$ records that do not belong to the sets built in step 2, the algorithm returns to step 1, removing from the dataset the records used to build r and s in step 2.
4. If the number of records that do not belong to the sets built in step 2 is between k and $2k$, the algorithm builds a new set with these records and it finishes.
5. But if the number of records that do not belong to the sets built in step 2 is less than k , the algorithm adds each one to the closest set.

When the algorithm has finished, the result is a k -partition of the initial dataset.

MDAV is an extremely fast algorithm. It executes $\lfloor n/2k \rfloor$ iterations and the computational cost of the algorithm is $O(n^2)$.

3.2.2 HAC

HAC (*Hierarchical agglomerative clustering*) is a clustering algorithm that successively merges pairs of clusters until all of them have been merged into a single cluster. Agglomerative clustering is a bottom-up method. Thus, each observation starts in its own cluster and merges pairs of clusters as moving up the hierarchy.

There is also another type of hierarchical clustering called *divisive* clustering. In this approach, all observations start in one cluster and they are splited successively while moving down the hierarchy.

The HAC algorithm consists in the following stages:

1. The algorithm builds a distance matrix between all the records in the dataset. At the initial phase, one cluster will correspond to one record. The distance depends on the metric selected. There are several metrics that can be used for this purpose. The most used are:
 - Euclidean distance.
 - Squared Euclidean distance.
 - Manhattan distance.
 - Maximum distance.
 - Mahalanobis distance.
 - Levenshtein distance.

¹The average record

- Hamming distance.
2. Then, the algorithm examines the distance matrix and selects the closest two clusters.
 3. Once both clusters are selected, the algorithm merges them into one cluster and it updates the distance matrix depending on a linkage criteria between both clusters. There are several linkage criterias, the most common are:
 - Complete-linkage (maximum).
 - Single-linkage (minimum).
 - Average linkage (UPGMA).
 - Centroid linkage (UPGMC).
 - Minimum energy.
 4. The algorithm returns to step 2 and it continues merging the clusters until all the records in the dataset form one cluster.

Compared to MDAV, HAC is significantly slower. The computational cost of this algorithm is $O(n^3)$, while the computational cost of MDAV is $O(n^2)$.

3.3 Similarity between the k -partition elements

The efficiency of microaggregation algorithms can be measured using the SSE (*Sum of Squared Errors*) metric. This metric is computed using the formula:

$$SSE = \sum_{j=1}^g \sum_{i=1}^{n_j} (x_{ij} - \bar{x}_j)^2 \quad (3.1)$$

Therefore, the distance between each data record and the centroid of its group is squared and then summed up.

The goal of microaggregation is to produce groups as homogeneous as possible in order to minimize the information loss. A microaggregation algorithm tries to find a partition that minimizes the sum of within-group squared error. Thus, the lower the SSE metric is, the more homogeneous the groups are.

CHAPTER 4

Environment

The preparation of the environment consists of the installation of the necessary languages and software packages in order to implement and test the microaggregation algorithms.

The software packages used in this project are:

- **Anaconda:** a free open-source distribution of Python and R programming languages. These languages are widely used in machine learning and data science. This package can be downloaded from <https://www.anaconda.com/distribution/>.
- **Notepad++:** it is an open-source text and source code editor. It can be downloaded from: <https://notepad-plus-plus.org/download/>
- **R:** it is a free software package for statistical computing and graphics. It can be downloaded from: <https://www.r-project.org/>.

4.0.1 Anaconda

Anaconda package was installed in a Windows x64 operating system. After executing the downloaded Windows executable file, the installation wizard appears.

The first steps are the license acceptance and the multi-user configuration. Next, the installation directory should be selected.

In the last step, the user has to select if he/she wants to use Anaconda as the default Python and if he/she wants to add Anaconda to the PATH environment variable.

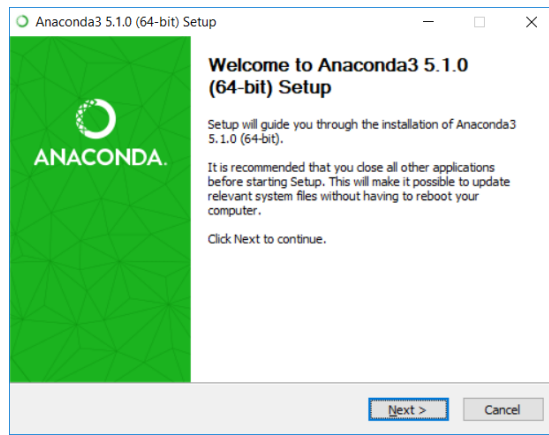


Figure 4.1: Anaconda installation wizard

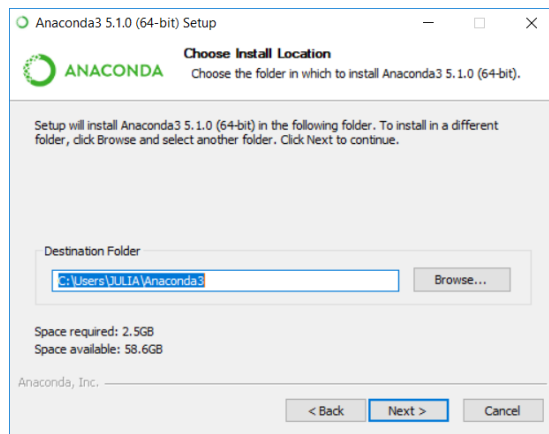


Figure 4.2: Anaconda directory selection

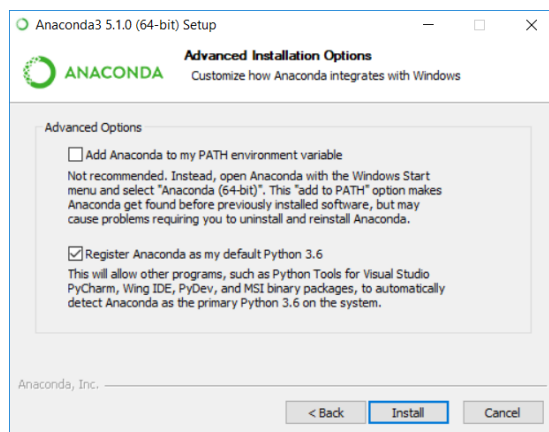


Figure 4.3: Anaconda installation

4.0.2 Notepad++

Any text editor can be used to write Python code. However, Notepad++ was chosen because its features provide multiple benefits for coding in several languages. Note that there are several text editors for coding similar to Notepad++, which have the same or even better features.

Notepad++ package was downloaded and installed in a Windows x64 operating system. After executing the Windows executable file, the installation wizard appears.

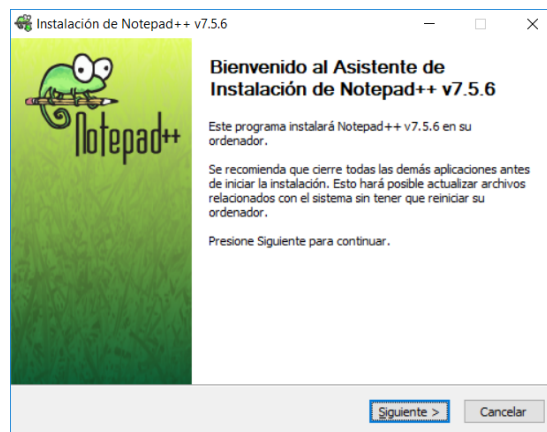


Figure 4.4: Notepad++ installation wizard

After the license acceptance, the installation directory has to be selected.

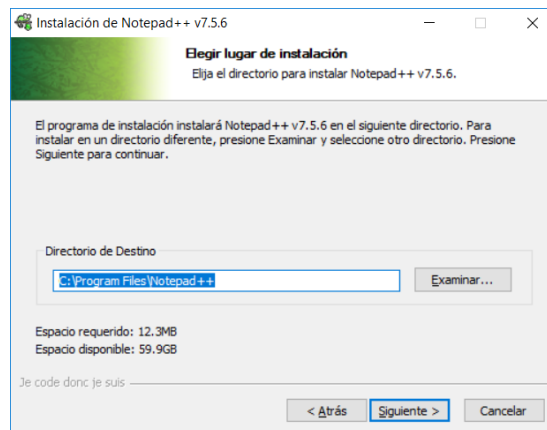


Figure 4.5: Notepad++ directory selection

The following step is the selection of the components to be installed.

To finish, several configuration settings related to the installation path have to be selected.

4.0.3 R

The decision of using R package in order to test the algorithms and plot the clustering results is based on the fact that R package is a very powerful package for statistical computing and

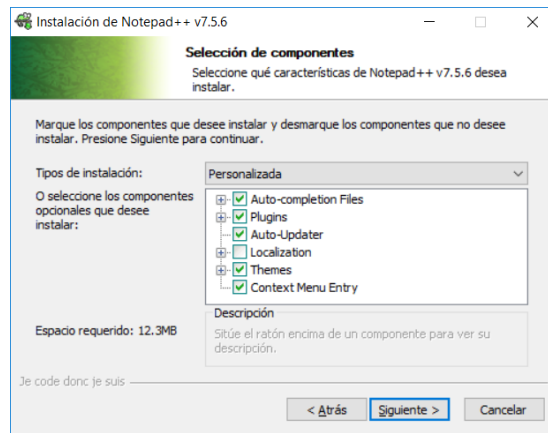


Figure 4.6: Notepad++ components selection screen

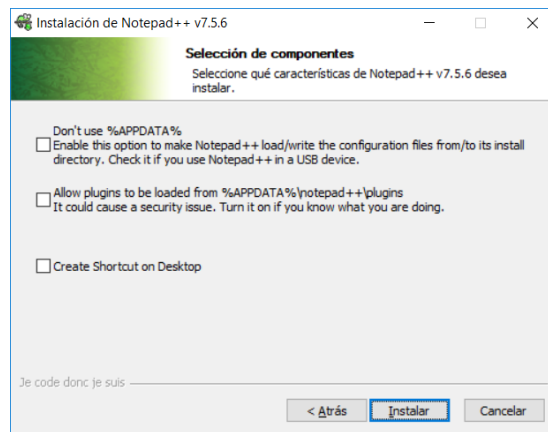


Figure 4.7: Notepad++ configuration settings screen

graphics.

R package was downloaded and installed in a Windows x64 operating system. After executing the downloaded executable package, the installation wizard appears.

After accepting the software license, the installation directory has to be selected.

In the next step the user is asked to select the components to be installed.

Next, the user will be requested to configure the program while installing it.

Finally, several configuration settings related to the Windows environment will be selected.

In order to work with MDAV algorithm, a module called “sdcMicro” must be installed. The “install module” option is located inside the “modules” menu. To continue with the installation of the module, the installation mirror should be selected.

The next step is the selection of the modules to be installed. In this case the module is “sdcMicro”.

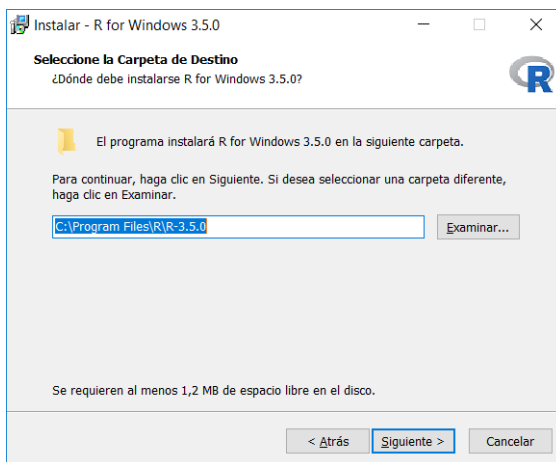


Figure 4.8: R directory selection

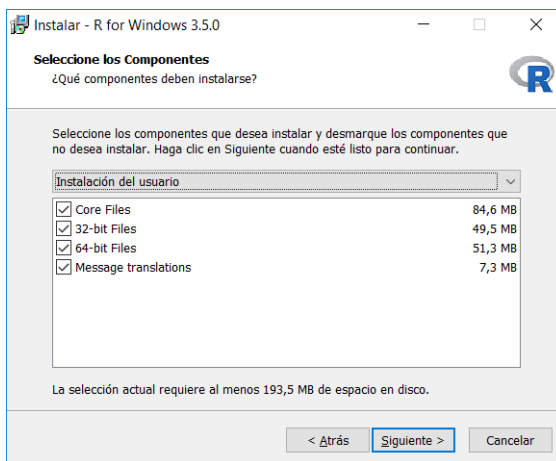


Figure 4.9: R components selection

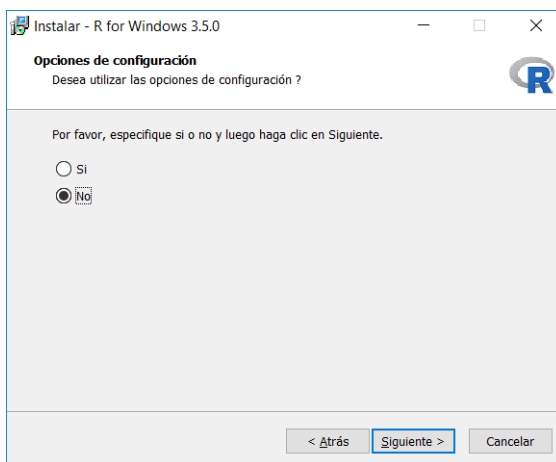


Figure 4.10: R configuration mode

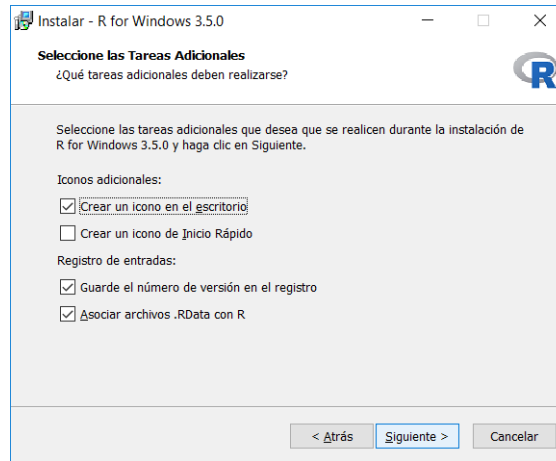


Figure 4.11: R configuration settings screen

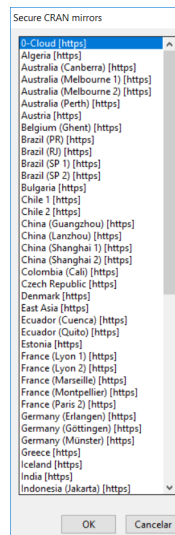


Figure 4.12: R modules mirror selection

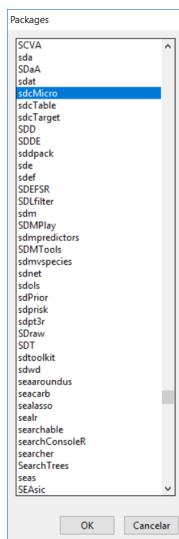


Figure 4.13: R modules installation selection

5.1 Algorithm

The *MDAV* algorithm is based on an heuristic algorithm that clusters records in a microdata file so each that cluster contains between k and $2k - 1$ records. This algorithm is also valid for univariate and multivariate microaggregations.

The algorithm has been implemented in Python following these steps:

1. First, the data has to be standardized to compare the data loss caused by microaggregation. If a variable v_i takes a value x , the standardization algorithm replaces it with $(x - \bar{v}_i)/s_{v_i}$, where \bar{v}_i is the average of the values taken by v_i , and s_{v_i} is the standard deviation of these values.
2. Next, the program computes the centroid of all the records in the dataset. This is done by computing the mean \bar{v}_i of each variable v_i in the dataset. Once the centroid has been computed, it finds the furthest record from the centroid, called r , and the furthest record from r , called s .

The furthest record from the centroid (r) is computed by measuring the Euclidean distance between the centroid and all the records in the dataset. The maximum distance determines the r record. The furthest record from r (s) is computed analogously.

3. In the following phase, two sets of records near r and s are built. The first set includes r and the $k - 1$ records closest. The other one includes s and the $k - 1$ records closest. These sets of records closest to r and s are also computed with the Euclidean distance.
4. Steps 2 and 3 are repeated, removing from the dataset the records used to build r and s in step 2, until there are less than $2k$ records in the dataset.

Var1	Var2
2	7
3	6
1	1
1	4
2	12
4	14
5	8
6	2
7	4
3	3
5	9
6	9
1	3
3	13
6	4
4	6
3	7
2	9
4	10

Table 5.1: Algorithm test file

5. Once the loop has finished, if the number of records that do not belong to the sets built in step 2 is between k and $2k - 1$, the algorithm builds a new set with this records and it finishes. However, if the number of records that do not belong to the sets built in step 2 is less than k , the algorithm adds each one to the closest set.
6. Finally, the program replaces each record by its centroid in order to guarantee the k -anonymity property.

The final result of the algorithm is a k -partition of the initial dataset.

When the algorithm has finished, the SSE (3.1) algorithm can be computed in order to check the homogeneity of the resulting k -partition.

5.2 Test

A small test dataset was created so that we can check that the algorithm works properly. This dataset is composed of 19 records, each one with 2 numeric variables. The dataset was stored in a "csv" file. The contents of the file are shown in table 5.1.

The results of executing the algorithm for the test dataset with $k = 4$ are:

- **SSE:** 7,93
- **Time:** 0 sec.

Figure 5.2 shows the result of executing the *MDAV* algorithm for the test dataset. Each dot colour represents a group of records with size between k and $2k - 1$, so it can be observed that records have been classified into 4 distinct groups (1 of them with 4 elements and the other 3 with 5 elements each).

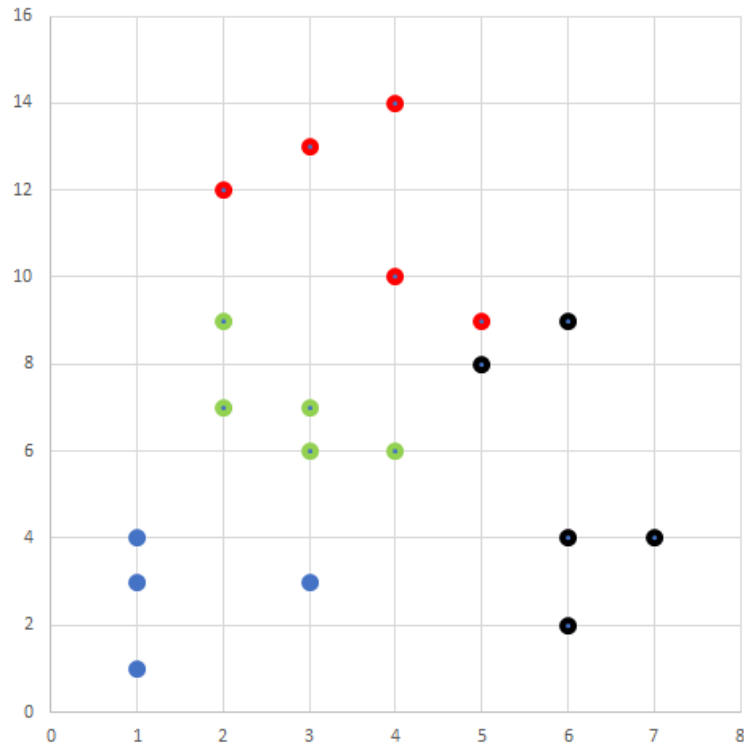


Figure 5.1: MDAV test file k -partition

The same test dataset was evaluated using *R*, the software environment for statistical computing and graphics. A package called ‘*sdcmicro*’ was used in order to compute the *MDAV* algorithm for $k = 4$. The following instructions were executed:

```
temp = read.csv("C:/Users/Julian/Desktop/UOC/r.csv", sep=";")
temp <- as.matrix(temp)
microData <- as.data.frame(temp)
m1 <- microaggregation(microData, method="mdav", aggr=4)
y <- m1$mx
```

Once the instructions were executed, the variable y contained the result of the microaggregation, which is shown in table 5.2:

Figure 5.2 shows the result of executing the *MDAV* algorithm (for $k = 4$) with *R* for the test dataset. It can be observed that records have been classified into 4 distinct groups (1 of them with 7 records and the other 3 with 4 records each).

The SSE was computed by squaring and then summing up the distance between each data record and the centroid of its group (3.1). The SSE obtained was 8,20, which is slightly higher than

Var1	Var2
3.428571	7.428571
3.428571	7.428571
1.500000	2.750000
1.500000	2.750000
3.250000	12.250000
3.250000	12.250000
3.428571	7.428571
6.250000	4.750000
6.250000	4.750000
1.500000	2.750000
3.428571	7.428571
6.250000	4.750000
1.500000	2.750000
3.250000	12.250000
6.250000	4.750000
3.428571	7.428571
3.428571	7.428571
3.428571	7.428571
3.250000	12.250000

Table 5.2: R output for *MDAV* test file

k	SSE	Time (s.)
3	798,44	39
4	1051,28	32
5	1274,83	27
10	1985,65	19

Table 5.3: MDAV algorithm evaluation

the SSE obtained with the developed algorithm. This situation is due to the fact that once the algorithm has finished its main loop, if the number of unassigned records is less than k , both algorithms assign them to the closest cluster in different ways.

5.3 Evaluation

In order to evaluate the *MDAV* algorithm, the *Census* dataset was selected. This dataset includes 13 numeric variables and 1080 records. The experiments¹ were done for several k values (3, 4, 5 and 10). The results for each k value (including SSE and time in seconds) are shown in Table 5.3.

The experiments reveal that lower k values obtain more homogeneous groups, so there is less information loss. On the other hand, higher k values obtain more heterogeneous groups, which indicates there is more information loss.

¹All the evaluation experiments have been done with an Intel Core i7 4770k 16GB RAM

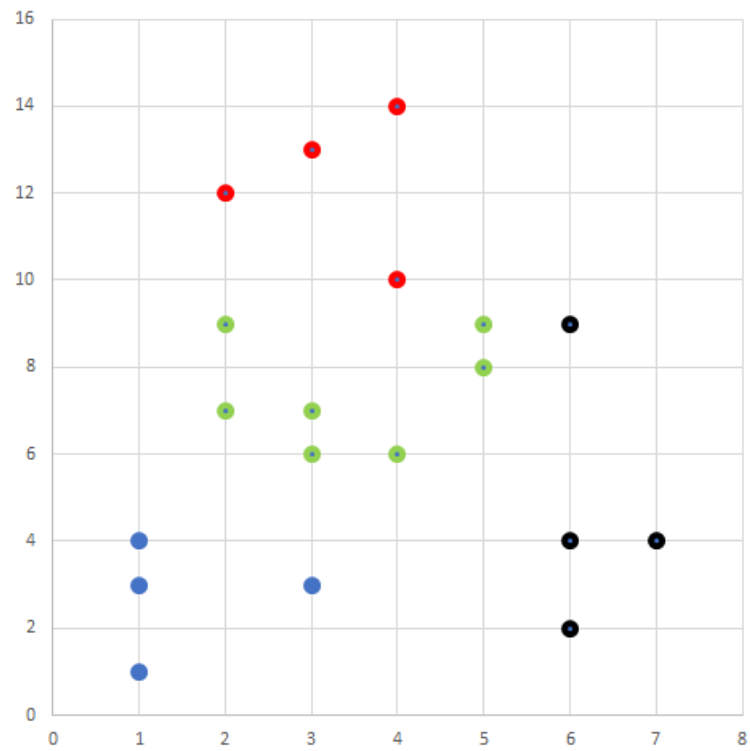


Figure 5.2: *MDAV* test file with R

However, the algorithm is faster for higher k values. For instance, the execution time for $k = 3$ is twice the execution time for $k = 10$.

The obtained results are similar to the results described in [16].

6.1 Algorithm

The *HAC* algorithm involves the successive combination of pairs of clusters until all of them have become a single cluster containing all the records.

Figure 6.1 shows the result of executing the *HAC* algorithm for the test dataset of table 5.1. The algorithm combines the clusters until all of them have been merged into one cluster.

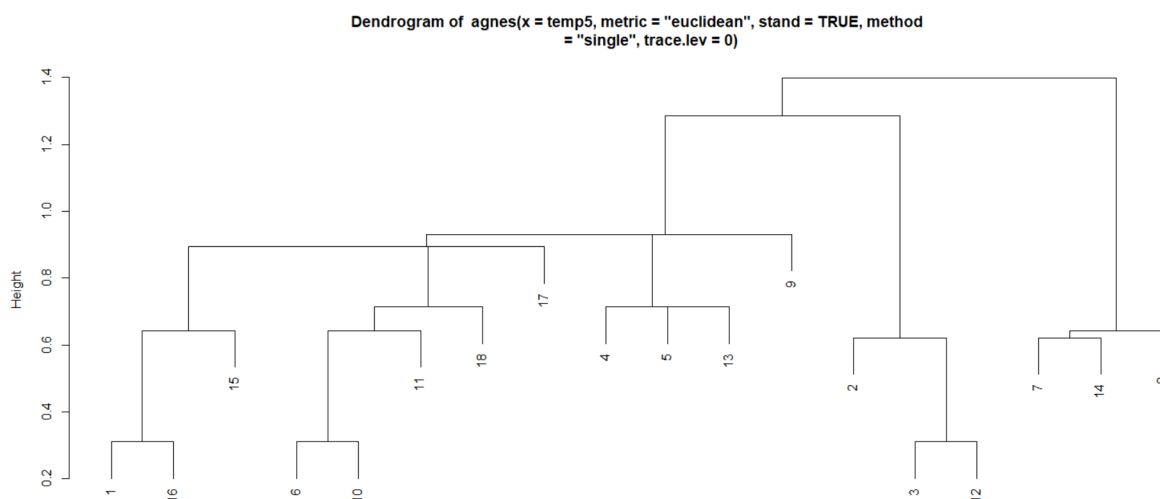


Figure 6.1: Hierarchical agglomerative clustering algorithm

As the *HAC* algorithm final result is one single cluster, it is necessary to establish some restrictions to guarantee the k -anonymity property. The main restriction will be that all clusters should contain between k and $2k - 1$ records.

The algorithm has been implemented in Python following these steps:

1. As in the *MDAV* algorithm, the first step is to standardize the data so that the data loss caused by microaggregation can be compared. If a variable v_i takes a value x , the standardization algorithm replaces it with $(x - \bar{v}_i) / s_{v_i}$, where \bar{v}_i is the average of the values taken by v_i and s_{v_i} is the standard deviation of these values.
2. The program builds a distance matrix between all the records in the dataset. These distances are measured using the Euclidean distance. However, other metrics can be used to measure the distances between two records.
3. When the distance matrix has been computed, the algorithm explores the matrix and then, if the union of the selected clusters do not contain more than $2k - 1$ records, it selects the clusters (or records) with the minimum distance.
4. The selected clusters are added to a new cluster and the distance matrix is updated following the *linkage criteria*. This criterion indicates the distance between the new cluster and the rest of the clusters in the distance matrix. There are several linkage criteria in literature (e.g. maximum or complete-linkage, minimum or single-linkage, average linkage, centroid linkage, etc.). In this case, the criterion used is called *minimum* or *single-linkage*.
5. Steps 3 and 4 are repeated until the algorithm can not create more new clusters because all the possible merge operations exceed the $2k - 1$ cluster size.
6. Once the loop has finished, if there are clusters smaller than k , the records that belong to these sets are merged with other clusters or form new clusters. Note that the resultant clusters size can not be less than k or greater than $2k - 1$.
7. Finally, the algorithm replaces each record by its centroid so that the k -anonymity property can be guaranteed.

The final result of the algorithm is a k -partition of the initial dataset.

After the algorithm has finished, the SSE (3.1) algorithm can be computed in order to check the homogeneity of the resulting k -partition.

6.2 Test

The test dataset of table 5.1 shows how to check the proper functioning of the algorithm. The dataset is composed of 19 records, each one with two numeric variables.

The results of executing the algorithm for the test dataset with $k = 4$ are:

- **SSE:** 20,56
- **Time:** 0 sec.

Figure 6.2 shows the result of executing the *HAC* algorithm for the test dataset. Each dot colour represents a group of records with size between k and $2k - 1$, so it can be observed that records have been classified into 3 distinct groups (1 of them with 7 elements and the other 2 with 6 elements each).

We can remark that this microaggregation is worse than the *MDAV* test. While with *MDAV* the SSE was 7,93, with *HAC* the SSE has been 20,56. Since the *HAC* algorithm produced only 3 clusters with more than k records, there is more information loss because the records are usually farther than the centroid of the group. The more clusters are produced, the less information is lost.

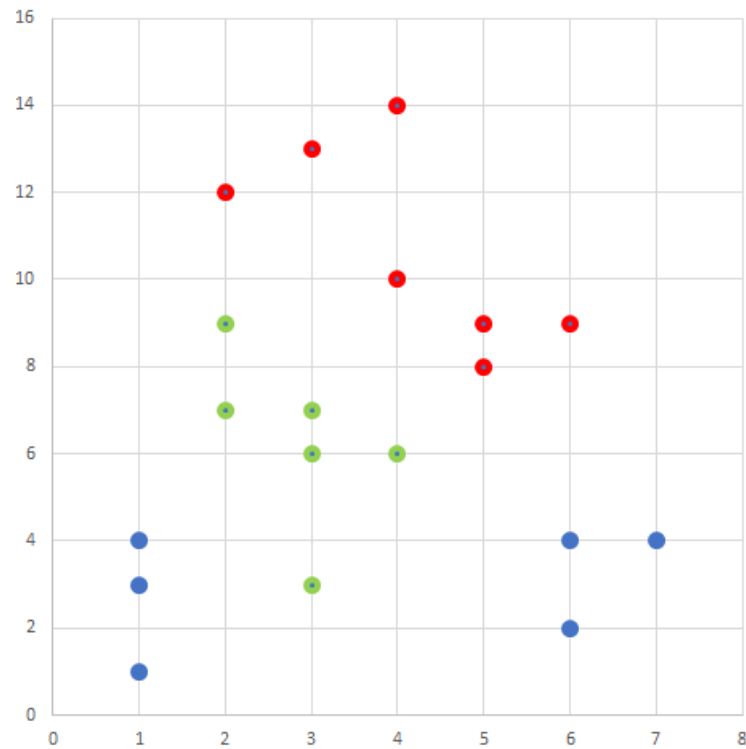


Figure 6.2: HAC test file k -partition

6.3 Evaluation

The Census dataset is selected with the purpose of evaluating the *HAC* algorithm. This dataset includes 13 numeric variables and 1080 records. The experiments were done for several k values (3, 4, 5 and 10). The results for each k value (including SSE and time in seconds) are shown in Table 6.1.

As it was stated in chapter 5, experiments reveal that lower k values obtain more homogeneous groups, which implies less information loss. Besides, higher k values obtain more heterogeneous groups, so there is more information loss.

k	SSE	Time (s.)
3	1284,95	192
4	1919,07	169
5	2154,53	175
10	3469,41	150

Table 6.1: HAC algorithm evaluation

HAC algorithm is significantly slower than *MDAV*. Note that, as in *MDAV*, *HAC* is faster for higher k values.

HAC improvements

This chapter presents several optimizations that can be applied to the *HAC* algorithm to increase the cluster homogeneity:

- The first optimization maximizes the number of clusters so their homogeneity increases.
- The next optimization evaluates different linkage criteria to test which criterion obtains more homogeneous clusters. This evaluation is done using the algorithm with the maximized number of clusters.
- The last optimization prioritizes the merging of the clusters whose sum of elements is near k . This last optimization includes the other two.

7.1 Maximization of the number of partitions

On the one hand, figure 6.2 shows that the 19 records included in the test dataset only formed 3 partitions when using the *HAC* algorithm. On the other hand, experiments reveal that smaller k values produce more homogeneous groups. Therefore, *HAC* algorithm can be modified to maximize the number of partitions created while the number of records of these partitions are not less than k .

The modified algorithm includes these steps:

1. The first step is to standardize the data. If a variable v_i takes a value x , the standardization algorithm replaces it by $(x - \bar{v}_i)/s_{v_i}$, where \bar{v}_i is the average of the values taken by v_i , and s_{v_i} is the standard deviation of these values.

2. Then, it builds a distance matrix between all the records in the dataset. These distances are measured using the Euclidean distance.
3. Once the distance matrix has been computed, the algorithm explores the matrix. Then, if the sum of the selected clusters do not contain more than k records, it selects the clusters (or records) with the minimum distance.
4. The selected clusters are added to a new cluster and the distance matrix is updated following the *linkage criterion*. In this case, the used criterion is *minimum* or *single-linkage*.
5. Steps 3 and 4 are repeated until the algorithm can not create more new clusters because all the possible merge operations exceed the k cluster size or the maximum number of clusters have been created. The maximum number of clusters is computed dividing the number of records in the dataset between k .
6. Once the loop has finished, there will probably be clusters with size lower than k . These clusters are merged with other clusters, prioritising those with size less than k .

The number of clusters with k size has to be maximum. Note that the size of the resultant clusters can not be lower than k or greater than $2k - 1$.

7. Finally, the algorithm replaces each record by its centroid so that the k -anonymity property is guaranteed.

Remark that this version of the *HAC* algorithm differs from the previous one (chapter 6) since it does not create clusters containing more than k records until the maximum number of clusters has been reached and the number of clusters with size equal to k is maximum.

The *Census* dataset was selected in order to compare this improvement with the original *HAC* algorithm. The experiments were also done for several k values (3, 4, 5 and 10). The results for each k value (including SSE and time in seconds) are shown in Table 7.1.

k	SSE	Time (s.)
3	1038,30	233
4	1363,40	210
5	1664,5	208
10	2379,23	146

Table 7.1: HAC maximum partitions algorithm evaluation

Table 7.1 shows a significant improvement as a result of the maximization of the number of partitions. For instance, for a k value equal to 4 the SSE is 1363,40 for the modified algorithm and 1919,07 for the original algorithm. For all the k values tested, the modified algorithm is significantly better than the original one. This is due to the fact that if the partitions have less records and these records are closer to the centroid, so the value of SSE is less than the *HAC* algorithm in chapter 6.

7.2 Linkage criteria evaluation

When two clusters are merged, linkage criteria establish the distance between the new cluster created and the rest of the clusters in the distance matrix. Therefore, the distance matrix needs to be updated with the distance from the new cluster to the rest of the clusters. Some of the most important linkage criteria are: maximum or complete-linkage, minimum or single-linkage, average linkage, centroid linkage, minimum energy clustering, etc.

In order to test the effectiveness in using a certain linkage criterion, some of them have been implemented and the obtained results have been analyzed. It is important to emphasize that these linkage criteria have been implemented for the *HAC* algorithm with the number of partitions maximized (section 7.1). The selected and implemented linkage criteria are:

- Minimum or single-linkage. When two clusters are merged, the algorithm updates the distance matrix with the minimum distance from either of the two merged clusters to each one in the matrix. Hence, every time the distance matrix is updated, the two clusters whose two closest members have the smallest distance are merged (the two clusters with the smallest minimum pairwise distance). The single-linkage criterion between 2 clusters A and B is:

$$\min\{d(a, b) : a \in A, b \in B\} \quad (7.1)$$

- Maximum or complete-linkage. When a merge operation between two clusters occurs, this criterion updates the distance matrix with the maximum distance from either of the two merged clusters to each one in the matrix. Hence, every time the distance matrix is updated, the two clusters whose two closest members have the smallest diameter are merged (the two clusters with the smallest maximum pairwise distance). The complete-linkage criterion between 2 clusters A and B is:

$$\max\{d(a, b) : a \in A, b \in B\} \quad (7.2)$$

- Centroid linkage. Due to a merge operation between two clusters, the distance matrix is updated with the distance from the centroid of the new created cluster to the centroid of the rest of clusters in the distance matrix. The centroid linkage criterion between two clusters A and B , where c_a is the centroid of A and c_b is the centroid of B , is:

$$\|c_a - c_b\| \quad (7.3)$$

Table 7.2 shows the comparison between the three implemented linkage criteria. This comparison is based on the results of executing the three algorithms for the *Census* dataset. It can be observed that single-linkage criterion obtains better SSE values than complete-linkage and centroid linkage. Centroid linkage obtains SSE values similar to single-linkage, but for each tested k size its SSE values are higher than single-linkage. Finally, complete-linkage SSE values obtained are far worse than centroid linkage and single-linkage SSE values.

Regarding the algorithm execution time, for the same k size complete-linkage execution times are slightly higher than single-linkage execution times. The centroid linkage execution times are extremely high because the centroid of each cluster has to be computed each time it is modified.

Therefore, as it obtains the lower SSE values, the single-linkage criterion produces more homogeneous clusters. In addition, this linkage criterion is also faster than the other implemented linkage criteria.

k	Single-linkage		Complete-linkage		Centroid linkage	
	SSE	Time (s.)	SSE	Time (s.)	SSE	Time (s.)
3	1038,3	233	1908,67	271	1703,3	20598
4	1363,4	210	2196,23	240	1679,7	19547
5	1664,5	208	3852,33	231	1979,91	19276
10	2379,23	146	4771,5	309	3112,41	20209

Table 7.2: HAC linkage criteria evaluation

7.3 Cluster creation priority

HAC algorithm creates and builds several clusters simultaneously. There are usually several uncomplete clusters and the algorithm decides in each iteration which clusters will be merged. An effective technique to improve the *HAC* algorithm could be to prioritize the completion of clusters. The prioritization of the cluster completion process has been developed by modifying step 3 of the *HAC* algorithm. Every time the distance matrix is updated, the 2 clusters selected for the next merge operation follow this criterion:

$$\min \left\{ \frac{d(a,b)}{|A|+|B|} : a \in A, b \in B \right\} \quad (7.4)$$

The distance (of the distance matrix) between a and b is divided by the sum of the sizes of the clusters A and B . The higher the sum of the sizes of the clusters A and B is, the lower the obtained value is. Therefore, the probability of selecting both clusters for the merge operation will be higher. Note that the sum of the sizes of the clusters A and B can not be higher than k to ensure the maximization of the number of clusters.

HAC algorithm with the number of partitions maximized with single-linkage criterion has been modified in order to implement and test this cluster creation criterion. Table 7.3 shows the result of this improvement using the *Census* dataset. It can be observed that for lower k cluster sizes, SSE values are better than the values obtained in section 7.1. As expected, for high k cluster sizes, this improvement is not decisive because it unduly penalises the creation of new clusters and the completion of clusters with few records. On the other hand, regarding the execution time, this criterion execution time is considerably higher because the algorithm has to compute the criterion for all values in the distance matrix, and then it has to select the minimum. Table 7.3 shows that the application of this improvement increases the execution time more than 10 times.

k	SSE	Time (s.)
3	877,39	5470
4	1338,52	3529
5	1624,83	4710
10	2802,90	4232

Table 7.3: Cluster completion priority evaluation

MDAV vs improved HAC

As part of this work consists of the adaption of a clustering algorithm such as *HAC* to be used for microaggregation, several improvements for *HAC* algorithm were done and tested in chapter 7. These improvements achieve cluster homogeneity values close to the *MDAV* algorithm, at least for small k values. On the other hand, the computation time needed by the improved *HAC* algorithm is considerably higher than the computation time needed by *MDAV*.

	MDAV		Improved HAC		
k	SSE	Time (s.)	SSE	Time (s.)	SSE diff.
3	798,44	39	877,39	5470	78,95
4	1051,28	32	1338,52	3529	287,24
5	1274,83	27	1624,83	4710	350,00
10	1985,65	19	2802,90	4232	817,25

Table 8.1: MDAV vs improved HAC

Table 8.1 shows the result of executing both algorithms with the *Census* dataset. There is an obvious relationship between the k size and the *SSE difference* (difference between the SSE values obtained by both algorithms).

For a k size of 3, the difference (column *SSE diff.*) between both SSE values is 78,95. This means that, even though *MDAV* is better, *HAC* algorithm obtains a similar cluster homogeneity, so the information loss will also be similar.

The difference between both SSE values rises up to 287,24 for a k size of 4. The value obtained by the *HAC* algorithm is similar to the value obtained by *MDAV*, but an increase of one unit in the k size also produced an increase in the *SSE difference* of about 3,5 times.

For a k size equal to 5, the SSE value obtained by the *HAC* algorithm is still similar to the value obtained by *MDAV*. The difference between both SSE values is 350, which is about 60 units higher than it is for a k size equal to 4.

Finally, for a k size equal to 10, the *SSE difference* between both algorithms is 817,25. The duplication of the k cluster size produced an increase of the *SSE difference* of about 2,5 times.

In conclusion, the experiments reveal that both algorithms obtain similar cluster homogeneity values for small k sizes. However, the obtained clusters when increasing k are not as homogeneous as they are for lower k values.

Regarding the execution time, table 8.1 shows that *HAC* algorithm is more than 100 times slower than *MDAV* algorithm for all k cluster sizes. However, table 7.1 shows that *HAC* algorithm with the number of partitions maximized has a computation time higher than *MDAV*. However it is more than 10 times lower than the improved version (which includes cluster creation priority) of the *HAC* algorithm.

Conclusions

As the amount of data collections containing person-specific information is growing exponentially, and a great variety of devices collect and store information about us, it is necessary to guarantee our privacy protection.

The protection model named k -anonymity ensures that each person contained in a released data collection can not be distinguished from at least $k-1$ people also appearing in that same collection.

Typical clustering algorithms are *MD*, *MDAV*, *V-MDAV*, *K-means*, *TFRP*, etc. They can be used in microaggregation to ensure the k -anonymity property of data collections. This work has evaluated the cluster homogeneity obtained by *MDAV* algorithm with the *Census* dataset. In addition, a clustering algorithm, called hierarchical agglomerative clustering (*HAC*) has been implemented and improved in order to obtain satisfactory results.

HAC algorithm starts considering each record of the dataset as one cluster and then, it merges pairs of clusters until they become one single cluster. Therefore, it has been modified to stop at a maximum cluster size, because cluster size can not be greater than $2k - 1$. Once implemented, the algorithm has been evaluated and compared to *MDAV*. This comparison reflects the need to improve the standard *HAC* algorithm in order to obtain clusters with high homogeneity. This is due to the fact that the more cluster homogeneity is achieved, the less data loss is produced.

The improvements made to the *HAC* clustering algorithm were:

- Maximization of the number of clusters produced by the algorithm. This implies that the number of clusters of k size has to be maximum.
- Evaluation of different linkage criteria in order to determine which criterion obtains better SSE values for the *Census* dataset.

- Assignment of priorities to the cluster creation process. This priority encourages the creation of clusters with size near to k instead of creating new clusters.

The improved *HAC* algorithm obtained good SSE values, similar to *MDAV* for small k sizes. However, for greater k cluster size values, the third improvement (cluster creation priority) obtained worse results than if this improvement was not applied. Regarding the execution time, the improved *HAC* algorithm takes a long time compared to *MDAV*. In addition, it is also significantly slower than the *HAC* algorithm without the third improvement.

Possible future work could be the reduction of the execution time of the improved *HAC* algorithm. This could be achieved by storing some frequent operations, as the centroid computation, in data structures to avoid computing them in each iteration. Furthermore, the third improvement (cluster creation priority) should be studied in order to improve the results for large k cluster sizes. This could be accomplished by ponderating the priorities for different k cluster sizes. Experiments could be designed in order to obtain the best ponderation values for different k sizes.

Bibliography

- [1] Chin-Chen Chang, Yu-Chiang Li, and Wen-Hung Huang. Tfrp: An efficient microaggregation algorithm for statistical disclosure control. *J. Syst. Softw.*, 80(11):1866–1878, November 2007.
- [2] R Lopez de Mantaras and L Saitia. Comparing conceptual, divisive and agglomerative clustering for learning taxonomies from text. In *ECAI 2004: 16th European Conference on Artificial Intelligence, August 22-27, 2004, Valencia, Spain: Including Prestigious Applicants [sic] of Intelligent Systems (PAIS 2004): Proceedings*, volume 110, page 435. IOS Press, 2004.
- [3] D. Defays and P. Nanopoulos. Panels of enterprises and confidentiality: The small aggregates method. pages 195–204, 1992.
- [4] J. Domingo-Ferrer and J. M. Mateo-Sanz. Practical data-oriented microaggregation for statistical disclosure control. *IEEE Trans. on Knowl. and Data Eng.*, 14(1):189–201, January 2002.
- [5] Josep Domingo-Ferrer and Vicenç Torra. Ordinal, continuous and heterogeneous k-anonymity through microaggregation. *Data Mining and Knowledge Discovery*, 11(2):195–212, Sep 2005.
- [6] Josep Domingo-Ferrer and Vicenç Torra. On the connections between statistical disclosure control for microdata and some artificial intelligence tools. *Inf. Sci. Inf. Comput. Sci.*, 151:153–170, May 2003.
- [7] A. W. F. Edwards and L. L. Cavalli-Sforza. A method for cluster analysis. *Biometrics*, 21(2):362–375, 1965.
- [8] E. C. for Europe. Statistical data confidentiality in the transition countries: 2000/2001 winter survey. *Joint ECE/Eurostat Work Session on Statistical Data Confidentiality*, (invited paper n.43), 2001.
- [9] A. D. Gordon and J. T. Henderson. An algorithm for euclidean sum of squares classification. *Biometrics*, 33(2):355–362, 1977.
- [10] Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.
- [11] Md. Enamul Kabir, Abdun Naser Mahmood, and Abdul K. Mustafa. K-means clustering microaggregation for statistical disclosure control. In Aswatha Kumar M., Selvarani R., and T V Suresh Kumar, editors, *Proceedings of International Conference on Advances in Computing*, pages 1109–1115, New Delhi, 2012. Springer India.
- [12] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.
- [13] Guillermo Navarro-Arribas and Vicenç Torra. Privacy-preserving data-mining through micro-aggregation for web-based e-commerce. *Internet Research*, 20:366–384, 01/2010 2010.

-
- [14] M. Rosemann. Erste ergebnisse von vergleichenden untersuchungen mit anonymisierten und nicht anonymisierten einzeldaten amb beispiel der kostenstrukturerhebung und der umsatzsteuerstatistik. *G. Ronning y R. Gnos (editores) Anonymisierung wirtschaftsstatistischer Einzeldaten, Wiesbaden: Statistisches Bundesamt*, pages 154–183, 2003.
- [15] Agustí Solanas and Antoni Ballesté. V-mdav: Variable group size multivariate microaggregation. *COMP-STAT 2006*, pages 917–925, 2006.
- [16] Agustí Solanas, Antoni Martínez-Ballesté, Josep Domingo-Ferrer, Susana Bujalance, and Josep M. Mateo-Sanz. Métodos de microagregación para k-anonimato: privacidad en bases de datos. *Dept. Enginyeria Informàtica i Matemàtiques, Universitat Rovira i Virgili*.
- [17] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002.

Julián Alarte Aleixandre
Barcelona, 2018