U. PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

MSc

MSc

2.º
CICLO

FCUP
2020/2021

U. PORTO

Prediction of Privacy Preferences with User Profiles:
A Federated Learning Approach

André Xavier Ribeiro
de Almeida Brandão

FC

U. PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Prediction of Privacy Preferences with User Profiles: A Federated Learning Approach

André Xavier Ribeiro de Almeida Brandão

Mestrado em Ciência de Dados
2020/2021

Orientador
João P. Vilela, Professor auxiliar, Faculdade de Ciências da Universidade do Porto

Supervisor
Ricardo Mendes, Investigador, Faculdade de Ciências e Tecnologias da Universidade de Coimbra

MSc

Abstract

Smartphones prevalence in today's society comes from the multitude of extremely useful services provided. However, this depends on concerning access to amounts of information, such as the one obtained from body sensors, calendars events, contacts, location, phone messages, internal and external storage, and many more. Smartphones' privacy managers allow the user to control the application's access to this sensitive data, however, the current permission systems often misrepresent human intentions, since most of the times the permission decision (granted or denied) remains the same after the first time it is asked to the user. Nonetheless, the decision of the user can change depending on the context of the requesting application (app) and the context of the user. For example, a user might allow a permission for an application that he is currently using, but deny the same permission if the app is currently running in the background.

In this thesis we develop a secure strategy to learn and predict the users' privacy preferences according to the current context, such as if the app is in foreground or not, and the location of the user. Our strategy is secure in the sense that the users' data remains private and it is not shared with any entity, including the server responsible for the learning process. We propose an efficient and secure distributed k-Means algorithm, that is robust to data that is not independent and identically distributed (IID). The base idea of our proposal consists in each client computing the k-Means algorithm locally, with a variable number of clusters. The server will use the resultant centroids to apply the k-Means algorithm again, discovering the global centroids. To maintain the client's privacy, homomorphic encryption and secure aggregation is used in the process of learning the global centroids, such that the server has no access to the original data, except its cyphertext. This algorithm is efficient and reduces transmission costs, since only the local centroids are sent to the server to find the global centroids.

Using our proposed strategy we can generate privacy profiles from the resulting clusters in a secure way. Then, using federated learning we can train a neural network to predict the users' grant decisions (accept or deny a permission) using the context variables and the generated privacy profiles as features, while guaranteeing the user privacy. To evaluate our strategy, we resort to a dataset of privacy decisions collected from a field study with 93 participants, whereby user's grant decisions are securely predicted using federated learning neural networks. Our secure strategy leads to an F1-score of 90%, thus matching the state-of-the-art strategy for privacy decisions prediction, with the advantage of our approach providing privacy guarantees.

Resumo

A prevalência de *smartphones* na sociedade atual provém da multiplicidade de serviços extremamente úteis fornecidos pelos mesmos. No entanto, isso depende do acesso a uma quantidade de informação preocupante, como a de sensores corporais, eventos de calendários, contatos, localização, mensagens, armazenamento interno e externo e muito mais. Os gestores de privacidade dos *smartphones* permitem que o utilizador controle o acesso das aplicações a estes dados sensíveis, no entanto, os sistemas de gestão das permissões atuais muitas vezes não representam as intenções humanas, já que na maioria das vezes a decisão de permissão (concedida ou negada) permanece a mesma após a primeira vez que é pedida ao utilizador. No entanto, a decisão do utilizador pode mudar dependendo do contexto da aplicação (app) e do utilizador. Por exemplo, um utilizador pode permitir uma permissão para uma aplicação que está a usar no momento, mas negar a mesma permissão se a aplicação estiver a ser executada em segundo plano.

Nesta tese, desenvolvemos uma estratégia segura para aprender e prever as preferências de privacidade dos utilizadores de acordo com o contexto, como a aplicação estar em *foreground* ou não, e a localização do utilizador. A nossa estratégia é segura no sentido em que os dados dos utilizadores permanecem privados e não são partilhados com nenhuma entidade, incluindo o servidor responsável pelo processo de aprendizagem. Propusemos uma maneira eficiente e segura de aplicar o algoritmo k-Means de forma distribuída, que é robusta em casos extremos de dados não distribuídos de forma idêntica e independente. A ideia base da nossa proposta consiste em cada cliente executar o algoritmo k-Means localmente, com um número variável de clusters. O servidor usará os centróides resultantes para aplicar o algoritmo k-Means novamente, descobrindo os centróides globais. Para manter a privacidade do cliente, encriptação homomórfica e agregação segura são usadas no processo de aprendizagem dos centróides globais, de forma a que o servidor não tenha acesso aos dados originais, apenas aos dados cifrados. Este algoritmo é eficiente e reduz os custos de transmissão, uma vez que apenas os centróides locais são usados para encontrar os centróides globais.

Usando o algoritmo proposto geramos perfis de privacidade de forma segura. De seguida, usando *federated learning*, podemos treinar uma rede neuronal para prever as decisões de permissões dos utilizadores (concedem ou negam uma permissão) usando as variáveis de contexto e os perfis de privacidade como *features*, garantindo a privacidade dos utilizadores. Para avaliar a nossa estratégia, tiramos partido de um *dataset* de decisões de privacidade recolhidas a partir de um estudo de campo com 93 participantes, em que as decisões de permissão do utilizador são previstas com segurança usando redes neuronais usando *federated learning.* A nossa estratégia alcança um F1-Score de 90%, correspondendo assim à performance da estratégia do estado da arte para previsão de permissões de privacidade, com a vantagem de a nossa abordagem dar garantias de privacidade.

Acknowledgements

I wish to express my gratitude to my supervisors, Professor João Vilela and Ricardo Mendes, for all the support and knowledge they gave me to develop this dissertation, as well the publication of the paper published in IDA 2021. I also would like to thank UP Cybersecurity and Privacy Centre (C3P) for providing me a place to work at their laboratory. Finally, I would like to thank my family for their support.

The work presented in this thesis was carried out in the scope of project COP-MODE, that has received funding from the European Union's Horizon 2020 research and innovation programme under the NGI TRUST grant agreement no 825618, the project Project SNOB-5G with Nr. 045929 (CENTRO-01-0247-FEDER-045929) supported by the European Regional Development Fund (FEDER), through the Regional Operational Programme of Centre (CENTRO 2020) of the Portugal 2020 framework and FCT under the MIT Portugal Program, and the project AIDA: Adaptive, Intelligent and Distributed Assurance Platform (POCI-01-0247-FEDER-045907), co-financed by the European Regional Development Fund through the COMPETE2020 program and by the Portuguese Foundation for Science and Technology (FCT) under the CMU Portugal Program.

Contents

Chapter 1

Introduction

Smartphones are mobile devices that gather almost the same capabilities as personal computers. Besides the traditional voice calls and text messages, they contain multimedia functionalities like music, image, video or gaming. They incorporate a variety of sensors like magnetometer, gyroscope, accelerometer or proximity sensors and support communication protocols such as Wi-Fi, Bluetooth or satellite navigation, which allows the collection of a variety of data [1]. Current modern smartphones are able to collect vast amounts of information like location, photos, messages, call log, contacts or emails. On top of this, is possible to extract high level information from this data, like home address, work address or close friends. Due to the widespread adoption of these devices, and the ease of developing applications for smartphones, the barrier to collect personal information from the masses is now much lower. This led to privacy concerns, specially after events where the data collected was used to malicious purposes, like the Cambridge Analytica scandal[1].

All this data collection is done by the apps installed on the users' smartphone. So, the users can reduce the amount of data shared with the applications by allowing or denying specific permissions for each application, such as location, camera, contacts, microphone, storage or call logs. To allow for user control over these permissions, smartphones implement two permission systems: Ask-On-Install (AOI) and Ask-On-First-Use (AOFU), the first asks the user to define the permissions when the application is installed and the latter when it is first used. The user can always change the permissions configuration on the phone settings latter on in the AOFU strategy.

This thesis is incorporated in the COP-MODE project[2]. Context-Aware Privacy Protection for Mobile Devices (COP-MODE) is a research project led by the University of Coimbra, the University of Porto and the University of Cambridge aiming at enhancing the privacy of mobile devices, with privacy-preserving mechanisms that empower users with control over their data.

[1]https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election (2021, August 17)

[2]https://cop-mode.dei.uc.pt (2021, August 17)

In the scope of this project, several data gathering campaigns were performed in order to gather the necessary data to enable automated privacy protection. This thesis targets performing such automated privacy protection in a private manner, i.e. without access to user data.

1.1 Problem Definition

There are problems with both AOI and AOFU strategies. In AOI prompts, people do not pay attention or cannot understand the prompts. The users' answers are also static, *i.e.* they will remain with the initial configuration until the user decides to change them on the phone's settings. These static permissions do not account for the user's context, like user's location, if the application asking the permission is being used or not or the time of the day [2–5].

In Ask-on-first-use system, every time an application asks for a permission for the first time, the user has to choose to allow or deny it. This decision will be based on that specific context, which makes it better than the AOI system [6, 7]. However, after being allowed a permission once, the app will have this permission automatically granted for all subsequent uses, even without the user noticing. Therefore, AOFU does not account for the context when automatically granting permissions.

Taking into consideration what was said above, there is still a gap in the research on what influences users' decisions in terms of app permissions. The current model (AOFU) does not represent how the users think about privacy. According to the data collected by our campaign, each user has on average about 35 permission requests per hour, which makes manual answering unfeasible. So, there is a need to create tools to automate these decisions, that are capable to represent the user intentions as a function of the context of the user and of the context of the device.

Currently, there are systems capable of automating these decisions. However, some of these mechanisms are trained locally, which require intensive users' input, which leads to user fatigue, and others rely on a centralized approach, where all the users' privacy preferences data must be sent to a central server.

Our goal is to have a system capable of learning the users' privacy preferences according to the context, and the learning mechanism must to be private, *i.e.* the users' data about their privacy preferences and context should not be disclosed to anyone.

1.2 Goals and Contributions

The main goal of this thesis is to develop a secure strategy to learn and predict the user's privacy preferences according to different contexts, while protecting the users' privacy.

The contributions of this thesis are as follows:

- We propose an efficient and secure way to apply the K-Means algorithm in a distributed fashion, that is robust to extreme cases of non-IID, where each user belongs to only one or two clusters, which is the case in mobile privacy preferences' profiles.
- Secure privacy profile generation: Using the algorithm developed in the previous item, we are able to generate privacy profiles in a secure way, without sharing user's personal data.
- Privacy profiles' usefulness testing: In this thesis we also show how we can compute the usefulness of the generated privacy profiles in a secure way, using federated learning.
- Security complements to the Privacy Preserving Distributed Hierarchical Clustering (PPDHC) algorithm. PPDHC is a general framework for constructing a private agglomerative hierarchical clustering algorithm over partitioned data, based on secure scalar product and secure hamming distance. However the implementation presented by the authors contained three security problems related to data leakage. We provided a solution to two of those problems.

As a result, from these contributions, the following paper was published and presented at the IDA 19th Symposium on Intelligent Data Analysis, in Apr 2021: A. Brandão, R. Mendes, J. P. Vilela, "Efficient Privacy Preserving Distributed K-Means for Non-IID Data", International Symposium on Intelligent Data Analysis, Porto, Portugal, April 2021.

1.3 Structure

This dissertation is divided in the following six chapters:

Chapter 1 - Introduction: Brief description of the problem, its context and motivation. We defined the goals of this dissertation as well its structure.

Chapter 2 - Background: Explanation of core concepts necessary for the understanding of this dissertation, such as machine learning and user profiling.

Chapter 3 - State-of-the-Art: Description of the state-of-the-art approaches to the problems we will face in this dissertation.

Chapter 4 - Efficient Privacy Preserving Distributed k-Means for Non-IID Data: Description of a new clustering technique for distributed non-IID data, that is also efficient and private.

Chapter 5 - Secure Generation of Privacy Profiles: Real World Application: We present our approach to solve a real world problem, using the strategies described in the previous chapters, of generating privacy profiles in a secure way.

Chapter 6 - Conclusion and Future Work: We summarize our findings, together with some reflections on the results obtained, together with some guidelines for future work.

Chapter 2

Background

In this chapter, we provide the necessary background information for. In particular, we start by describing the Cross Industry Standard Process for Data Mining (CRISP-DM) in section 2.1, an open standard describing how to approach a data mining problem. This is the guideline we used in this thesis.

In section 2.2 we explain what supervised learning (section 2.2.1) and unsupervised learning (section 2.2.2) is, and common models used in both techniques. The unsupervised learning algorithms will be used to generate privacy profiles and the supervised learning algorithms to evaluate their usefulness.

Finally, in section 2.3 we describe three strategies for user profiling, neighbourhood based techniques (section 2.3.1), machine learning techniques (section 2.3.2), and ontology based techniques (section 2.3.3). This techniques make use of unsupervised learning algorithms and other strategies such as collaborative filtering to generate user profiles. Our goal is to generate privacy profiles using mobile privacy preferences data, so we apply some of this strategies to accomplish that goal.

2.1 CRISP-DM

Cross-Industry Standard Process for Data Mining (CRISP-DM), conceived in late 1999, is a process model that specifies the data mining life cycle (figure 2.1). This model describes a systematic and flexible approach to tackle data mining problems.

This model is composed by six phases:

- **Business Understanding**: In this first phase, the focus is on understanding the goals and requirements of the project from a business perspective. Only then, we can convert it to a data mining problem and formulate plan to achieve the desired goals.
- **Data Understanding**: This phase involves the data collection and continues with actions

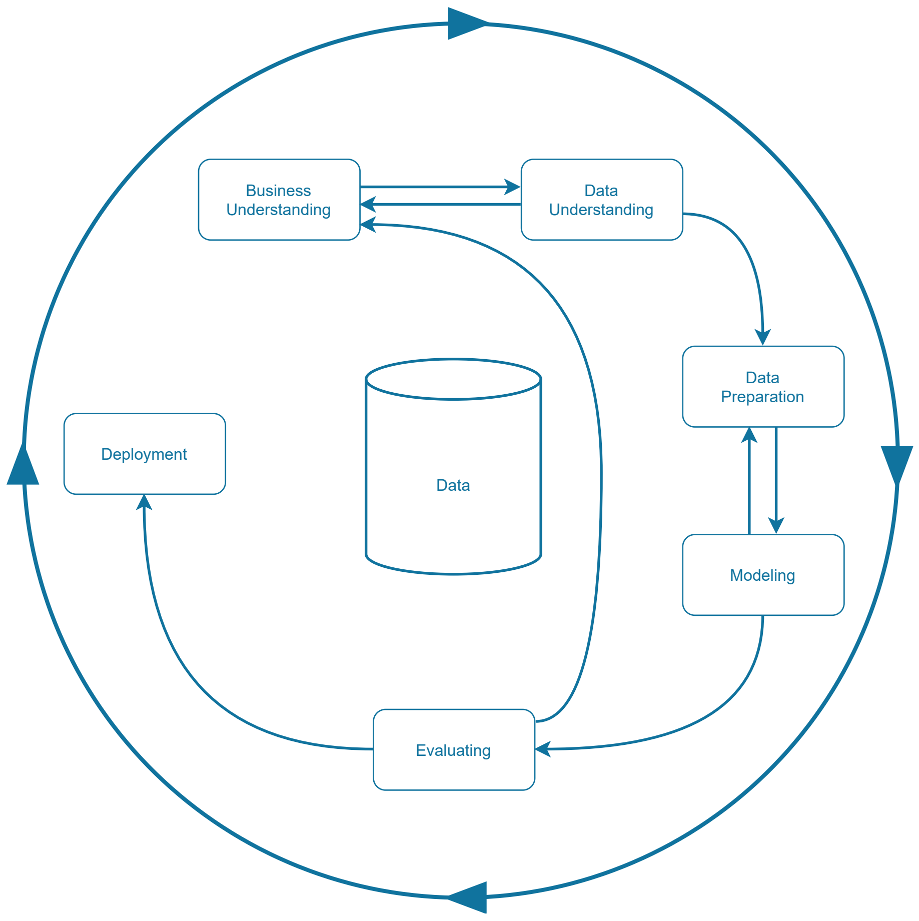


Figure 2.1: Phases of the CRISP-DM model.

that allows one to became familiar with data. Evaluate the quality of the data, acquire the first insights and formulate possible hypothesis.

- **Data Preparation**: After the data collection, the raw dataset needs to be cleaned and transformed in order to be ready to be fed into the model(s). It is normal to perform this phase multiple times throughout the life cycle.
- **Modeling**: In this phase the data prepared in the previous phase is going to be used to train model(s) and fine-tune the respective parameters.
- **Evaluation**: At this phase of the life cycle, we have a final model(s) trained with the collected data. Now, before deploying the model we need to verify it it achieves the business goals defined in the first phase.
- **Deployment**: In this final phase we already have the model(s), fully validated and ready to use. Depending on the objectives, this phase could consist in a simple report or in a complex setup requiring continuous learning.

2.2 Machine Learning

Although the definition of machine learning varies in the literature and there is terminological confusion, one of the most famous definitions is by Thomas Mitchell, which in 1997 described machine learning as follows [8]:

Definition 2.2.1 (Machine Learning) *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

This definition is very broad and encompasses a large variety of problems, Shai *et. al* [9] provide a thorough taxonomy of learning paradigms. Here we would like to detail the differences between supervised and unsupervised learning and describe some algorithms from both learning strategies.

2.2.1 Supervised Learning

In supervised learning algorithms, the goal is to learn a function that maps an input to an output. In the training phase, the algorithm has access to examples where both input and correct output are present. Hence the name "supervised", because the learning is being supervised and the algorithm knows when the answer is right or wrong. The following sections detail supervised learning techniques that are relevant in the context of this work.

2.2.1.1 Linear Regression

Linear regression with one explanatory variable is one of the simplest supervised learning models. This model assumes a linear relationship between the input X and the output (equation 2.1).

$$h(X, \theta) = X\theta + b \tag{2.1}$$

The parameters θ and b are going to be estimated from the data using the least squares method. The least squares method finds the optimal θ values by minimizing the Residual Sum of Squares (RSS) (equation 2.2).

$$RSS(\theta) = \sum_{i=1}^{n}(y_i - h(x_i, \theta))^2 \tag{2.2}$$

To find the minimum value of the RSS function, the gradient descent is used (algorithm 1).

The weights θ are randomly initialized, then the gradient of the cost function (RSS) is computed. The gradient gives the direction to which the cost function is increasing, so the

Algorithm 1 Gradient Descent

```
1: $\theta_0 \leftarrow$ random initialization
2: while $\theta_t - \theta_{t-1} > \epsilon$ do
3:     Grad $\leftarrow \nabla * RSS(\theta_t)$
4:     $\theta_{t+1} \leftarrow \theta_t - \lambda$ Grad
5: end while
```

gradient multiplied by a learning rate λ is subtracted to the current weights θ.

In figure 2.2 we can observe a linear regression applied to an artificial dataset with two variables X and Y, where the goal is to predict Y based on X.

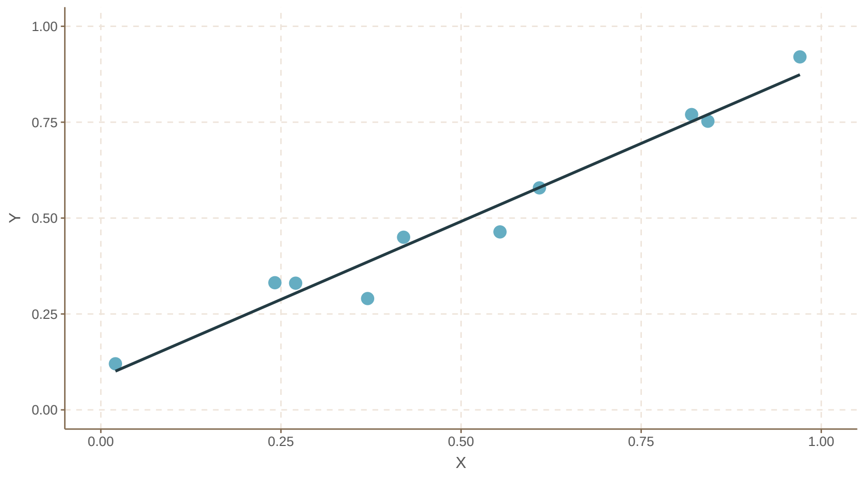


Figure 2.2: Example of a linear regression.

The final model, *i.e.* the black line represented in figure 2.2 has the following equation (2.3).

$$Y = X * 0.81359 + 0.08441 \tag{2.3}$$

With a Residual Sum of Squares of 0.02114558.

2.2.1.2 k-Nearest Neighbours

The k-Nearest Neighbour (k-NN) classification algorithm is a non-parametric machine learning method, where the output of the model is the class to which the input object belongs to. To predict the class of a new object, the most common class among its k neighbours is outputted (equation 2.4).

$$h(x) = \text{argmax}_{C_j} \#\{x_i \in N(x,k) | y_i = C_j\} \tag{2.4}$$

Where $N(x,k)$ is the set of k neighbours of x and C_j is the class j. We want to find the class that contains the most neighbours. This algorithm is a type of lazy learning method, since the function is locally approximated and all the computations are delayed until the function evaluation. In figure 2.3 we can observe the result of the k-NN applied to an artificial dataset with two continuous variables: X and Y and one binary variable, represented by the color: blue or black.

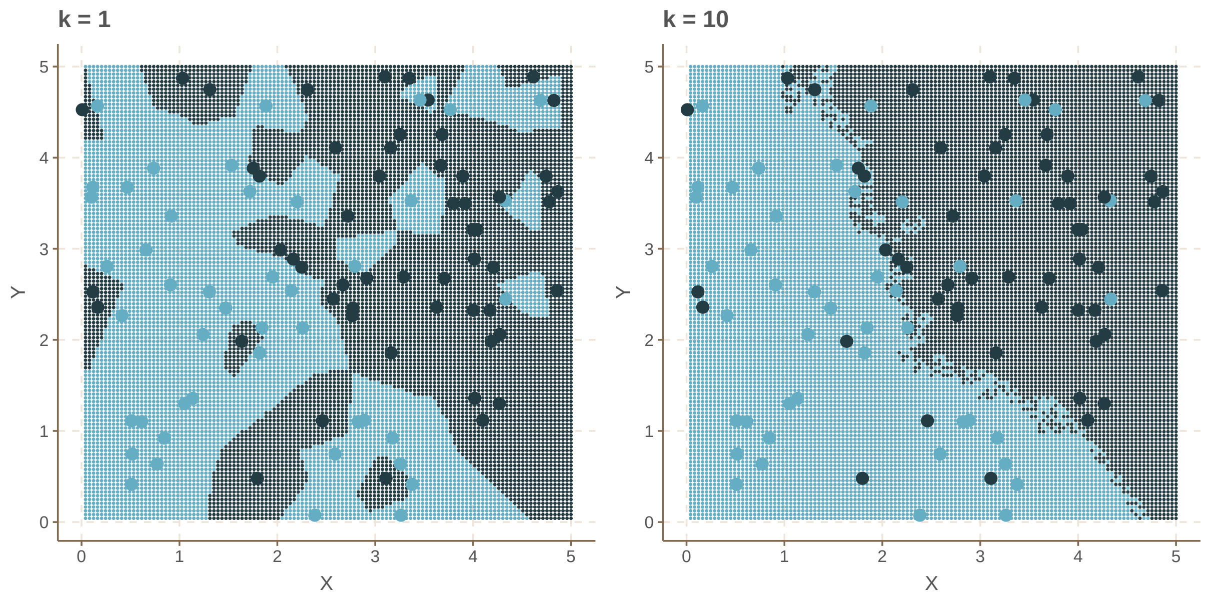


Figure 2.3: k-Nearest Neighbours example with artificial dataset for $k = 1$ and $k = 10$.

On the left side of figure 2.3 it is represented the k-NN model for k equal to 1. We can observe that the boundary is very complex and it correctly describes almost every point. However, this model is overfitted to the training data, this means that the model will fail to correctly predict new observations. On the right side, we observe the resulting model for k equal to 10. In this case the boundary is smoother, although the majority of observations are well categorized, there are around 14 who are not. In conclusion, by tweaking the parameter k we can manipulate the complexity of the model, increasing or decreasing the bias.

2.2.1.3 Neural Networks

Neural networks are models inspired by biological neural networks present in animal brains. The simplest neural network configuration is a perceptron (figure 2.4).

The output function of this model is 0 or 1 according to a threshold ϵ defined in the step function (equation 2.5).

$$h(x) = \begin{cases} 0 & \text{if } \sum_{i=0}^{n} \theta_i x_i < \epsilon \\ 1 & \text{if } \sum_{i=0}^{n} \theta_i x_i \geq \epsilon \end{cases} \tag{2.5}$$

This model represents a linear combination of the inputs with a discretized ouput, achieved

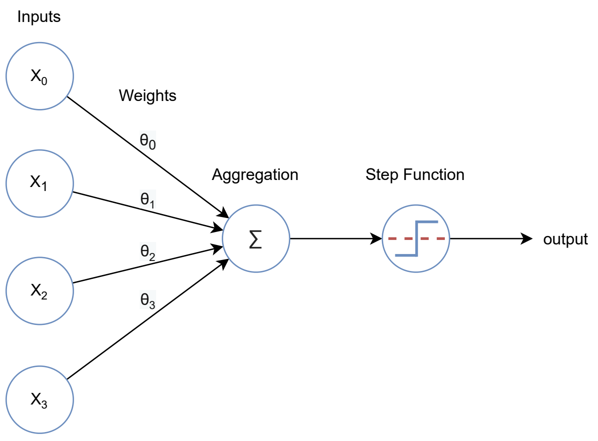


Figure 2.4: Perceptron diagram.

using the activation function **step function**. In order to model more complex, non-linear functions, we need to add more perceptrons, a multi-layer perceptron or neural network (figure 2.5).

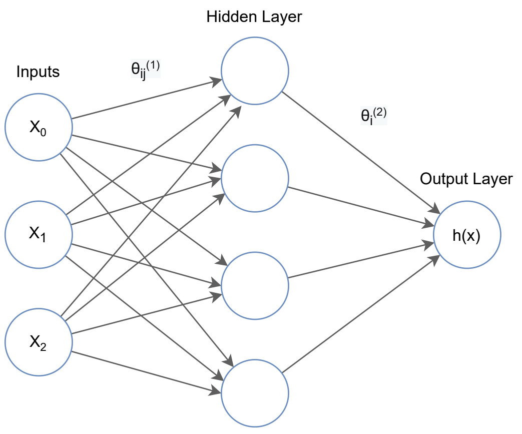


Figure 2.5: Multi-layer perceptron or Neural Network diagram.

Each perceptron in the hidden layer will serve as an input to the perceptrons in the output layer, resulting in the following output (equation 2.6).

$$h(x) = f^{(2)}(\sum_j f^{(1)}(\sum_i x_i\theta_{ij}^{(1)}) \cdot \theta_j^{(2)}) \tag{2.6}$$

where $f^{(i)}$ corresponds to the activation function of the neurons in layer i. This process of computing the output by computing the perceptron values from the left to right is called **feedforward**. The goal of a neural network is to find the weights that better fit the training data. Since there is no analytical solution, in general, we use the same search procedure used in the linear regression, the gradient descent. In the case of the neural networks, we call this process **backpropagation**.

2.2.2 Unsupervised Learning

As opposed to supervised learning, unsupervised learning does not have access to the output, only the input. Since the algorithm does not have an explicit feedback, the goal of the algorithm is to learn patterns in the input data [10]. One of the most common unsupervised learning tasks is clustering: grouping objects in groups (clusters) such that the objects in the same cluster are more similar between each other than to the ones in other clusters.

2.2.2.1 K-Means Clustering

K-Means, proposed by Stuart Lloyd in 1957 but only published in 1982 [11], is one of the most known and widely used clustering algorithms. It is simple to implement, easily adapts to new examples and it is able to create clusters of different shapes and sizes. However it is dependent on the initial centroids and does not scale with the number of dimensions, given the curse of dimensionality. In high dimensional data, all objects appear to be sparse and dissimilar in many ways and the euclidean distance loses statistical significance.

The goal of K-Means is to assign each observation to a single cluster minimizing the within-cluster Euclidean distance:

$$C_1^* \dots C_k^* = \underset{C_1 \dots C_k}{\operatorname{argmin}} \sum_{k \in K} \sum_{x_i \in C_k} ||x_i - c_k||^2 \tag{2.7}$$

where K is the set of cluster IDs, C_k is a set of points representing a cluster and c_k is the centroid value for cluster k, the mean point of that cluster k. The C_i^* are the resulting clusters.

The algorithm works as follows: given a set of k initial centroids $c_1^{(1)}, c_2^{(1)}, \dots, c_k^{(1)}$, the algorithm iteratively executes the following two steps:

1. **Assignment**: Assign each observation to the cluster with the nearest mean, with respect to the Euclidean distance.
2. **Update**: Compute the new centroids' values.

$$c_i^{(t+1)} = \frac{1}{|C_i^{(t)}|} \sum_{x_p \in C_i^{(t)}} x_p \tag{2.8}$$

The algorithm repeats these steps until the shift between the new centroids and the previous ones is lower than a specified threshold $||c^{(t+1)} - c^{(t)}|| \leq \epsilon$. In figure 2.6 we can visualize an example of the iterations done by the K-Means algorithm on the Iris dataset [12]. It is possible to observe that the initial centroids started all inside the same cluster, the *versicolor* species and iteratively spread away towards the other clusters.

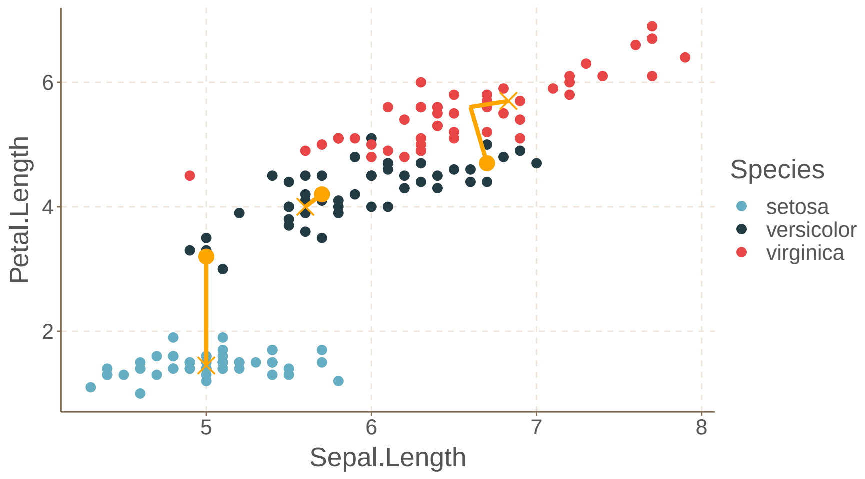


Figure 2.6: K-Means iterations visualization. The initial centroids are represented by the orange points, then at each iteration they change position, until they converge to their final centroids, represented by the orange crosses.

2.2.2.2 Agglomerative Hierarchical Clustering

Agglomerative Hierarchical Clustering is an algorithm for cluster analysis that builds a hierarchy of clusters. Unlike other cluster algorithms, Hierarchical Clustering only requires a measure of similarity between groups of data points. This similarity distance is called linkage criterion, some commonly used linkage criteria between two groups of observations A and B are:

- **Single-Linkage**: $\min\{d(a,b) : a \in A, b \in B\}$.
- **Complete-Linkage**: $\max\{d(a,b) : a \in A, b \in B\}$.
- **Centroid Linkage**: $||C_A - C_B||$, where C_i is the centroid of the group of observations i.

The algorithm is simple in terms of complexity, however it has a time complexity of $O(n^3)$ and requires $O(n^2)$ memory:

1. Create a group for every point in the dataset.
2. **Repeat**: iteratively merge the groups that are closer to each other, according to the similarity distance.
3. **Until**: All the data points belong to the same group.

In figure 2.7 we can observe every step in the hierarchical clustering algorithm (complete-linkage) applied to an artificial dataset with two variables. In the first iteration we see the points 5 and 6 are merged into the same group, represented by the red line. In the second iteration, the groups corresponding to the points 1 and 2 are the closest ones, so they are merged together. In the next iterations the process is repeated until in iteration 6 all points belong to the same group.

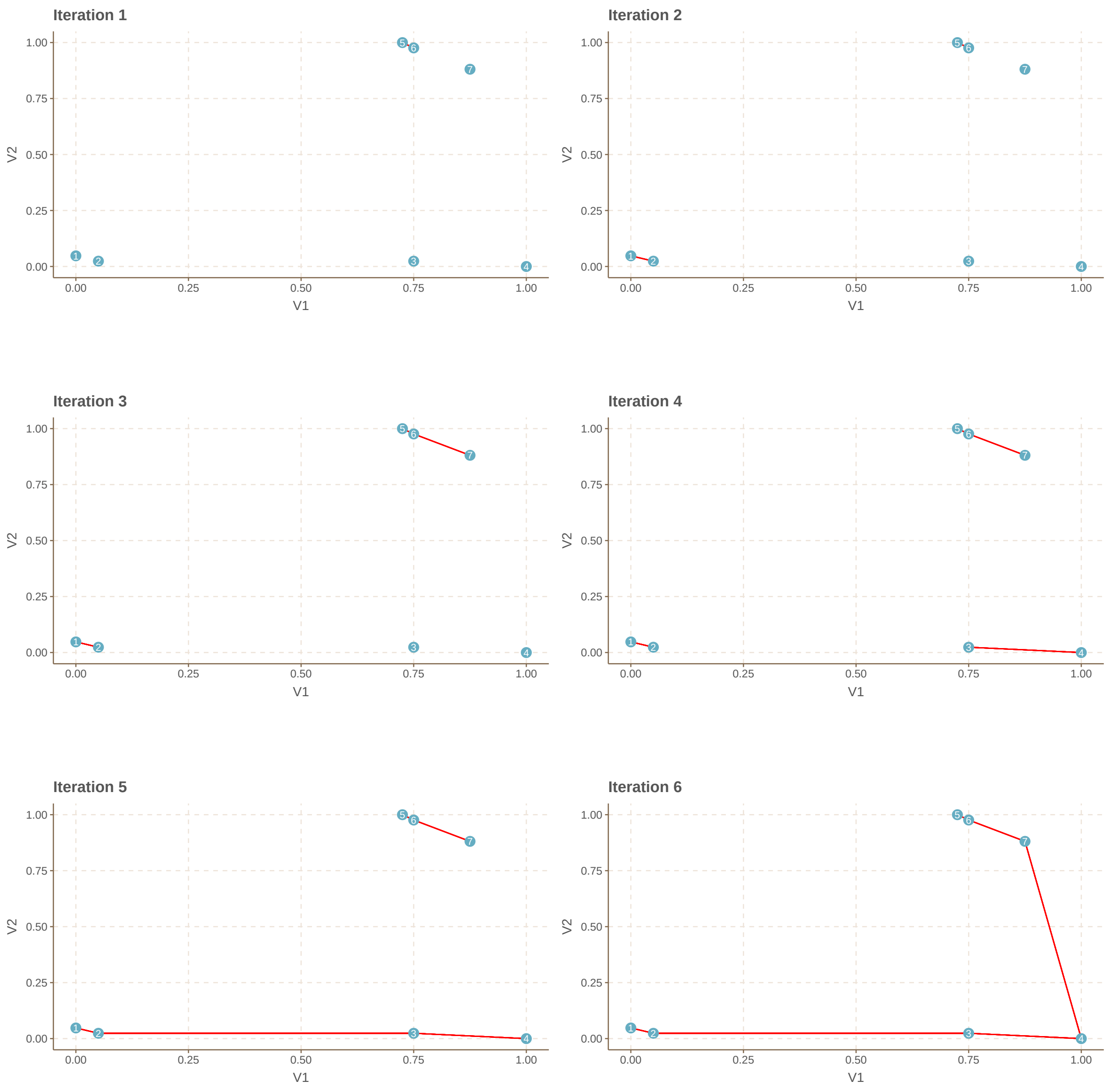


Figure 2.7: Hierarchical Clustering example on an artificial dataset with two variables (V1 and V2). Each blue point represents a data point and the points connected by a red line belong to the same group. The points are numbered for easier reference.

This iteration process can also be visualized in a dendrogram, represented in figure 2.8. Here the x-axis represent the points, with the respective numeration. The y-axis represents the *height*, in this case the height is the similarity measure between two groups. For example, in iteration 3 the distance between the point 7 and the group with points 5 and 6 is close to 0.2.

As stated before, hierarchical clustering does not need a previous specified number of clusters. Using the dendrogram we can *cut* it and decide the optimal number of clusters. One heuristic commonly used is cutting where the gap between two successive combination similarities is largest. For our example that gap would be between 0.3 and 1, creating a total of 3 clusters: $\{\{7, 5, 6\}, \{1, 2\}, \{3, 4\}\}$.

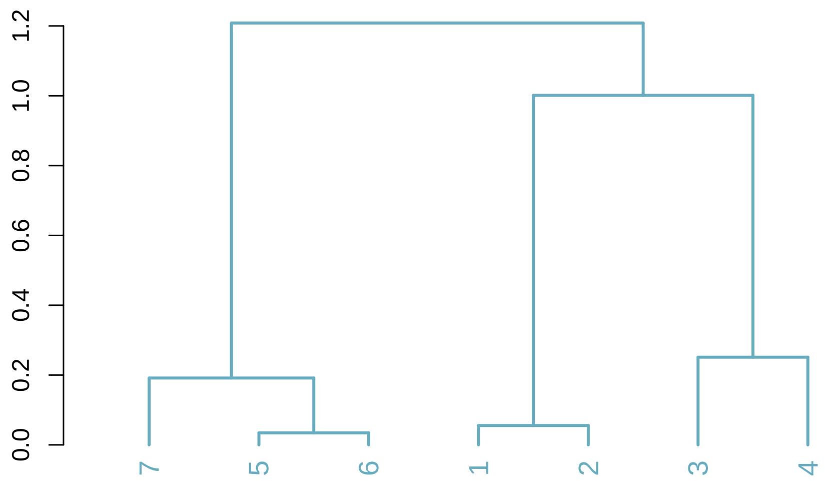


Figure 2.8: Final dengrogram created by the agglomerative hierarchical clustering with complete-linkage.

2.3 User Profiling

In order to help users define their privacy rules, we need to understand how they think, how they react to a certain permission request in a specific context. With this data, we then need to extract knowledge so we understand which characteristics affect specific decisions, *i.e.* profiling. In this section we will explore different techniques to perform user profiling.

According to Eke *et al.* [13] we can categorize the different techniques in the following main categories: Neighbourhood based techniques, Machine learning techniques and Ontology based (figure 2.9).

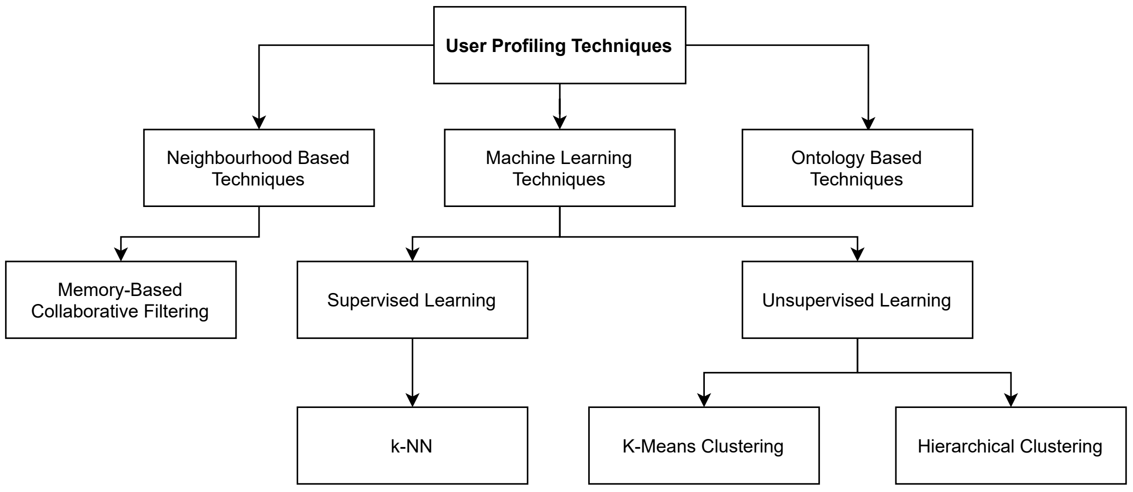

User Profiling Techniques
Neighbourhood Based Techniques
Machine Learning Techniques
Ontology Based Techniques
Memory-Based Collaborative Filtering
Supervised Learning
Unsupervised Learning
k-NN
K-Means Clustering
Hierarchical Clustering

Figure 2.9: User profiling techniques diagram. Adapted from Eke *et al.* [13].

2.3.1 Neighbourhood Based Techniques

Neighbourhood based techniques focus on the commonality between users. On example of this technique is Memory-Based Collaborative Filtering (CF), commonly used in recommender systems [14]. The standard Collaborative Filtering techniques uses rating from a group of users, this information is stored in a $n \times m$ matrix like the one in table 2.1.

Users	**Movies**			
	Good Bye Lenin!	**The Lives of Others**	**Persepolis**	**Les Misérables**
User 1	4	3	5	1
User 2	2	4	1	$\emptyset$
User 3	$\emptyset$	3	2	5
User 4	5	4	4	2
User 5	2	5	$\emptyset$	1

Table 2.1: User ratings matrix examples for 5 users and 4 movies. Each user gives a rating between 0 and 5, if the user did not saw the movie, then the rating is null ($\emptyset$).

The user-based CF stores the ratings matrix in memory and the users ratings vector $r_u = \{r_{u1}, r_{u2}, \ldots, r_{um}\}$ is used to compute the similarity between them. Using similarity metrics like the Pearson correlation (equation 2.9).

$$sim_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \hat{r}_u)(r_{vi} - \hat{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \hat{r}_u)} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \hat{r}_v)}} \tag{2.9}$$

$\hat{r}_x$ represents the average rating for user x and the resulting value varies between -1 and 1. -1 indicates total negative linear correlation, 1 indicates total positive linear correlation and 0 no linear correlation.

For our example in Table 2.1, the pearson correlation between users is presented in table 2.2. From this results we can conclude, for example, that user 1 is very similar to user 4 and very different from users 2 and 3.

	User 1	**User 2**	**User 3**	**User 4**	**User 5**
User 1	1.00				
User 2	-0.98	1.00			
User 3	-0.98	1.00	1.00		
User 4	0.81	-0.19	-0.94	1.00	
User 5	0.42	1.00	-1.00	0.42	1.00

Table 2.2: Pearson correlation coefficient matrix between every user.

2.3.2 Machine Learning Techniques

Some of the techniques explained in section 2.2 can be used for user profiling [15]. In the supervised learning category, k-NN can be used for user profiling. Bradley *et al.* [16] used k-NN to create user profiles based on the job's history and likes and dislikes of job's postings. In order to understand if a job posting is relevant or not to a specific user.

The unsupervised techniques are very useful in user profiling, since many times we are trying to learn the profiles from the data, without having the corresponding labels. Both K-Means and Hierarchical Clustering can be useful, for example, Paireekreng and Wong [17] used K-Means to construct user profiles based on their demographics for mobile content personalisation. Liu *et al.* [18] used hierarchical clustering to build user profiles based on users' privacy app permissions settings. These profiles were then used in a supervised learning model to recommend the users app permissions configurations that are better suited to their profile.

2.3.3 Ontology Based Techniques

Ontology is the branch of philosophy which deals with the nature and the organisation of reality [19]. Ontology based techniques use domain knowledge to build entities, attributes and relationships between them to develop a representation of the users. These techniques usually involve data collections, through questionnaires for example, and then a manual analysis and construction of the ontology model has to be done by specialists of the domain. Tao, Li and Zhong [20] developed an ontology model to discover user background knowledge for personalized web information gathering. For each user, a personalized ontology is built, according to a specific topic, with the goal to gather and query useful from the web.

In this chapter we provided the background knowledge required to understand the following chapters. Mainly, how we structured our project using the CRISP-DM framework, what machine learning is and the different types of learning, and how we can apply them and other strategies to generate user profiles.

Chapter 3

State of the Art

In this chapter is presented a review of the state of the art done for this dissertation. In section 3.1 we review the most common strategies for the creation of privacy profiles. In section 3.2 we introduce the concept of federated learning that will be used in section 3.3, were we describe the state-of-the-art strategies for privacy preserving distributed k-Means. In section 3.4 we describe a privacy preserving distributed strategy to generate profiles using hierarchical clustering and secure scalar product. In the final three sections, we introduce three techniques for clustering data with privacy guarantees: homomorphic encryption (section 3.5), local representatives (section 3.6) and differential privacy (section 3.7).

3.1 Privacy Profiles

Privacy profiles are explicit representations of the users' privacy decisions for a given system. In order to generate these privacy profiles many user profiling techniques have been applied.

Agarwal and Hall [21] developed **ProtectMyPrivacy**, a system for iOS devices that recommends privacy decisions to users based on crowdsourcing. All users send their privacy decisions to a centralized server, then the "experts", about 1% of the most active users, contribute to the recommendation system. This approach does not protect users' privacy and the system only provides universal recommendations, *i.e.* only one profile exists and the same recommendation is sent to every user. They do not take into account each users' individual characteristics or the context, they assume instead that a consensus will emerge from the crowd. Rashidi, Fung, and Vu [22] also developed a crowdsourcing system named **RecDroid**. This system, despite having a more advanced system for recommending privacy decisions than **ProtectMyPrivacy**, it suffers from the exact same problems. Users' privacy decisions are sent to the server, exposing their data to anyone who has access to the server, in a lawful or unlawful way, and only one profile exists for every user.

Y. Zhao, J. Ye, and T. Henderson [23] proposed collaborative filtering strategies to recommend privacy settings for location permissions. However, the data collection was done using surveys, where specific scenarios where presented to the participants and they had to select a privacy setting. It is hard to understand if this simulation is representative of real life decisions as aspirational responses often diverge from real behavior [24]. J. Xie, B. P. Knijnenburg, and H. Jin [25] also proposed a location-privacy recommender system based on collaborative filtering. The data, collected by 40 volunteers in University of St. Andrews, is only related to the location status on the Facebook[1] app. Ismail et al. [26] conducted a user study with 26 participants to create a collaborative-filtering recommender system for Instagram[2] privacy settings. They also used K-nearest neighbours to create profiles with similar users, from which the collaborative-filtering will recommend the permission setting.

Lin *et al.* [27] used Amazon Mechanical Turk to collect privacy preferences with regard to over 800 apps from over 700 participants. The data collected corresponded to the privacy decision for each (permission, purpose) tuple. They applied hierarchical clustering over their dataset and identified four distinct clusters. With the results obtained, they concluded that the profiles were capable of predicting many of a user's mobile app privacy preferences. Liu *et al.* [28] collected privacy preferences from the LBE Privacy Guard[3] application. The collected data from users mainly based in mainland China, corresponds to over 239000 users and 12119 apps, resulting in a total of 28630179 decisions. A problem with their dataset is that this data was collected from users who already had rooted phones, so their dataset is very biased since all participants are all tech savvy. They used k-Means, with each user represented by one vector with every (app, permission, decision) tuple combination, to generated six privacy profiles. With the resulting profiles they were capable of improving the users' permission preferences predictions.

Ravichandran *et al.* [29] generated profiles for location-sharing applications, where users have to choose if they are willing to let others see their locations under specific conditions. The data used in the project was collected in 2008 from 30 users over a period of one week. In order to generate the profiles, they represented each user by training a decision tree to extract policies. With each user represented as a vector of policies, K-means was used to cluster those policies in profiles. They concluded that the resulting profiles were capable of improving the predicting accuracy of the users' choices, in comparison to the model without the profiles.

Sanchez *et al.* developed a privacy-settings recommendation system for fitness devices [30]. Using Amazon Mechanical Turk, they recruited 310 participants to answer a survey. They collected data related to demographics, phone permissions, the type of data collected by the sensors and the entity to which the data is going to be shared with. They proceeded to apply the K-Modes algorithm, similar to the k-Means but more suitable to nominal variables, where instead of using the mean to compute the centroid, the mode is used. With the resulting 6 profiles, where each user belongs to more than one profile, they conclude that the profiles helped the recommendation system to make better decisions.

Liu *et al.* [18] collected permission settings from 84 Android users around the world.

[1] www.facebook.com (2021, August 17)
[2] www.instagram.com (2021, August 17)
[3] http://www.lbesec.com (2021, August 17)

Representing user data as a vector of decisions for every combination of the (app category, permission, purpose) tuple, they applied hierarchical clustering to obtain 7 privacy profiles. These profiles were then used as input in a SVM classifier to predict user preferences to apps' future permission requests. Using profiles as input in the SVM classifier improved the F1-score of the model from 74.24% to 90.02%.

From the reviewed literature we conclude that the strategies for generating privacy profiles can be divided in two categories: hierarchical clustering and k-means, where the user is represented by a single point or multiple points. However, all proposed methodologies rely on a centralized server to create the profiles using the privacy preferences of the users. In this thesis, our objective is to generate these profiles while preserving users' privacy even against the centralized server. Therefore, in the next sections we introduce federated learning, an efficient way to learn neural networks from decentralized data, that we can use to test privacy profiles. The concept of federated learning can also be applied to the k-Means algorithm and learn clusters from decentralized data, that we can use to generate privacy profiles. We also discuss a private and distributed strategy for hierarchical clustering, homomorphic encryption, differential privacy and local representatives. All these strategies are applied in order to obtain clusters in a distributed and private manner, that can be used to generate privacy profiles.

3.2 Federated Learning

McMahan *et al.* presented in 2017 an efficient strategy to learn neural networks from decentralized data [31]. This algorithm, designated by FederatedAveraging takes advantage of the local clients' computing power to apply the gradient descent algorithm to the clients' data in their own devices. At each iteration, every client will send the gradient to the server where it will be averaged and distributed to every client.

In algorithm 2 we can observe the the pseudo-code presented by McMahan *et al.*. The server initializes the global weights, and for each round a random number of clients is chosen. Each one of these chosen clients, using the global weights sent from the server, applies batch learning, as explained in section 2.2.1.3, and returns the resulting weights to the server. The server, after receiving every clients' contribution, applies a weighted average to the weights according to the percentage of points ($\frac{n_k}{n}$) they have. Since the data never leaves the clients' devices, their privacy is safeguarded.

Even though federated learning proved to be an effective way to learning neural networks from decentralized data, it also carries some challenges. For example, expensive communication, every device must communicate their weights every round, and the server must wait for the responses from every device as well. There is also the problem of system heterogeneity, every device contains different hardware with different computational capacities, some low-end devices may not be able to train neural networks locally, stopping them from participating in the learning process.

Algorithm 2 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

```
1:  Server executes:
2:      initialize $w_0$
3:      for each round $t = 1, 2, \dots$ do
4:          $m \leftarrow \texttt{max}(C \cdot K, 1)$
5:          $S_t \leftarrow$ (random set of $m$ clients)
6:          for each client $k \in S_t$ in parallel do
7:              $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
8:          end for
9:          $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
10:     end for
11:
12: ClientUpdate$(k, w)$:
13:     $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size $B$)
14:     for each local epoch $i = 1, \dots, E$ do
15:         for batch $b \in \mathcal{B}$ do
16:             $w \leftarrow w - \eta \nabla \ell(w; b)$
17:         end for
18:     end for
19:     return $w$ to server
```

Virgin *et. al* outlines current works that aim to address these challenges [32]. In order to improve communication efficiency, three strategies have been used: local updating methods, compression schemes, and decentralized training. Local updating methods make devices apply updates locally multiple times for each round, reducing the overall communication. Model compression schemes such as sparsification, subsampling, and quantization can also reduce the size of the messages significantly. Traditionally, all devices connect to the central server to perform the training, however, decentralized training where devices communicate with their neighbours instead of the central server can drastically improve the algorithm performance. To counter effect the system heterogeneity problem, current work focuses on three main strategies: asynchronous communication, active device sampling, and fault tolerance. Asynchronous communication can help mitigate stragglers in the face of device variability. Since some devices may disconnect during the training process, an active device sampling, where a small subset of devices is deliberately selected for training at each round, can reduce the number of dropouts. Implementing fault tolerance strategies allows the model to complete the training process even if some devices are disconnected.

Federated learning will be used in this thesis to test the usefulness of the generated privacy profiles. By training a neural network to predict the `grantResult` variable we can understand if the privacy profiles generated are useful or not by comparing the performance of a neural network trained with the privacy profiles as a feature and a neural network without those features.

In order to maintain the users' privacy during this phase, federated learning is used to train these neural networks.

The concept of federated learning can also be applied to unsupervised learning, in section 3.3 we present a strategy to apply federated learning with k-Means.

3.3 Federated k-Means

Triebe and Rajagopal applied mini batch k-Means together with `FederatedAveraging` (section 3.2) and created the federated k-Means algorithm [33]. It works in a similar way to the `FederatedAveraging`, but instead of the clients sharing weights, they share the centroids' positions and number of observations per cluster. The server applies a weighted average over the centroids' positions based on the cluster size and send back the results to the clients.

One of the problems of this strategy is the centroids initialization method, since we do not have access to the data, we have to randomly select k points from the input space $\mathcal{X}$. However, this method achieves poor performance and may lead to empty clusters [34]. Another problem is that every client will generate the same number of clusters, k. In a scenario where the data is distributed in an extreme non-IID manner, *e.g.* each client only has data belonging to one cluster, the federated k-Means will perform poorly.

3.4 Privacy Preserving Distributed Hierarchical Clustering

As explained in sections 2.2.2.2 and 2.3.2, hierarchical clustering can be used for user profiling, by building a hierarchy of clusters (dendrogram). In order to build this hierarchy, we need the distance between every point in the dataset. If we have n points, the dissimilarity matrix is a $n \times n$ matrix that contains the distance between every point. Using the dissimilarity matrix and a distance measure (section 2.2.2.2) it is possible to build the intended dendrogram.

When the data is distributed across multiple devices, centralized approaches send the clients data to a server which will then proceed to compute the dendrogram. However, this strategy does not protect users' privacy.

Hamidi, Sheikhalishahi and Martinelli [35] presented a general framework for constructing any type of agglomerative hierarchical clustering algorithm in a distributed and secure way. The base concept of this strategy is to construct the dissimilarity matrix without revealing individual points. To compute the distance between two points X and Y, the algebraic equation (3.1) is used, where the server only needs to know $X \cdot X$, $Y \cdot Y$ and the scalar product between the two $X \cdot Y$, where, none of this information reveals the actual individual points.

$$(X \cdot X + Y \cdot Y - 2X \cdot Y)^{\frac{1}{2}} \tag{3.1}$$

The $X \cdot X$ and $Y \cdot Y$ are easy to calculate, the owner of the point can compute the result locally and send it to the server. In order to compute the scalar product between the two in a secure way, the authors introduced the Secure Scalar Product (algorithm 3).

Algorithm 3 Secure Scalar Product

```
1: **Data**: Client$_i$ and Client$_j$ have vectors $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$, respectively.
2: **Result**: Client$_i$ and Client$_j$ obtain securely $X \cdot Y$.
3: Client$_i$ and Client$_j$ agree on random $n \times n$ matrix $C$
4: **Client$_i$** do
5:     Generate random vector $R \leftarrow (r_1, r_2, \ldots, r_n)$
6:     $Z \leftarrow C \times R$
7:     $X_1 \leftarrow X + Z$
8:     Send $X_1$ to Client$_j$
9: Ending
10: **Client$_j$** do
11:     $S_1 \leftarrow \sum_{i=1}^{n} x_{1i} \cdot y_i$
12:     $Y_1 = C^T \times Y^T$
13:     Send $S_1$ and $Y_1$ to Client$_i$
14: Ending
15: **Client$_i$** do
16:     $S_2 \leftarrow \sum_{i=1}^{n} y_{1i} \cdot r_i$
17:     $S = S_1 - S_2$
18: Ending
```

In this algorithm, each client holds one point, a vector of length n. Each client has a different role, client$_i$ which we define as privilege client, because he is the one to first observe the scalar product, and has the duty to share it with client$_j$. First, the two clients agree on a random $n \times n$ matrix C. This can be done using diffie-hellman key exchange algorithm, for example. The privilege client will generate another matrix R and applies the scalar product with C and sums the result with his point X. The result is sent to the second client as matrix S_1. The second client applies the dot product between the received matrix and his point Y and also sends this result to the privilege client as Y_1. The second client also sends the scalar product between matrix C and his point Y to the privilege client. Finally, to obtain the scalar product $X \cdot Y$, the privilege client applies the dot product between the random vector R and the vector Y_1, and subtracts the result to the matrix S_1.

In the end, each client and the server only hold the scalar product and euclidean distances between every point. With the distances between every point, it is possible to build the dendrogram using a distance measure that only uses the euclidean distance between two points. For example, the centroid linkage (section 2.2.2.2) could not be used in this strategy.

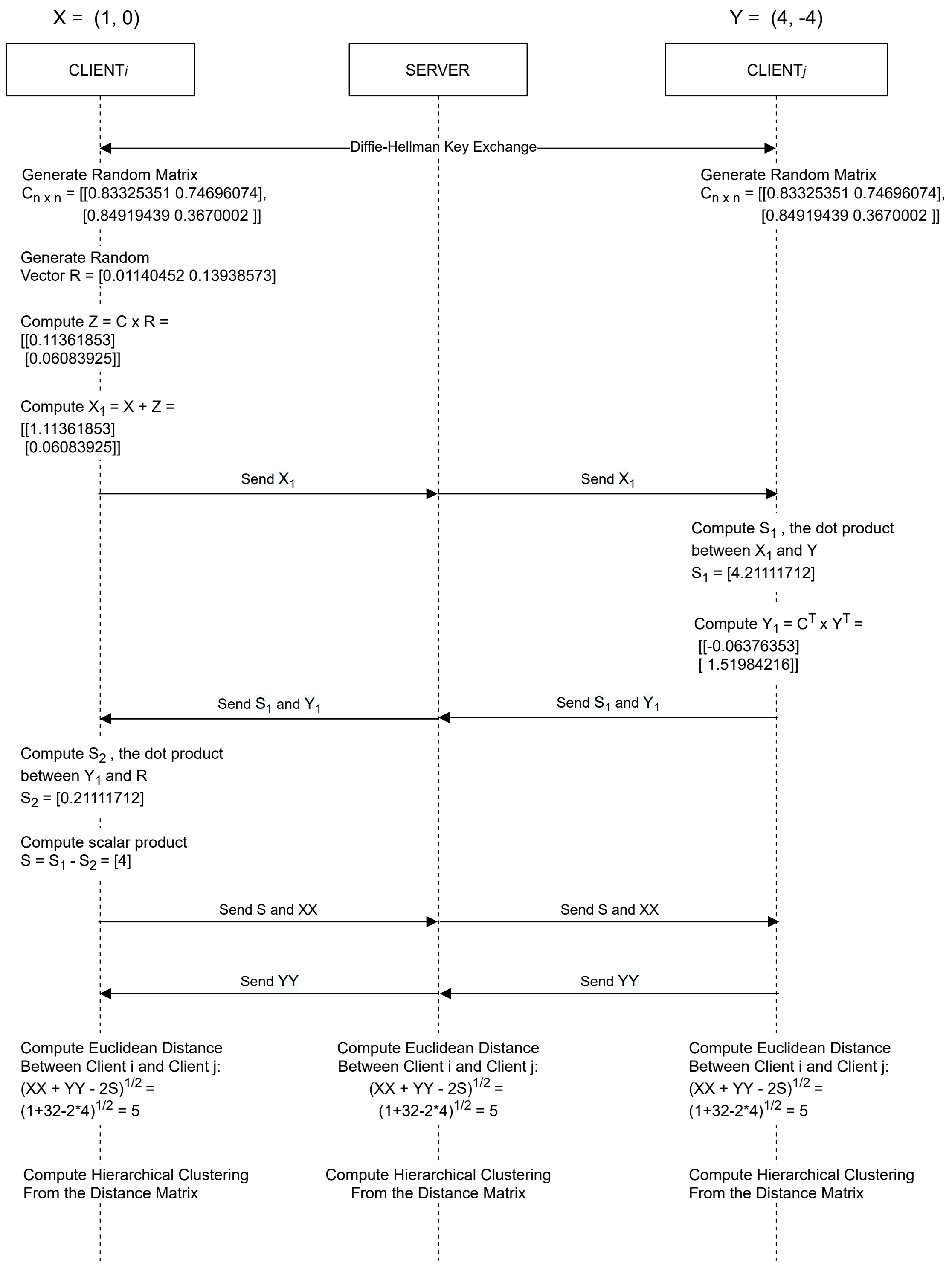

X = (1, 0)
Y = (4, -4)
CLIENT*i*
SERVER
CLIENT*j*
Diffie-Hellman Key Exchange
Generate Random Matrix $C_{n \times n}$ = [[0.83325351 0.74696074], [0.84919439 0.3670002]]
Generate Random Matrix $C_{n \times n}$ = [[0.83325351 0.74696074], [0.84919439 0.3670002]]
Generate Random Vector R = [0.01140452 0.13938573]
Compute Z = C x R = [[0.11361853] [0.06083925]]
Compute X_1 = X + Z = [[1.11361853] [0.06083925]]
Send X_1
Send X_1
Compute S_1 , the dot product between X_1 and Y S_1 = [4.21111712]
Compute $Y_1 = C^T \times Y^T$ = [[-0.06376353] [1.51984216]]
Send S_1 and Y_1
Send S_1 and Y_1
Compute S_2 , the dot product between Y_1 and R S_2 = [0.21111712]
Compute scalar product $S = S_1 - S_2$ = [4]
Send S and XX
Send S and XX
Send YY
Send YY
Compute Euclidean Distance Between Client i and Client j: $(XX + YY - 2S)^{1/2}$ = $(1+32-2*4)^{1/2}$ = 5
Compute Euclidean Distance Between Client i and Client j: $(XX + YY - 2S)^{1/2}$ = $(1+32-2*4)^{1/2}$ = 5
Compute Euclidean Distance Between Client i and Client j: $(XX + YY - 2S)^{1/2}$ = $(1+32-2*4)^{1/2}$ = 5
Compute Hierarchical Clustering From the Distance Matrix
Compute Hierarchical Clustering From the Distance Matrix
Compute Hierarchical Clustering From the Distance Matrix

Figure 3.1: Secure Hierarchical Clustering Diagram.

3.4.1 Security Concerns

The algorithm presented by Hamidi, Sheikhalishahi and Martinelli [35] allows the computation of the dendrogram without sharing individual points. However we found some security problems, that allows the extraction of information or the manipulation of the final model.

- Although the server and the users do not know the other users' individual points, they will know the distance between every two clients. So, they are able to understand how similar or different they are between each other.
- There is a need to trust that the privileged client, the one who computes S first, will be well behaved and send the real S to the server. Otherwise, the resulting dendrogram for the server and the other clients will be manipulated.
- In a situation where one client has n points with n coordinates and the other client has 1 point with n coordinates, if all the n points the first client has are n unit vectors, the dot products will reveal the second client's point. Which is particularly dangerous if the attacker is the privileged client, because the victim has no way to know it is under attack.

These security problems are not addressed in [35], however we found solutions to for the first two problems.

- TOR[4] is an open-source, free software which enables anonymous communication by relaying the traffic through an overlay network. If clients use TOR, or another anonymization technique like VPN or proxies to communicate with the server, it is not possible anymore to know which users are similar or different, because they are anonymous.
- For the manipulation vulnerability, it can not be prevented, but it can be detected if instead of running the algorithm only once, the algorithm can be executed multiple times, with different clients as the privileged ones. In this situation, the server can compare the resulting scalar products, if they are different, it means that at least one user faked the results.

The third vulnerability is harder to detect or avoid, the only scenario where the attacker's ability is restricted is when his number of points is much lower than the number of coordinates. Since an attacker must have a point for each coordinate (unit vectors), the less points the attacker can use, the less coordinates the attacker can uncover from the other users' points. This problem makes this strategy suitable for the case where a client is represented by a single vector. In this case an attacker can only reveal one feature from the other clients' vectors.

We can group all the remaining work we reviewed about privacy preserving clustering in three categories: homomorphic encryption, differential privacy and local representatives. In the next three subsections we describe these approaches, their advantages and issues.

3.5 Homomorphic Encryption

Many approaches have used homomorphic encryption based on the Pailliar cryptosystem [36–39]. They used on privacy preserving clustering algorithms by encrypting the entire dataset before

[4]https://www.torproject.org (2021, August 17)

sending it to the server. By using homomorphic encryption, the server is able to compute the distances between the clients' data and the global centroids without decrypting it. This information is sent back to the clients, who will decrypt and assign each point to the nearest cluster. By sending back the sum and number of points in each cluster, the server can compute the new centroids' values. While this strategy is robust to non-IID data, it is not efficient or scalable as it requires the encryption and decryption of the entire dataset. Additionally, the computations over encrypted data need more computation power when compared to the *plaintext* computations. The sum and number of points per cluster shared with the server could be used by the server to know which cluster(s) does the clients belong to. Since the client will have the distances from every point to every global cluster, it is possible for the client to apply trilateration to find the global centroids kept by the server [40], so this strategy is also not mutually private.

3.6 Differential Privacy

Another common strategy is combining k-means clustering with differential privacy (DP) [41–44]. Differential privacy is considered the state of the art in anonymization mechanisms and consists on adding "statistical noise" that is significant enough to protect client's privacy, but small enough to not affect the model performance. Each client applies differential privacy to their local datasets and send them to the server. The server will aggregate all datasets and apply the k-Means algorithm. Although this strategy is very efficient and robust to non-IID data, since the data points from all clients are sent to the server, it lacks a systematic approach, it is hard to define the amount of noise as it highly depends on the dataset [45]. Other problems of differential privacy include the inherent uncertainty in the answer and the fact that the guarantees of immunity to background knowledge attacks are overstated [45]. Background knowledge may allow an adversary to learn information on one individual from a differentially private answer that is computed from the values of other individuals. It has also been show recently, that DP can increase existing biases and have substantial impacts on the accuracy [46].

3.7 Local Representatives

Another common strategy is to select a subset of local points and apply the k-Means algorithm over them. Soliman *et al.* [47] presented a strategy that is efficient and robust against non-IID distributions. They run the k-Means algorithm locally on each client and using HyperLogLog counters they share the centroids and the approximate number of observations per centroid in a decentralized fashion with the other clients. Then a weighted averaging over all the centroids is done to find the global centroids. Januzaj *et al.* [48] have a similar strategy, where local representatives are extracted on site and then shared with the server, who will then perform a global clustering over the representatives. Both strategies are efficient and robust to non-IID data, but lack privacy. In the first strategy, the clients will know to which cluster(s) the other clients belong to. The second strategy it is focused on efficiency and the clients send actual observations to the server, but fewer than the entire dataset.

Table 3.1: Summary of current strategies.

Category	**Strategies**	**Efficiency**	**Privacy**	**Robustness to Non-IID**
Federated k-Means	[33]	Good	The server knows which cluster(s) the clients belong to.	Bad, by averaging the local centroids, in extreme non-IID cases, the final centroids will reside in the middle of the dataset.
Homomorphic Encryption	[36, 38]	Bad, because it needs to encrypt entire/decrypt datasets.	Good	Good
Homomorphic Encryption	[37]	Bad, because it needs to encrypt entire/decrypt datasets.	Server knows which cluster(s) the clients belong to.	Good
Homomorphic Encryption	[39]	Improved by using a Map Reduce infrastructure. Clients still need to encrypt and decrypt the entire dataset and send it to the server.	Good	Good
Differential Privacy	[41–44]	Good	Depends on the implementation.	Good
Local Representatives	[47]	Good	Clients know which cluster(s) the other clients belong to, and the approximate number of observations per cluster.	Good
Local Representatives	[48]	Good	Server knows which cluster(s) the clients belong to. The local representatives sent to the server are actual observations from the dataset.	Good

Table 3.1 summarizes the aforementioned works by their efficiency, privacy and robustness to non-IID data. From this table, we can observe that none of these works fully accomplishes the three requirements. In the case of federated k-Means, it is efficient but has an issue with privacy and it is not robust to non-IID. The homomorphic encryption strategies are not efficient, because they involve encrypting and decrypting entire datasets. The privacy provided by Differential Privacy is highly dependent on the implementation and the local representatives also have issues with privacy, mainly disclosing which cluster(s) the clients belong to.

Another aspect that we found in the literature is the lack of approaches to securely evaluate the final model. A very important step in the machine learning pipeline is the model evaluation, where we use one or more metrics to understand how well the model fits the data. These metrics will allow the comparison between models, since we are implementing strategies to build models in a secure way, we also need strategies to compute these metrics in a secure way as well. Another problem arises in privacy preserving k-Means, where we cannot see the data anymore, which is how to choose the ideal number of clusters k.

In this chapter we provided the state-of-the-art of privacy profile generation, privacy preserving clustering and neural networks training from distributed data using federated learning. In chapter 4 we present our approach for privacy preserving distributed k-Means that fulfills three requirements: efficiency, privacy, and robustness to non-IID data, that the current state-of-the-art strategies cannot not satisfy. Our strategy will be later used in chapter 5 to generate mobile's privacy preferences profiles.

Chapter 4

Efficient Privacy Preserving Distributed k-Means for Non-IID Data

In this chapter we detail our proposed approach towards an efficient and secure k-means algorithm that is robust to non-IID data. As explained in section 3.3, there is no strategy that is capable of fulfilling the three requirements: Efficiency, Privacy, and Robust to non-IID data. So, in order to tackle this problem, we developed our own strategy capable of meeting the three requirements, and an approach to select the best number of clusters and test the resulting model in a secure way as well. This mechanism can be used for any generic type of data. However, in Section 5 we implement it towards generating privacy profiles and compare it to centralized and other secure approaches

In section 4.1 we describe our proposed approach and in section 4.2 we present a detailed evaluation of the privacy, efficiency and robustness to non-IID data of our strategy.

4.1 Description

We consider a setup where a set of clients, each possessing its own dataset of N points with M features, aims to cluster their data. In this scenario, we propose a mechanism that allows the clients to cluster their data, alongside the other clients without the server knowing the clients' points. The operation of our proposed mechanism is illustrated via an example in figure 4.1. In the first stage, each client performs Diffie-Hellman Key Exchange so as to agree on a seed with the server. Then, they will compute the sum and number of points in their local dataset. These two statistics will be masked and securely sent to the server using secure aggregation ("Send Masked Statistics" step on figure 4.1). **Secure aggregation** presented by Bonawitz *et al.* [49] allows the secure sum of vectors using Diffie-Hellman Key Exchange, where the resulting secret will be used as a seed to generate random vectors. These random vectors will be the same for each pair of clients: one of the clients adds the random vector and the other subtracts the vector to their contributions and both send the result to the server. To the server each contribution will be indistinguishable from a random vector. But when the server adds all the contributions, the random vectors cancel out, retaining only the real sum of every client's contribution. Using this strategy the server is able to compute the center point of all clients' datasets. In the example from figure 4.1, $\frac{(28+56,50+58)}{100+200} = (0.28, 0.36)$. To improve the k-Means performance we choose k random points close to the center point as initial centroids. Since the server cannot choose k points from the data (known only to the clients) and choosing random points from the input space would result in a poor performance [34].

While the server is initializing the centroids, the clients will apply the k-means to their local dataset to generate local clusters. Each client uses the silhouette score to estimate the best number of local clusters as follows. Given the following metrics:

$$a(i) = \frac{\sum_{j \in C_i, i \neq j} d(i,j)}{|C_i| - 1} \qquad b(i) = \min_{k \neq i} \frac{\sum_{j \in C_k} d(i,j)}{|C_k|} \qquad s(i) = \frac{a(i) - b(i)}{max\{a(i), b(i)\}}$$

where C_i is the set of points in cluster i and $d(\cdot)$ is the distance metric. We can interpret $a(i)$ has how well is point i assigned to its cluster and $b(i)$ as the smallest mean dissimilarity of any cluster except C_i. The silhouette score is defined as the mean $s(i)$, over all observations of the local dataset. We compute the silhouette score for $2 \leq k \leq K$, where K is the number of global clusters, defined by the server. Then, the number of clusters for the model with the maximum silhouette score is chosen as the optimal local number of clusters.

The resulting centroids for the local clusters will be sent to the server in a secure way, using homomorphic encryption. **Homomorphic encryption** allows one to perform calculations over encrypted data, in such a way that when the result is decrypted it yields the same result as if the operations were made over unencrypted data. The centroids will be encrypted and sent to the server, that will compute the distances between the encrypted local centroids and the unencrypted global centroids. To calculate the distances between the pairs of points, we will need a schema that is able to subtract scalars from encrypted numbers and multiply an encrypted number by itself. The CKKS schema [50] offers us that possibility. By only providing approximate results, it is more efficient, but still precise enough for machine learning models [50], as we also assess. Even with the encrypted centroids, the server would still be able to known how many clusters each client has. In order to hide this informantion, each client will always send information about k clusters, by adding random centroids whenever needed. In the example from figure 4.1, the server defined $k = 2$ as the number of clusters and Client B found a single centroid, namely $(0.28, 0.29)$. So, the client is going to send $[c_1 = (0.28, 0.29), c_2 = (0.56, 0.89)]$, where the second centroid is randomly created to deceive the server.

The server computes the squared distance between the encrypted local centroids and every global centroid. The encrypted distances are sent back to the users, who decrypt them and assign each local centroid to a global cluster. Now each client sends to the server the sum and the number of points for each cluster – this information is needed for the server to compute the new centroids (equation (2.8)). In the example from figure 4.1, the resulting statistics are $[((0.28, 0.29), 1), ((0, 0), 0)]$, for client B, where $(0.28, 0.29)$ represents the sum of points in this cluster and 1 the number of points in the cluster. The distances for the random centroid are ignored and both the sum and the number of points are set to zero. Since only the local centroids are sent to the server, this will difficult the trilateration attack, since in order to infer the global centroids from the distances, the client needs to have more points than the number of features[51]. To update the global centroids in the server without revealing the clients' individual statistics, secure aggregation is employed again to send the masked statistics. With these statistics, the server is able to compute the new global centroids' values by dividing the total sum by the total number, just like in equation (2.8). These steps will be repeated until the new centroids are equal to the previous ones or a maximum number of iterations is performed.

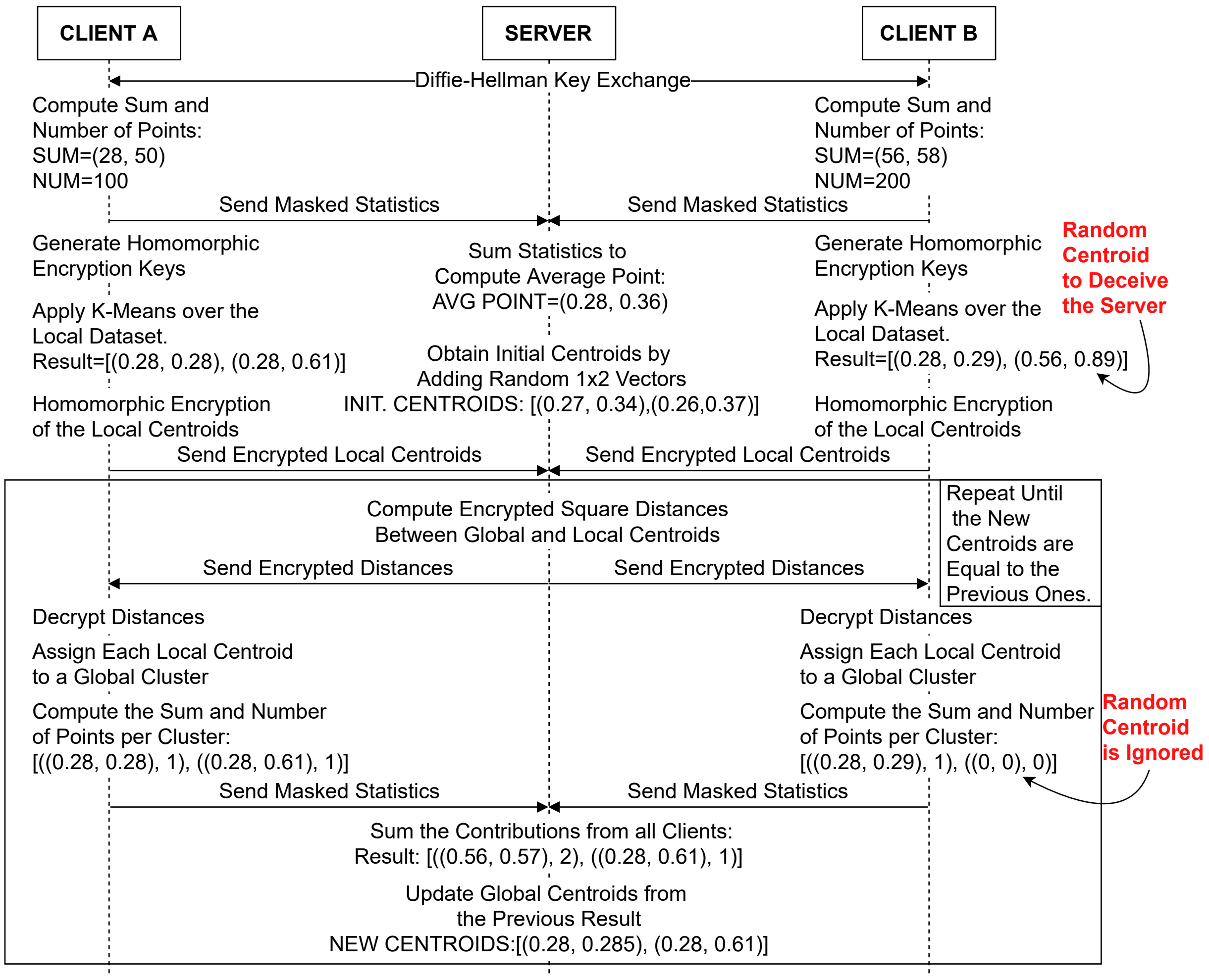

CLIENT A
SERVER
CLIENT B
Diffie-Hellman Key Exchange
Compute Sum and Number of Points: SUM=(28, 50) NUM=100
Compute Sum and Number of Points: SUM=(56, 58) NUM=200
Send Masked Statistics
Send Masked Statistics
Generate Homomorphic Encryption Keys
Sum Statistics to Compute Average Point: AVG POINT=(0.28, 0.36)
Generate Homomorphic Encryption Keys
Random Centroid to Deceive the Server
Apply K-Means over the Local Dataset. Result=[(0.28, 0.28), (0.28, 0.61)]
Obtain Initial Centroids by Adding Random 1x2 Vectors INIT. CENTROIDS: [(0.27, 0.34),(0.26,0.37)]
Apply K-Means over the Local Dataset. Result=[(0.28, 0.29), (0.56, 0.89)]
Homomorphic Encryption of the Local Centroids
Homomorphic Encryption of the Local Centroids
Send Encrypted Local Centroids
Send Encrypted Local Centroids
Compute Encrypted Square Distances Between Global and Local Centroids
Repeat Until the New Centroids are Equal to the Previous Ones.
Send Encrypted Distances
Send Encrypted Distances
Decrypt Distances
Decrypt Distances
Assign Each Local Centroid to a Global Cluster
Assign Each Local Centroid to a Global Cluster
Compute the Sum and Number of Points per Cluster: [((0.28, 0.28), 1), ((0.28, 0.61), 1)]
Compute the Sum and Number of Points per Cluster: [((0.28, 0.29), 1), ((0, 0), 0)]
Random Centroid is Ignored
Send Masked Statistics
Send Masked Statistics
Sum the Contributions from all Clients: Result: [((0.56, 0.57), 2), ((0.28, 0.61), 1)]
Update Global Centroids from the Previous Result NEW CENTROIDS:[(0.28, 0.285), (0.28, 0.61)]

Figure 4.1: Sequence diagram (with example) of the proposed algorithm.

Since the central server does not have access to the data anymore, in order to find the best global number of clusters, we developed a privacy preserving way to use the elbow method heuristic to determine the best number of clusters in the dataset. It's possible to plot the inertia vs. the number of clusters without sharing any individual data to the server. Using this plot, the server can make an informed decision about the optimal number of global clusters. Since the inertia is the sum of squared distances of every observation to their closest cluster center, this information can be computed locally. The sum of squared distances of all local observations (local inertia) is sent to the server using the secure aggregation strategy, where all the users' contributions would be summed to obtain the global inertia.

4.2 Evaluation

To evaluate this strategy we ran the algorithm through 114 artificial benchmark datasets[1]. To simulate an environment of non-IID data, we created 20 clients, where each has a random number of clusters from 1 to k (depending on the dataset). Every client has a random number of 70 to 90 observations from one cluster and a random number of 1 to 30 observations for each of the remaining clusters, except in the case of only one cluster, with 100 observations from the same cluster. All random numbers are generated from a uniform distribution.

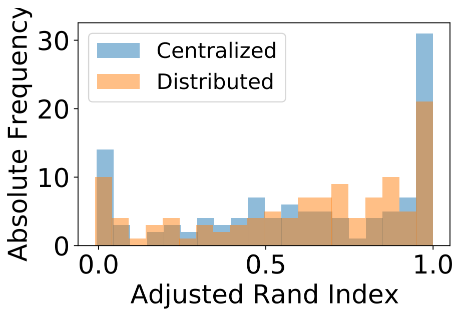
Since we have access to the cluster labels of the benchmark datasets, we used the Adjusted Rand Index (ARI) metric to compare our strategy with the centralized k-means over the entire dataset [52]. Let K_t be the clustering ground truth and K_p the clustering done by the model. If a is the number of pairs of elements that are in the same set in K_t and K_p and b the number of pairs of elements that are in different sets in K_t and K_p, then

$$\mathrm{RI} = \frac{a+b}{C_2^{n_{\text{samples}}}} \tag{4.1}$$

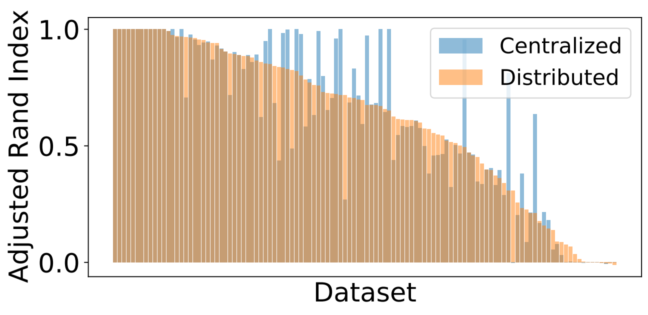
$$\mathrm{ARI} = \frac{\mathrm{RI} - E[\mathrm{RI}]}{\max(\mathrm{RI}) - E[\mathrm{RI}]} \tag{4.2}$$

Since the Random Index (RI) in (4.1) does not guarantee a value close to zero for random label assignments, we resort to the ARI in (4.2) that counters this effect by subtracting the expected RI. We run the strategy 30 times for each dataset and the one with lowest inertia is chosen. The inertia metric corresponds to the sum of the distances of all points within a cluster to the respective centroid [53]. Each client computes the local inertia and, using secure aggregation, the server obtains the total inertia, without knowing individual contributions. This metric can be used to compare models and allows the server to use the *elbow method* to estimate the best number of clusters in the dataset [54]. We chose the best inertia because our goal is to prove that our strategy can achieve a good performance with few repetitions. Additionally, this procedure replicates a real world scenario where one computes the inertia securely in order to choose the best model. We compared the results of our strategy to the centralized version in terms of efficiency and robustness to non-IID, since the centralized version achieves the best results in these two criteria.

[1] https://github.com/deric/clustering-benchmark (2021, August 17)



(a) Adjusted rand index distribution.



(b) Adjusted rand index by dataset.

Figure 4.2: Adjusted rand index distribution and adjusted rand index by dataset.

4.2.1 Robustness to non-IID Data

Figure 4.2a shows the distribution of the ARI over the 114 datasets of the proposed strategy (orange) versus the centralized k-means (blue). From this plot, we can see that our strategy and the centralized k-means have a similar distribution, however our proposal achieves smaller values at the extremes. In fact, our strategy was not able to achieve ARI values above 0.9 in around 10 datasets. However, it also produced fewer ARI values closer to 0 and achieved more ARI values between 0.6 and 0.9. Overall, we can conclude that this strategy is consistent with the centralized version of k-means.

Figure 4.2b presents the ARI score for every dataset. The number of datasets for which our approach performed better than the centralized model is 56, worse in 43 and had the same performance in 15 of the 114 datasets. As aforementioned, when the centralized model is better, it is usually better by a larger difference than when the centralized model is worse.

Figure 4.3 compares the differences between ARI scores when the centralized model is worse than our strategy (blue distribution) and when the centralized model is better than our strategy (orange distribution). We can observe, as previously discussed, that the two distributions are similar. More than 50% of the times, when the centralized model is better, the ARI differences are lower than 0.1 and the same happens when the centralized model is worse. However, we see a more uniform distribution in the interval between 0 and 0.1 when the centralized model is worse, as to when the centralized model is better, where approximately 45% of the times the difference is practically 0.

From figures 4.2 and 4.3 we conclude that the proposed strategy achieves a similar performance to the centralized k-means. Specifically, it achieves a better ARI score in more datasets than the centralized version, but the majority of the differences are lower than 0.1. Therefore, our strategy, while decentralized, is robust against non-IID data.

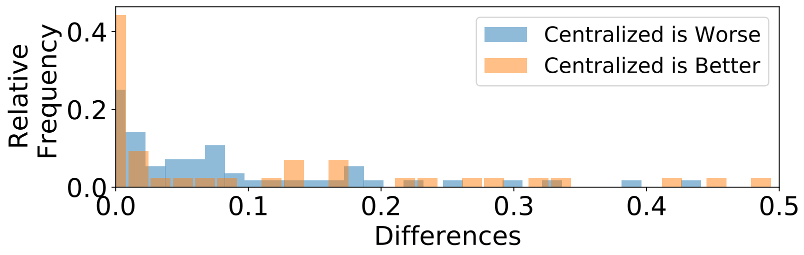
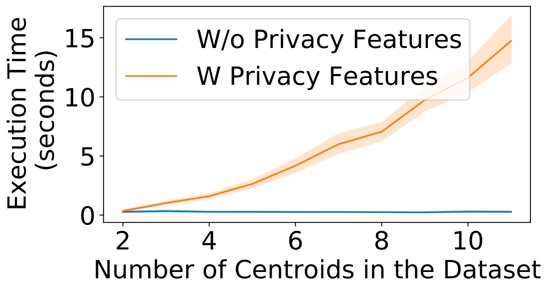
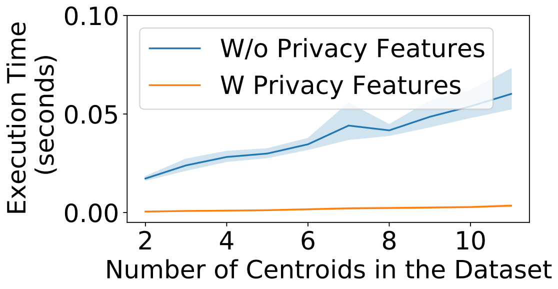


Figure 4.3: Adjusted rand index differences.



(a) Clients' execution time.



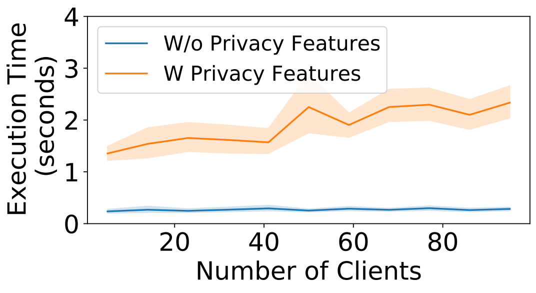
(b) Server's execution time.

Figure 4.4: Execution time versus the number of centroids on the clients and server side (95% confidence intervals are shaded). The case with privacy features corresponds to our proposed method.

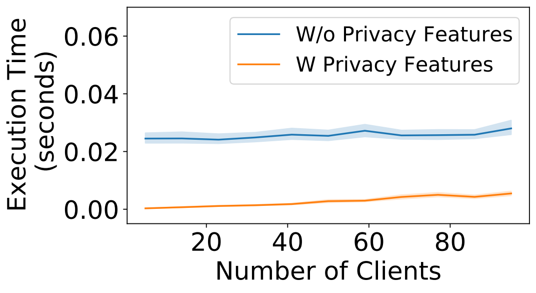
4.2.2 Efficiency

In order to understand the efficiency of our final strategy we compared it to the strategy without privacy features, *i.e.* without secure aggregation and homomorphic encryption. To measure the efficiency, we measured the execution time, without taking into account the network delay, in three dimensions: number of points, number of clusters and number of clients. For each dimension we measured the time spent by the clients and by the server. To facilitate the efficiency evaluation, we assume the server to know how many clusters there are in the dataset. Since the steps executed by the clients are done in parallel, the reported clients' execution time corresponds to the time of the *slowest* client. We trained the model 30 times and in the following figures is presented the mean values and the 95% confidence interval.

In Fig. 4.4 we present the execution time according to the number of centroids in the dataset. We can observe that for the strategy with privacy features, most of the work is done by the clients (see Fig. 4.4a). This is expected, since the clients perform local clustering followed by the encryption of the centroids. The server execution time shown in Fig. 4.4b is much higher without privacy features than with privacy. This result is expected since without privacy features the server needs to apply the full clustering algorithm as opposed to only computing the distances and updating the global centroids. Nevertheless, for the server side, both strategies have a rather low server execution time – below 0.1 seconds. Thus, the predominant execution time is the



(a) Clients' execution time.



(b) Server's execution time.

Figure 4.5: Execution time versus the number of clients on the clients and server side (95% confidence intervals are shaded). The case with privacy features corresponds to our proposed method.

clients' time.

In Fig. 4.5 we present the execution time according to the number of clients. The strategy with privacy takes more time to execute in the client side and slowly increases with the number of clients. Specifically, its execution times are 1 to 2 seconds higher than the strategy without privacy features. This latter strategy has a higher server execution time, but the difference is low, below 0.025, and thus the total execution time is mostly affected by the clients' execution time.

We additionally measured the execution time as a function of the number of points in the dataset. Our results indicate that on both strategies, the execution time is not affected by the number of points in the dataset (between 100 and 50000). On the clients' execution time, the strategy with privacy features takes around 1 second more than the strategy without privacy features. On the server side, this latter strategy takes around 0.025 seconds more than the strategy with privacy features. We omit these plots due to the lack of space.

Overall, we consider the algorithm to be efficient. While it takes more time than the strategy without privacy features it is only significant when the datasets have many centroids. In this case, our strategy will take longer to execute.

To conclude, our proposal is robust to non-IID data while preserving the privacy of individual contributions, as the server only accesses the encrypted local centroids. Additionally, it is efficient at the expense of a slightly higher execution time, yet within reasonable bounds when compared to the centralized version. In particular, letting n_{pts}, n_{col}, n_{cent} and n_{cli} respectively represent the number of points, coordinates per point, centroids and clients, our method achieves a complexity of $O(n_{cent}^2 \times (n_{pts} \times n_{col} + n_{col}^2 + 1) + n_{pts} \times n_{col})$ for each client and $O(n_{cli} \times n_{cent}(n_{cent} + n_{col}))$[2] for the server. When compared to the strategy without privacy features, with a complexity of $O(n_{pts} \times n_{cent} \times n_{col})$ for each client and $O(n_{cli} \times n_{cent}(n_{cent} \times n_{col}))$ for the server, this confirms that the cost of our scheme lies in the increase of the client execution time, with the server

[2]We assume a constant time complexity for multiplication between the encrypted centroids and the plaintext global centroids, according to [50].

execution time becoming even lower than for the base strategy without privacy.

In this chapter we propose a privacy preserving clustering algorithm that is private, efficient and robust to non-IID. As far as we known, there is no method that can fully fulfill the three. Our strategy based on local representatives, homomorphic encryption and secure aggregation is capable of outperforming or matching the centralized version in more than half of the datasets in terms of Adjusted Random Index in a non-IID scenario. In terms of efficiency, the time complexity moves from the server to the clients, yet leading to an overall time complexity within reasonable bounds when compared to the centralized version. Moreover, given the lack of secure metrics to evaluate models in a distributed environment, we introduced secure inertia, a method to compute the inertia of the model without sharing individual contributions. This strategy will be applied in chapter 5 to generate mobile's privacy preferences profiles.

Chapter 5

Secure Generation of Privacy Profiles

In this chapter, we show the usefulness of profiling mechanisms for automated privacy protection. For that, we resort to a real-world dataset obtained in the COP-MODE project. This dataset, that comprises 93 users, contains 2180302 permissions, from which 65261 were manually answered, with an average of 701.73 answered requests per users. We target prediction of the privacy decisions of users by clustering the users in order to generate privacy profiles. For this, we use two algorithms, hierarchical clustering and k-means, the resulting privacy profiles will be used as features to predict the users' answers to permission requests.

Our final goal is to design and evaluate a secure mechanism to predict the users' permission requests using a profile-based optimized model. To achieve this, we generate the privacy-profiles in a secure way, using privacy preserving distributed hierarchical clustering and efficient privacy preserving distributed k-means for non-IID data, and then, using these profiles, train a model in a secure way, using federated learning, to predict the users' responses to the permissions requests. We also need to evaluate these strategies against the non-secure ones, to understand if the security component of the strategies is influencing the models' performance. In this chapter, we describe the implementation process done in order to accomplish these objectives.

We start by describing the data collection process, how we choose to store the data, and finally, we characterize the data collected (section 5.1). We then proceed to explain how can we apply hierarchical clustering and k-Means to generate privacy profiles using the COP-MODE dataset (sections 5.2.1 and 5.2.2). Finally, we explain how we can use these two strategies with privacy guarantees to generate the privacy profiles (sections 5.2.1.1 and 5.2.2.1).

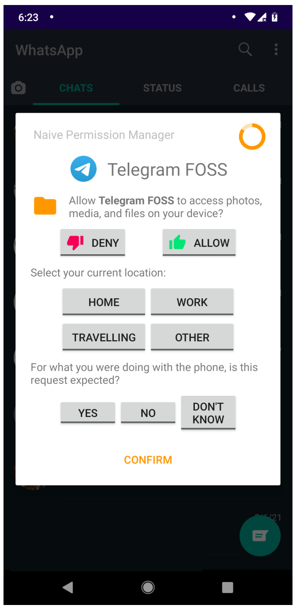
In order to evaluate the usefulness of the generated privacy profiles, we use them as features in a neural network to predict the grant result, *i.e.* whether the user would accept or deny a permission (Section 5.3). The score of this model is then used as a metric for the usefulness/utility of the privacy profiles (Section 5.4).

5.1 Data

In this section we describe and explore the data collected from the COP-MODE project. Section 5.1.1 describes the data collection process done in the COP-MODE project, Section 5.1.2 describes how and why the data collected was stored, and finally, in Section 5.1.3 an exploratory data analysis is done on the dataset.

5.1.1 Data Collection

To collect the necessary data an app was developed outside the scope of this thesis (COP-MODE Naive Permission Manager) to intercept permission checks and request specific data from the users. When the app intercepts a request, the popup presented in figure 5.1 is presented to the user.



Allow **Telegram FOSS** to access photos, media, and files on your device?

Select your current location:

For what you were doing with the phone, is this request expected?

Figure 5.1: Popup request for intercepted permission request.

The user provides the following data: answer to the request, the semantic location and if it was expecting that request to appear. At the time of the prompt, the app also collects contextual data regarding the phone state and the user context. Information such as background and foreground running apps, network status, geographic coordinates, devices in neighborhood, etc. The complete description of the data collected by the app is presented at appendix A.1 and an example of the data collected from one request is presented in appendix A.2.

The smartphones need to be rooted in order to intercept the permission requests, but we do not want to include only people with rooted phones, since it would produce a biased dataset. So, we provided people with a rooted smartphone with their personal apps already installed, along with the COP-MODE Naive Permission Manager.

The participants were recruited through word-of-mouth and university's mailing lists. Each user had to use the phone at least for one week, although they were sometimes allowed to continue using it for longer, if they desired it. Participation was rewarded with a FNAC gift card, given that they answered to a minimum number of 50 requests. The data was collected through several campaigns occurring between end of July 2020 and end of March 2021

5.1.2 Data Storage

All the data from the mobile users was stored as JSON format in compressed files. In order to manipulate and use the data, an adequate and efficient storage solution is needed. In table 5.1, a comparison of types of databases is done in order to identify the best suitable database for our problem.

Since our data is stored in JSON, where each document has a different number of fields, we need a database that is flexible and schemaless. Considering that we are going to perform analytical queries over millions of documents, we also want a database with good performance on analytic functions, like aggregation operations.

Looking at table 5.1, we can immediately exclude the relational databases, because they are suited for a single data model, the relational one, and we need support for JSON documents. We can also exclude key-value store databases, since they lack analytics features. Graph databases are not efficient at large-volume analytics queries and they are also more oriented towards graph data. Object oriented databases are ideal for complex object relationships and are not efficient for simple ones, which is the type of data we have, data with simple relationships. Between the two ones that were not excluded, columns-oriented and document store, we decided to chose document store, more specifically, MongoDB. Columns-oriented databases have some restrictions when it comes to data manipulation since it is difficult to manipulate an entire row, and MongoDB stores the data in JSON format directly and comes with a powerful querying language, allowing many analytics functions.

Databases	**Advantages**	**Disadvantages**
Relational (Ex.: MySQL, Microsoft SQL Server,PostgreSQL)	• Mature. • Standard query language. • ACID compliant. • Easy maintenance.	• Single data model (relational). • Difficulty to scale. • Lower Performance, given the atomicity of the operations.
Key-Value Store (Ex.: Redis, DynamoDB RIAK)	• Simple. • Efficient. • Schema less. • High scalability.	• Low consistency. • Not ACID compliant. • Lack of analytics features like joins and aggregate operations.
Columns-Oriented (Ex.: Big Table, Cassandra, Vertica)	• Very high scalability. • High availability. • Fault tolerance. • Easy maintenance.	• Difficult to manipulate an entire row. • Low insertion rate. • Low update rate.
Document Store (Ex.: MongoDB, CouchDB, RethinkDB)	• Great performance. • High scalability. • Flexible and schemaless. • Documents may have similar and dissimilar data.	• Bad performance for data with many relationships. • Not suitable for normalization. • Lack of the join operation.
Graph (Ex.: Neo4j,OrientDB AllegroGraph)	• Mature. • Optimal for storing connections between data. • ACID compliant. • Highly scalable.	• Difficult to achieve “sharding”. • Not optimized for processing high volumes of transacions. • Not efficient at large-volume analytics queries.
Object Oriented (Ex.: db4o, ObjectDB Perst)	• Object Oriented Programming features. • Fast access to the data. • Makes the software development process more agile. • Optimal for complex object relationships.	• Tied to a specific type of programming language. • Difficult to scale. • Not efficient for simple relationships.

Table 5.1: Advantages and disadvantages of the different types of databases.

5.1.3 Data Characterization Overview

The final dataset, comprising 93 users has 2180302 permissions, from which 65261 were manually answered by the users, with an average of 701.73 answered requests per user.

From the 65261 permissions, 66% were accepted and 33% denied. In figure 5.2 we can observe the grant result distribution per user.

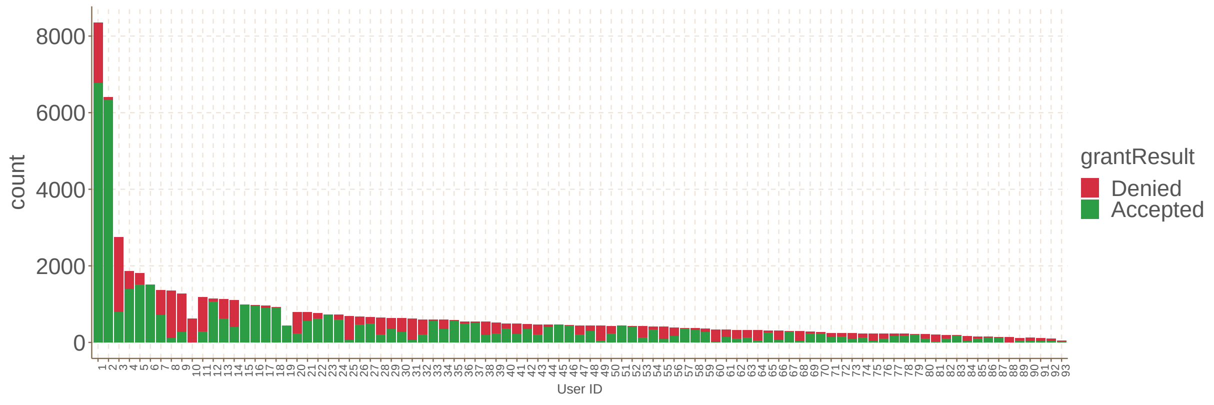


Figure 5.2: Grant result distribution for user answered permissions per user, where the x axis represents a user id.

We can observe that there more users that have more allowed permissions thant denied ones. Besides our users being more permissive, the two users who answered more permissions have accepted more than 80% of their permission requests.

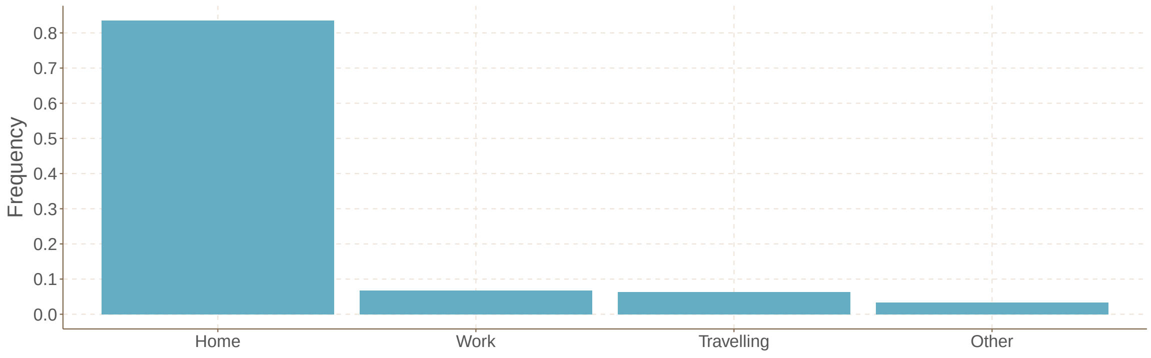
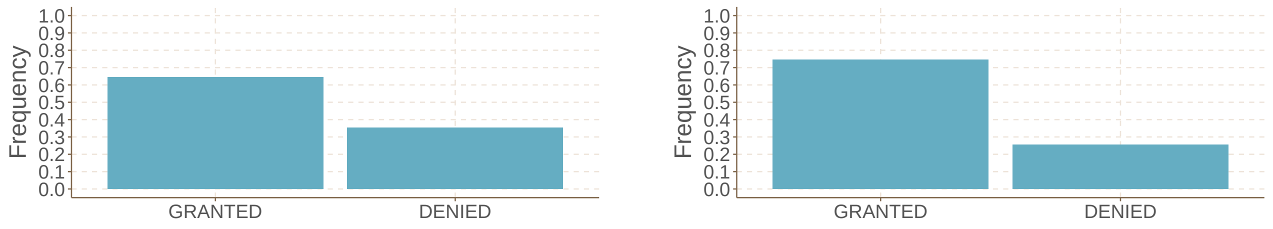
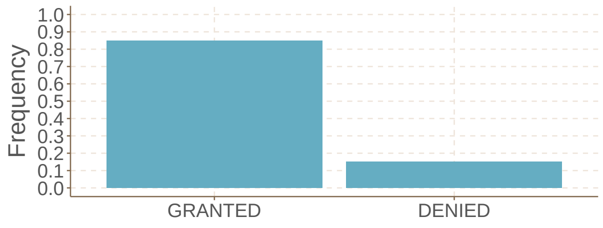


Figure 5.3: Semantic location distribution.

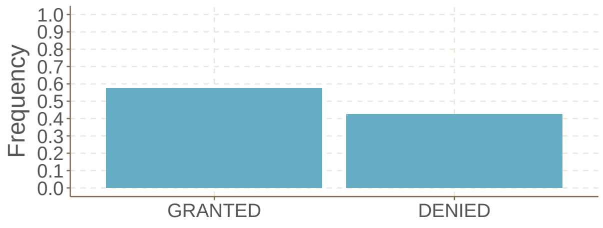
Since some campaigns were done during the COVID-19 confinement, we can observe that more than 80% of the permissions were asked when the users were at home, 6.8% at work, 6.3% while travelling and the remaining at other locations.



(a) Home. (b) Work.



(c) Travelling.



(d) Other.

Figure 5.4: Grant result distribution by semantic location.

Looking at the grant result distribution by the semantic location (figures 5.4a, 5.4b, 5.4c and 5.4d), we can extract interesting conclusions. At work and while travelling, the users tend to allow many more permissions, unlike at other location, where the users deny almost as many permissions as they grant. At home, users tend to grant twice as much as they deny.

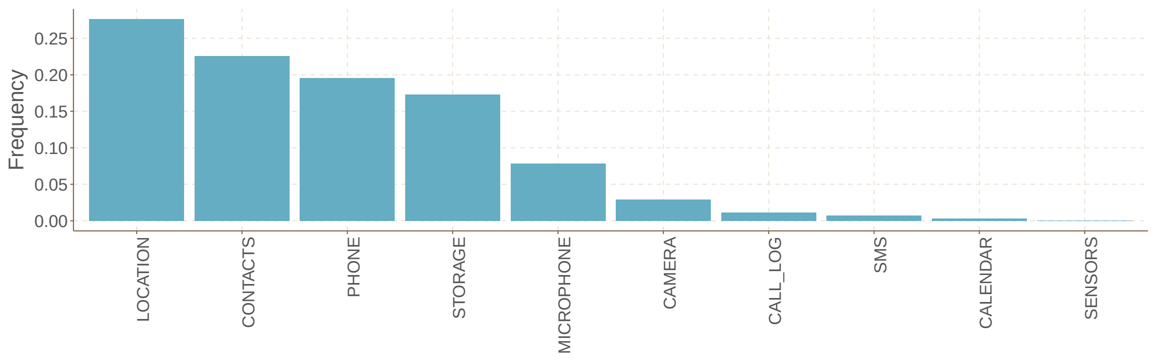


Figure 5.5: Permission distribution.

Currently, Android contains 12 permission groups: `ACTIVITY_RECOGNITION`, `CALENDAR`, `CALL_LOG`, `CAMERA`, `CONTACTS`, `LOCATION`, `MICROPHONE`, `NEARBY_DEVICES`, `PHONE`, `SENSORS`, `SMS`, `STORAGE`. These permission groups comprises a set of permissions, for example, the `SMS` group permission encompasses the permissions `READ_SMS` and `SEND_SMS`. Because the Android default permission manager only requests permissions at the group level, we focus our analysis on permission groups and refer to these simply as permissions. By observing figure 5.5 we conclude that there are two main permissions groups, `LOCATION` and `CONTACTS`, accounting for 50% of the requests. `PHONE` and `STORAGE` permissions account for 37% of the requests.

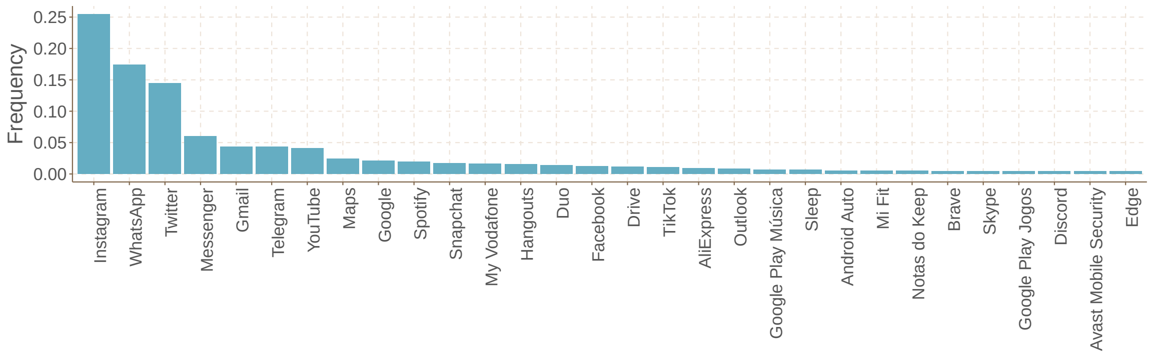


Figure 5.6: Request app name distribution (user answered).

In figure 5.6 we observe the distribution of the applications' names and categories, where social and communication apps account for the top 7 applications, corresponding to 76% of the requests.

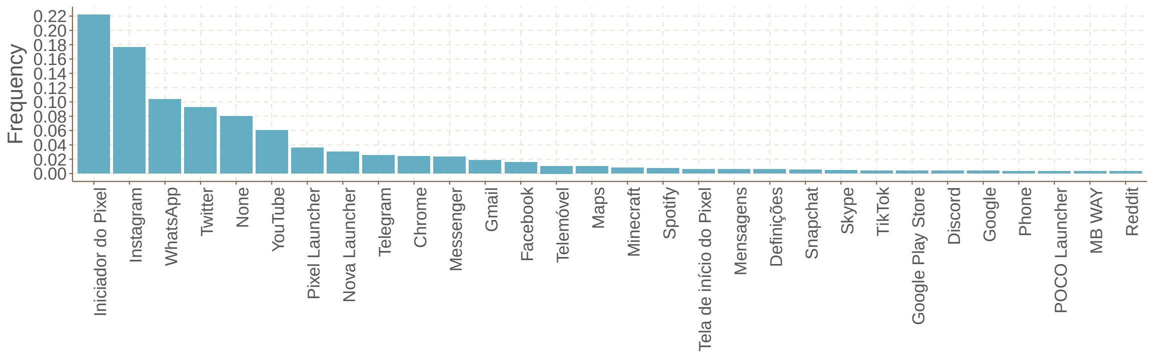


Figure 5.7: Top running app name distribution.

Analysing the top running applications, Instagram, WhatsApp and Twitter are the most frequent, accounting for 37% of the requests, with the exception of the Pixel Launcher (*Iniciador do Pixel*), which is a system app.

Figure 5.8 gives us an overall view of the users tendencies to allow and deny specific app categories and permissions. The app categories were extracted from the Google Play Store[1], from which `MUSIC_AND_AUDIO`, `ENTERTAINMENT`, `SPORTS`, `SHOPPING`, and `GAME` were merged together in the `ENTERTAINMENT` category. For each combination of app category and permission, it is represented the average decision of all users. For example, almost all requests for `CALENDAR` from `ENTERTAINMENT` apps were accepted by the users, and almost all requests for `PHONE` from `PHOTOGRAPHY` apps were denied.

We can observe that the most denied permissions are `MICROPHONE` and `PHONE` and the most denied app categories are `TOOLS`, `VIDEO_PLAYERS`, `FINANCE`, `NEWS_AND_MAGAZINES`. The most accepted permissions are `STORAGE`, `CALENDAR` and `CAMERA` and the most accepted app

[1]https://play.google.com/store (2021, August 17)

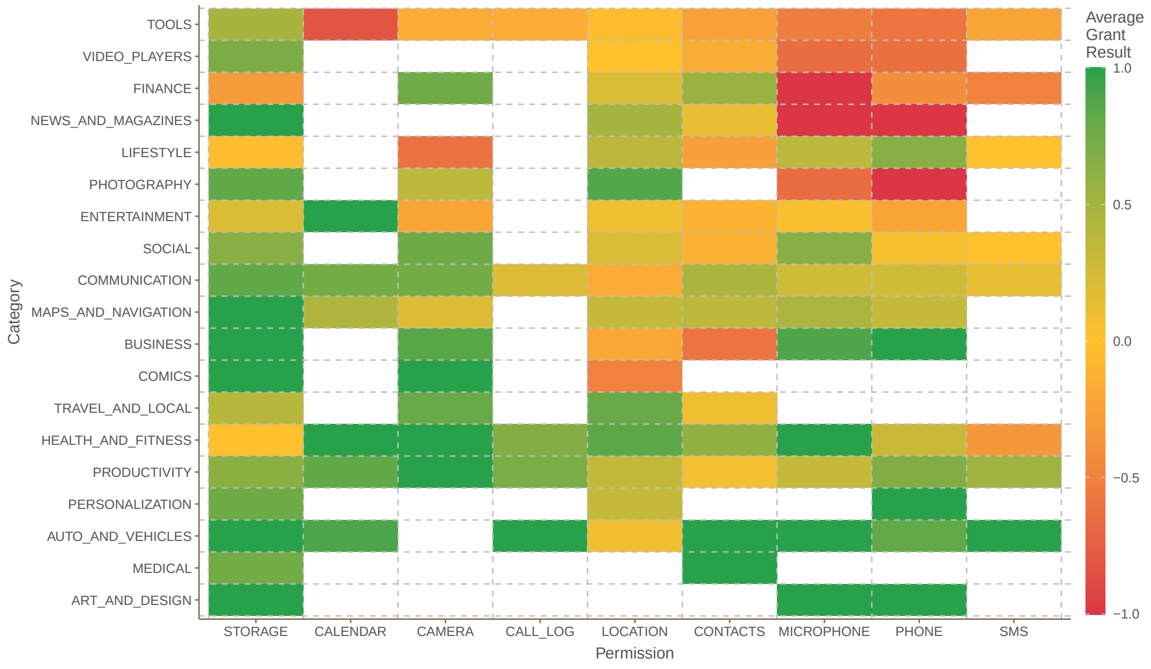


Figure 5.8: Average grant decision for app category and permission.

categories are `ART_AND_DESIGN`, `MEDICAL` and `AUTO_AND_VEHICLES`.

We can also observe that both `LOCATION` and `CONTACTS` permissions have many yellow/orange squares. This indicates that users are not in agreement with each other and/or there are more variables that makes the same user accept and deny the same permission in different contexts, like location for example.

Figure 5.9 show us the users location when they answered the permission requests. We see a very strong presence in Porto/Gaia, and also in Coimbra. Some users were in Guarda, Lisboa, and Algarve as well. Overall, we are looking at mainly northern users.

5.2 Privacy Profiles

As seen in section 2.3 there are many possible methods to generate profiles. In this section, we design strategies to generate privacy profiles with the COP-MODE data, using hierarchical clustering (section 5.2.1) and k-Means (section 5.2.2). Then, using the algorithms presented in section 3.1, we devise a strategy to apply them to the COP-MODE data, generating privacy profiles in a secure and private manner (sections 5.2.1.1 and 5.2.2.1).

We need to generate privacy profiles so we can increase the effectiveness of the users' grant results predictions. Liu *et. al* [18] demonstrated that a profile-based optimized model achieves far better results than a model without privacy-profiles. So, our goal in this chapter is to describe how we can use the algorithms described in section 2.3 to generate these privacy profiles in a secure way.

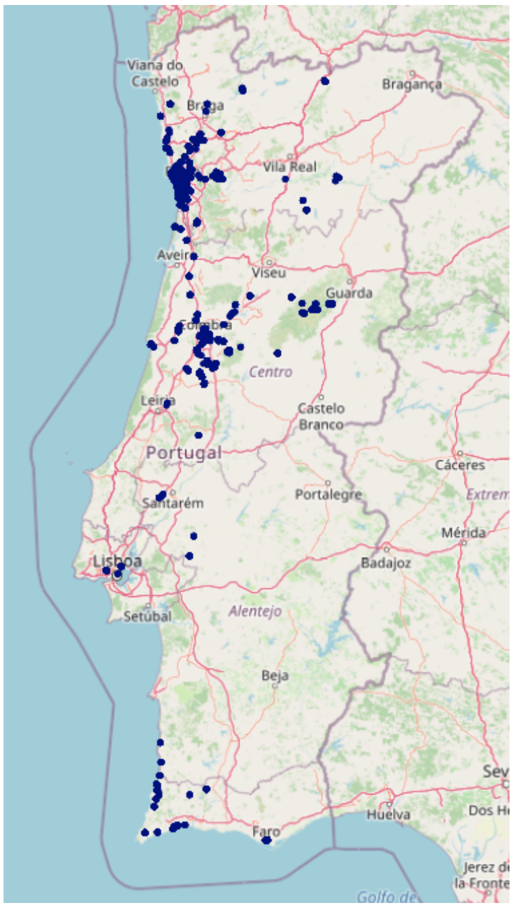


Figure 5.9: Users location, where each point represents a permission request.

5.2.1 Hierarchical Clustering

As seen in sections 2.3 and 3.1, one of the possibility towards the generation of user profiles is using the hierarchical clustering algorithm [18]. In order to apply this strategy to our dataset, for each user answered permission we group the data by `userID`, `category` and `permission`, with the average of the `grantResult`. So, for each user, we have the average `grantResult` for every combination of `category` and `permission`. However, most users do not have `grantResults` for all these possible combinations. So, these data points are filled by a multivariate imputer, where the `grantResult` is modeled as a polynomial function of all the remaining features. We chose this imputation strategy for being more sophisticated than the univariate imputation strategies. We used the `IterativeImputer` class from the Scikit-Learn package.

After the imputation, all the `category`, `permission` tuples are flattened in the same row, *i.e.* for each user we will have one column for every `category`, `permission` combination and its value is the average `grantResult` (table 5.2). With the resulting matrix, we can build a dendrogram and use the agglomerative clustering algorithm to generate the privacy profiles, as explained in section 2.2.2.2.

EVENTS CALENDAR	EVENTS CAMERA	…	AUTO_AND_VEHICLES PHONE	AUTO_AND_VEHICLES CONTACTS
0.9	0	…	0	0
0.2	0.1	…	0.35	0.4
		⋮		
0.6	0.2	…	0.15	0.2

Table 5.2: Matrix representation, where each row represents one user (grant result normalized between 0 and 1.

In figure 5.10 we can observe the resulting dendrogram and privacy profiles for 3 clusters represented in figures 5.11a, 5.11b and 5.11c.

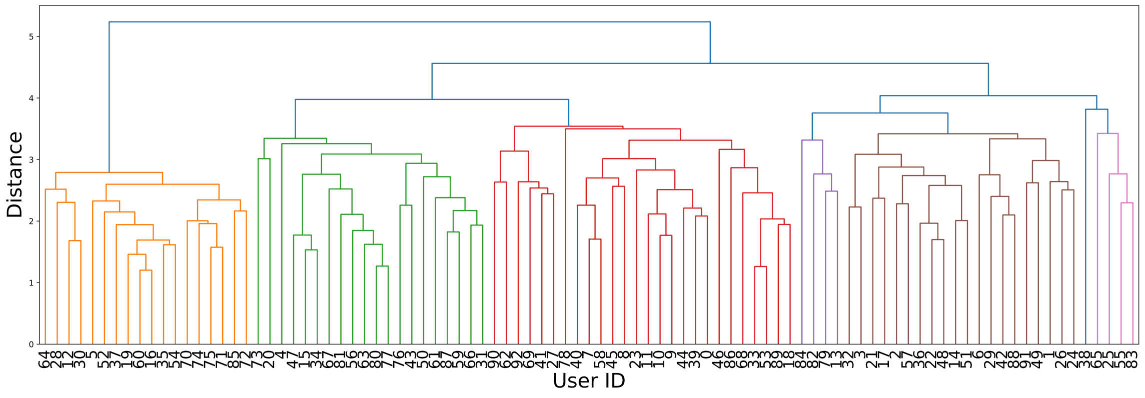
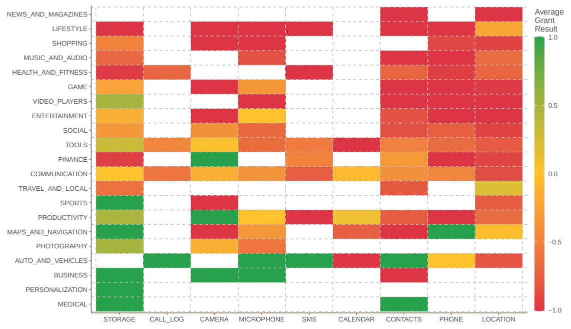
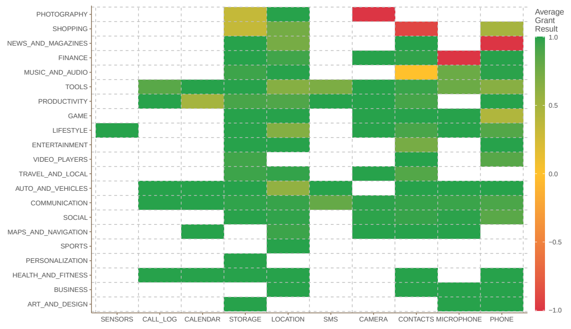


Figure 5.10: Resulting dendogram.

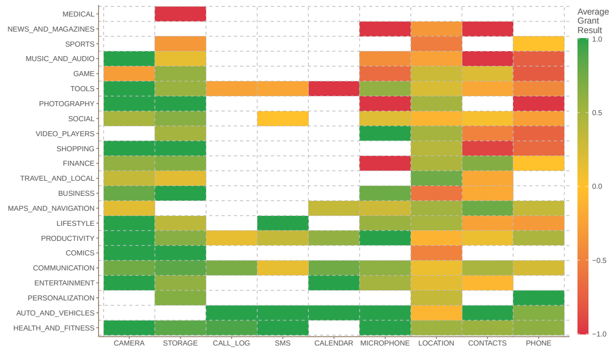
The dendrogram is a diagram representing the arrangement of the clusters produced by the hierarchical clustering algorithm. In the x-axis we have each user represented by a number. In order to create the clusters, for demonstration purposes, we used k equal to 3, so one cluster will contain the yellow users, another cluster will contain the green and the red, and the final cluster will contain the remaining users. In figures 5.11a, 5.11b and 5.11c, we observe the resulting profiles for those three clusters, and the average `grantResult` for each category, permission combination. For example, in profile 1 (figure 5.11a), the users tend to deny most of the permissions on most of the categories, in profile 2 (figure 5.11b), the opposite happens, users accept almost always all permissions related to all categories, with the exception of (`CAMERA`, `PHOTOGRAPHY`), (`MICROPHONE`, `FINANCE`) and (`PHONE`, `NEWS_AND_MAGAZINES`). Finally, in profile 3 (figure 5.11c), we observe less consistent behaviour, with many yellow/orange rectangles, meaning the users allow and deny that permission/category combinations more or less the same number of times.



(a) Profile 1.



(b) Profile 2.



(c) Profile 3.

Figure 5.11: Privacy profiles. (a) Profile of the privacy conscious. (b) Profile of the permissive users. (c) Profile of the "middle-ground" users.

5.2.1.1 Privacy Preserving Distributed Hierarchical Clustering

To generate privacy profiles using hierarchical clustering with privacy guarantees, we can use the privacy preserving distributed hierarchical clustering algorithm (section 3.4).

In this context, each user has only access to their own local dataset. So, the process referred above (section 5.2.1) needs to be applied independently to each local dataset. This will cause problems in the imputation step if our imputation algorithm needs access to all the users' data. So, our imputation strategy must rely solely on the local data or global statistics that can be acquired using the secure aggregation strategy.

For that, we perform imputation by using the `IterativeImputer` class from the Scikit-Learn Python package [2] and the flattening process, each user will have a single vector, since the process generates only one row per user (section 5.2.1). This vector will then be used to generate the profiles using the privacy preserving distributed hierarchical clustering algorithm.

[2]https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html, (17, August 2021)

5.2.2 k-Means

Another strategy discussed in sections 2.3 and 3.1 is to use k-Means to generate multiple profiles for each user [30].

To apply this algorithm to our data, we also group the data by `userID`, `category` and `permission`, with the average of the `grantResult`, for each user. But this time, each user is represented by multiple rows of `category`, `permission` and `grantResult` (table 5.3). Unlike the strategy in section 5.2.1, we do not need to flatten the users data in one row, so we also do not need to impute the missing combinations of `category` and `permission`. This is a clear advantage of this method, since it removes the added bias from the imputation. We can feed this data directly to the k-Means algorithm and generate the privacy profiles, where each user will have data points in one or more profiles.

UserID	Category	Permission	Average Grant Result
1	FINANCE	CALENDAR	0.90
2	FINANCE	CALENDAR	0.20
3	FINANCE	CALENDAR	0.60
1	FINANCE	CAMERA	0.00
2	FINANCE	CAMERA	0.10
3	FINANCE	CAMERA	0.20
	⋮		
1	AUTO_AND_VEHICLES	PHONE	0.00
2	AUTO_AND_VEHICLES	PHONE	0.35
3	AUTO_AND_VEHICLES	PHONE	0.15
1	AUTO_AND_VEHICLES	CONTACTS	0.00
2	AUTO_AND_VEHICLES	CONTACTS	0.40
3	AUTO_AND_VEHICLES	CONTACTS	0.20

Table 5.3: Matrix representation, where one user is represented by one or more rows.

However, the profile representation is less intuitive. In order to represent the users and keep as much information as possible, we decided to associate a percentage of each profile to every user. For example, if a user has 10 data points in profile 1, 30 in profile 2 and 60 in profile 3 on a total of 3 profiles, this user will be represented as $[0.1, 0.3, 0.6]$, instead of being represented as $[0, 0, 1]$, for instance. This representation is needed in order to use the profiles' information to predict the `grantResult` in section 5.3.

5.2.2.1 Efficient Privacy Preserving Distributed k-Means

To generate privacy profiles with privacy guarantees, using the k-Means algorithm, we can use the efficient privacy preserving distributed k-Means algorithm (section 3.3).

The exact same pre-processing will be applied here, but now independently on every local dataset. Since there is no imputation step, or any other step that involves other users' data, there is no bottleneck. The resulting matrix for each user will be directly feed into the efficient privacy preserving distributed k-Means algorithm.

In the end, each user will have the association between every local data point and the respective profile. With this information, the user can extract the profile representation described in section 5.2.2.

5.3 Neural Network for Grant Prediction

In this section we describe how to use a neural network to predict the users' answers to a permission request. Our final goal is to evaluate the results of federated learning. Since federated learning uses neural networks, we need to evaluate the results of a centralized neural network in order to understand how the federated component affects the results. With the data collected from the users and the assignment to profiles, we can predict the `grantResult` for a given permission request.

To use the data we collected from the users as input in the machine learning model, we scale the data using `MinMaxScaler`, which scales the data points to a range between 0 and 1. We also applied one-hot encoding to the dataset, so each categorical variable is represented by a vector of 0s and 1s. The following features are used to predict the `grantResult`:

- `app_category`
- `checkedPermissionGroup`
- `checkedPermission`
- `method`
- `hour`
- `weekday`
- `isForeground`
- `isTopAppRequestingApp`
- `screenIsInteractive`
- `networkStatus`
- `profile`

These are the non-unique features collected by the permission manager, *i.e.* the features that have repeating values, unlike `ID` like features. The `selectedSemanticLoc` and `wasRequestExpected` were removed, since they require user interaction. The timestamp was transformed into hour and weekday, since the timestamp by itself is unique.

The `profile` feature is represented as a one-hot encoding vector, in the profiles generated by the hierarchical clustering algorithms of the previous sections 5.2.1 and 5.2.1.1. For example, a user in profile 3 of a total of 5 profiles would be represented as (user 1 in table 5.4). In case the profiles are generated by the k-Means algorithm, the profiles are represented as percentage-wise, where we still have a column for each profile, but instead of 0s and 1s we have the percentage of

User	**Profile 1**	**Profile 2**	**Profile 3**	**Profile 4**	**Profile 5**
user 1	0	0	1	0	0
user 2	0.1	0	0.7	0.2	0

Table 5.4: Possible representations for the profile feature.

data points per profile (user 2 in table 5.4).

With the final dataset, we trained a neural network with 1 hidden layer with 100 neurons. In order to not overload the validation phase with too many hyper-parameters, we designed a simple test where we created Neural Networks with a hidden layer of sizes 50, 100, 150, 200, 250, 300, and 500 and tested in the fixed scenario, where we used hierarchical clustering with 3 clusters. The results showed that a hidden layer of size 100 achieved the best results, although the changes in performance observed, where not very significant.

Figure 5.12 represents the entire process in a diagram. The first step is to generate the privacy profiles using one of the methods described in section 5.2, after the clustering algorithm, the results are added to the local dataset, *i.e.* for each row in the dataset, we add the respective profile ID. Then, one-hot encoding is applied to the dataset, and a division of 66% for training and 33% for testing is applied. This strategy is called centralized training because the next step is for every client to send their corresponding training data to a centralized server, where a Neural Network will be trained, and the resulting weights will be sent back to the clients. Afterwards, each client will create a Neural Network with the received weights and use the test set to evaluate the model's performance, finally it will send the results to the server.

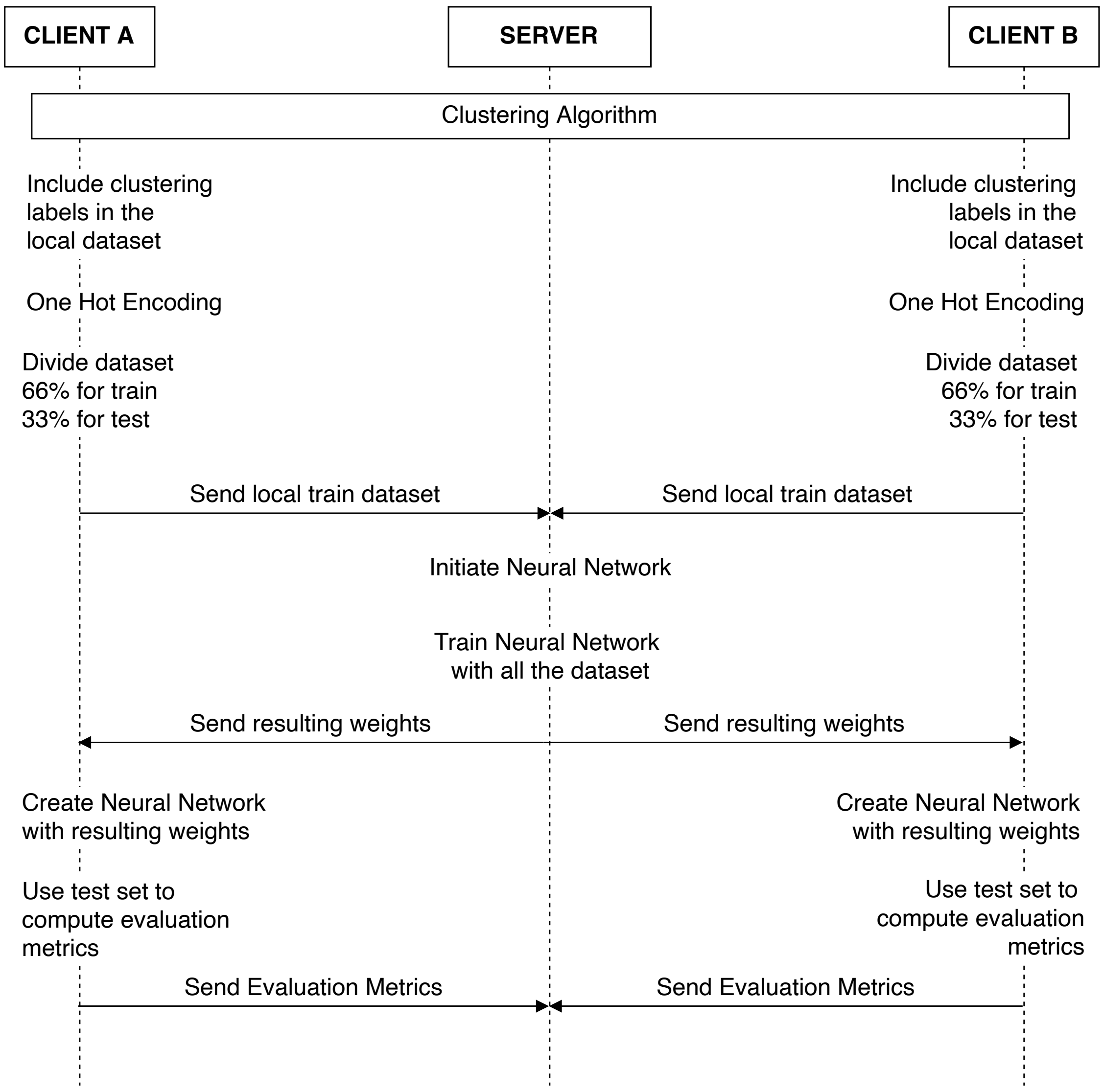

CLIENT A
SERVER
CLIENT B
Clustering Algorithm
Include clustering labels in the local dataset
Include clustering labels in the local dataset
One Hot Encoding
One Hot Encoding
Divide dataset 66% for train 33% for test
Divide dataset 66% for train 33% for test
Send local train dataset
Send local train dataset
Initiate Neural Network
Train Neural Network with all the dataset
Send resulting weights
Send resulting weights
Create Neural Network with resulting weights
Create Neural Network with resulting weights
Use test set to compute evaluation metrics
Use test set to compute evaluation metrics
Send Evaluation Metrics
Send Evaluation Metrics

Figure 5.12: Centralized training diagram.

We applied this process, using hierarchical clustering with 3 clusters, and the results obtained are summarized in figures 5.13, 5.14, and 5.15.

It is clear from these figures that the resulting model does not work very well for some users, more specifically, the ones with lower percentage of granted permissions in figure 5.14, *i.e* the ones who deny most of the requests. It is also possible to observe the advantages of using multiple metrics, while the accuracy metric has a very high score to these users, the F1-score and the precision-recall AUC do not, indicating that the model is always denying the permissions. In figure 5.15 we observe the boxplot of scores for each metric, the median score is always above 0.8, with 75% of the points being above 0.7 for all metrics. However there are some points with very low scores, bellow 0.4, mainly the F1-scores. In order to reduce the bias of the model against users with lower percentage of granted permissions, we decided to oversample each local training dataset after dividing it, obtaining 50% training points with granted permissions, and 50% with denied permissions. This way, the trained model will be less biased, yet the test set will have a

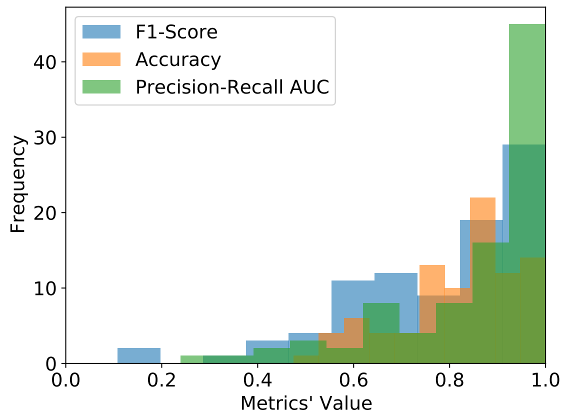


Figure 5.13: Histogram of scores.

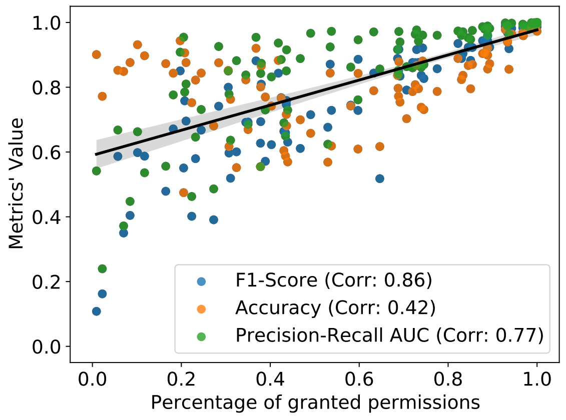


Figure 5.14: Scores vs. percentage of granted permissions.

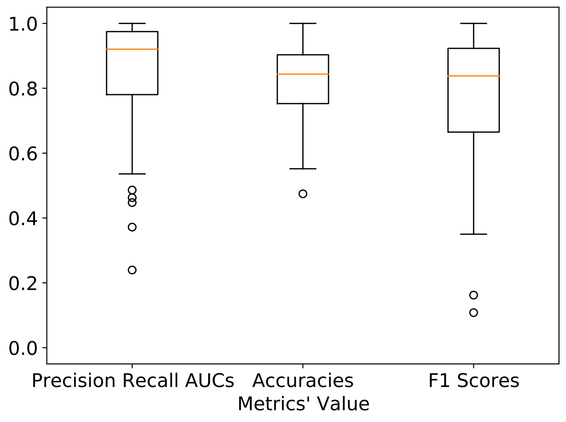


Figure 5.15: Boxplot of scores by metric.

realistic percentage of granted permissions, since it is not oversampled.

The results are summarized again in figures 5.16, 5.17 and 5.18. From these results we observe a reduction in the bias against users with lower percentage of granted permissions, improving the overall scores. From the boxplot is also very clear that there was a reduction in lower scores in all metrics, even tough the median score remained more or less the same.

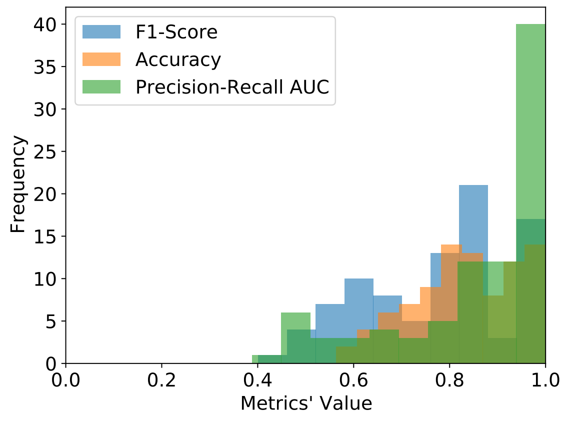


Figure 5.16: Histogram of scores.

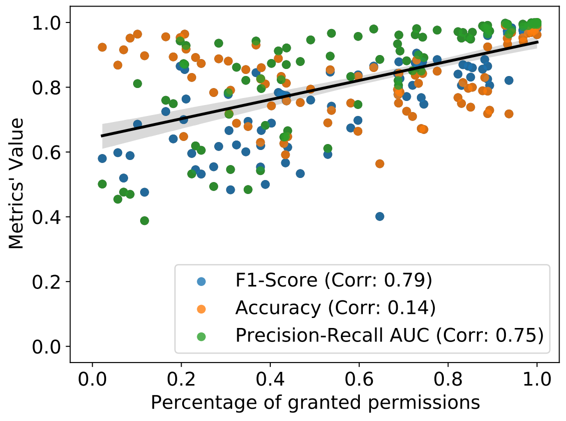


Figure 5.17: Scores vs. percentage of granted permissions.

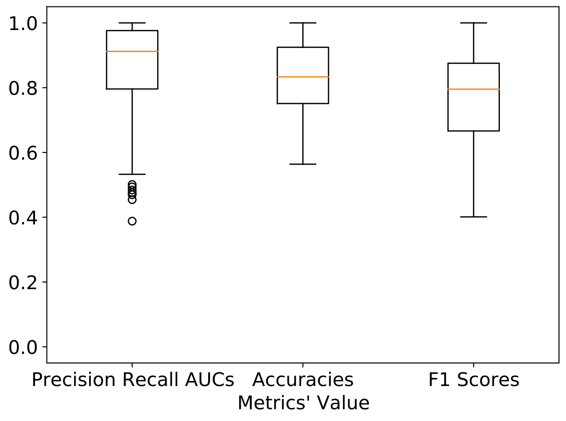


Figure 5.18: Boxplot of scores.

5.3.1 Federated Learning with Neural Networks for Grant Prediction

While neural networks can be used to predict the users' answers, we want a strategy to evaluate the privacy profiles with privacy guarantees. Federate learning provides us that possibility by training a model locally, on each smartphone, using only local data, and then sharing the neural networks' weights with a central server on each iteration. The central server averages the weights and returns the result to the clients, so they can use these new weights to continue the training. Here, the same processing referred in section 5.3 is done locally, using the same features. Figure 5.19 represents the entire process in a diagram.

The first 4 steps are exactly the same as the ones presented in figure 5.12, the oversampling of the training dataset, in order to have the same number of rows with granted permissions and denied permissions, was also described before. Now, instead of sending the training data to a centralized server, the centralized server creates a Neural Network and shares the weights with all the clients. Next, each client initiates a Neural Network with the received weights. Each client will iteratively train the network and send the local weights to the server, the server will average all the local weights and send back the results. Each client will set the received weights on their local Neural Network. This process is executed until convergence, when the average of all local weights is equal to the one in the previous iteration or when a maximum number of iterations is reached. With the test set, each user tests the model performance and sends the results to the server.

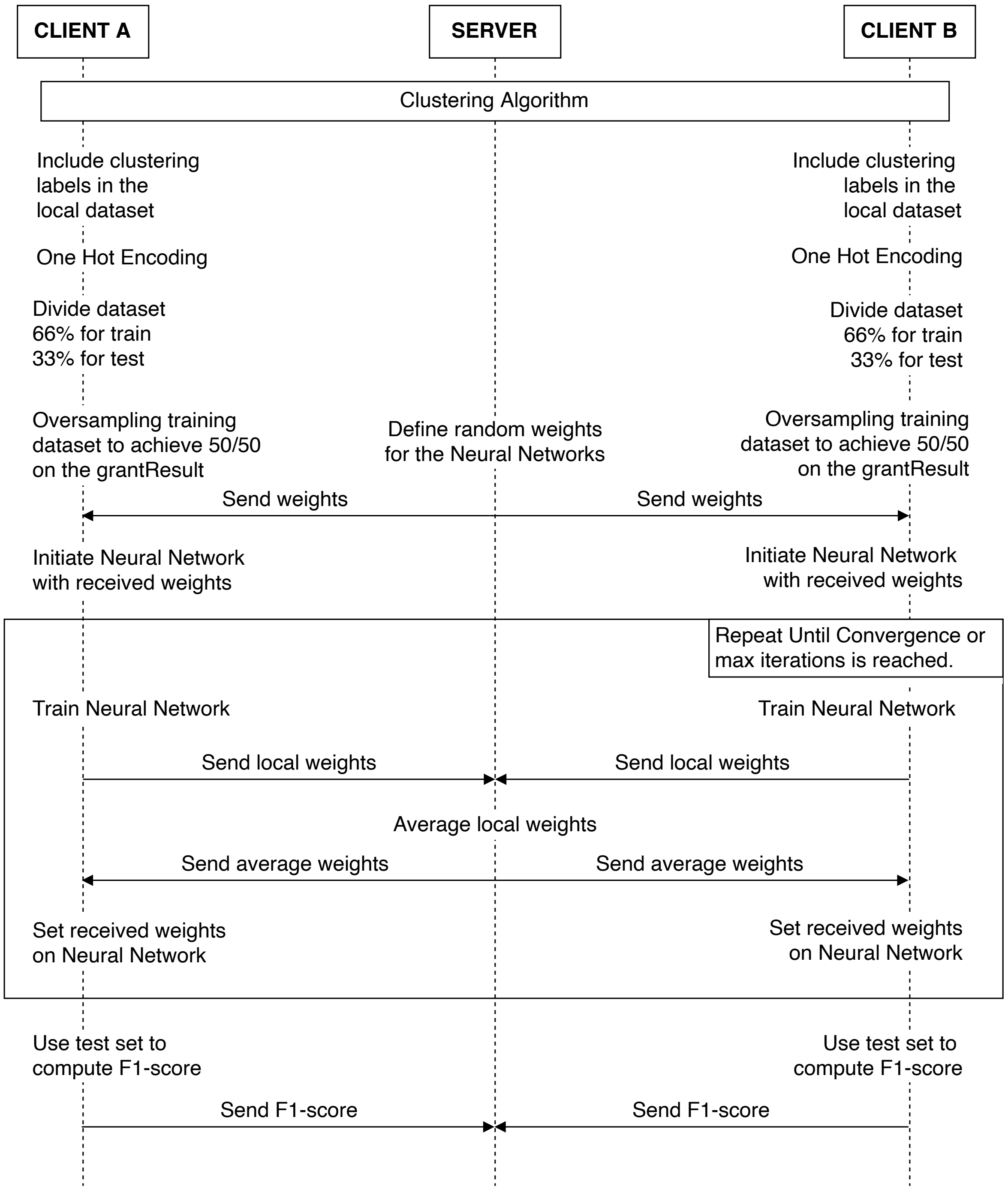

CLIENT A
SERVER
CLIENT B
Clustering Algorithm
Include clustering labels in the local dataset
Include clustering labels in the local dataset
One Hot Encoding
One Hot Encoding
Divide dataset 66% for train 33% for test
Divide dataset 66% for train 33% for test
Oversampling training dataset to achieve 50/50 on the grantResult
Define random weights for the Neural Networks
Oversampling training dataset to achieve 50/50 on the grantResult
Send weights
Send weights
Initiate Neural Network with received weights
Initiate Neural Network with received weights
Repeat Until Convergence or max iterations is reached.
Train Neural Network
Train Neural Network
Send local weights
Send local weights
Average local weights
Send average weights
Send average weights
Set received weights on Neural Network
Set received weights on Neural Network
Use test set to compute F1-score
Use test set to compute F1-score
Send F1-score
Send F1-score

Figure 5.19: Federated learning training diagram.

We applied this process, using hierarchical clustering with 3 clusters, and oversampling the training data as explained before, and the results obtained are summarized in figures 5.20, 5.21, and 5.22.

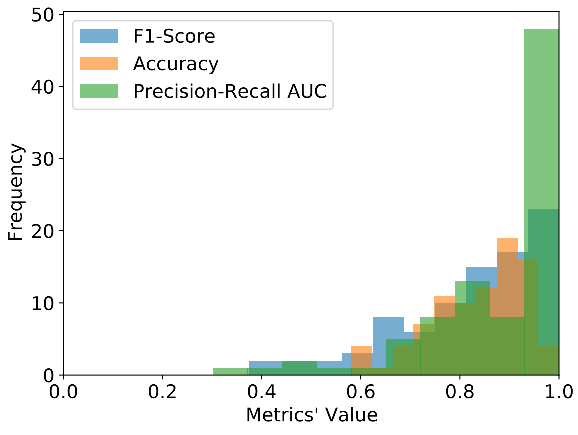


Figure 5.20: Histogram of scores.

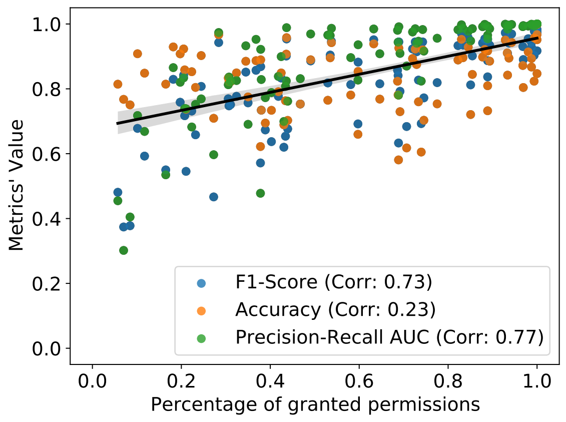


Figure 5.21: Scores vs. percentage of granted permissions.

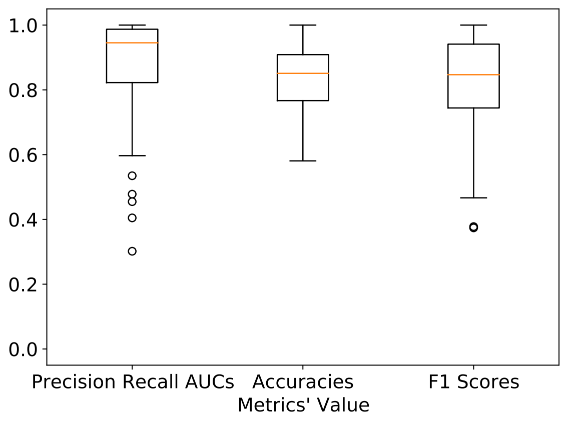


Figure 5.22: Boxplot of scores.

Looking at the results, we still observe some bias in the model, specially for users with more than 90% of denied permissions or, in order words, less than 10% percentage of granted permissions. However, the other users have even better results than the ones obtained before, using the centralized neural network. From these results we can conclude that not only it is possible to predicting the grant result with privacy guarantees, doing so leads to comparable or even better results than we using a centralized approach.

5.4 Evaluation

The previous section demonstrated the feasibility of predicting the grant result using federated learning. However, the profile assignment was still done in a centralized fashion, thus requiring the data of the users to go to a centralized server. In this section we assess the feasibility and performance of building profiles using the privacy-preserving strategies described in sections 5.2.2.1 and 5.2.1.1. However, because there is no ground truth for the users' privacy profiles, it is hard to evaluate them. So, in order to evaluate the utility of the generated profiles, we can use the grant prediction results.

Using the strategies described before, we can use the grant prediction evaluation metrics to compare the usefulness of the different profiles. If the neural network is capable of achieving a better score using a specific set of profiles, we consider them to be more useful than another other set, that achieves a lower score.

We use three metrics to evaluate the models' performance, F1-Score, Accuracy, and Precision-Recall AUC (Area Under the Curve). The F1-Score is the harmonic mean of the precision and recall, and we use it because it takes both false positives and false negatives into account, and this is also the metric used in previous works. We also use Accuracy, given the uneven class distribution in our dataset, the F1-Score is not always possible to calculate, *e.g.* when the dataset only contains one class, this way the accuracy can give us an idea of how the model is behaving. Finally, the Precision-Recall AUC summarizes the Precision-Recall curve, and can be used to understand the trade-off in performance for different threshold values when interpreting probabilistic predictions. For the evaluation of the performance, we divided the dataset in 80% for the validation and 20% for testing, as described in section 5.4.1 and section 5.4.2, respectively.

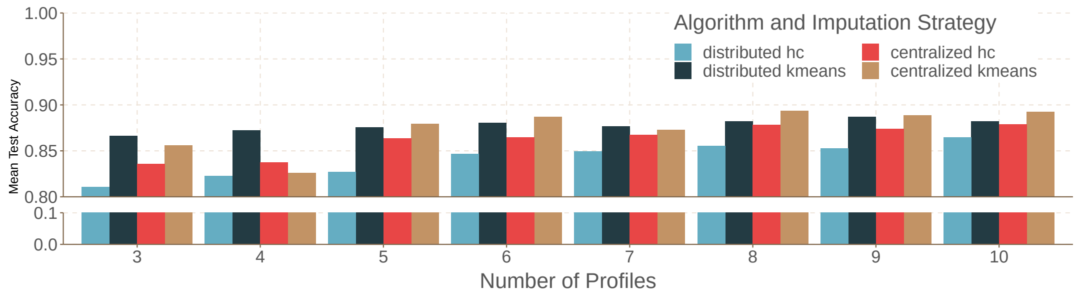
5.4.1 Validation

To find the best set of profiles in our dataset we preformed a grid search on the following parameters:

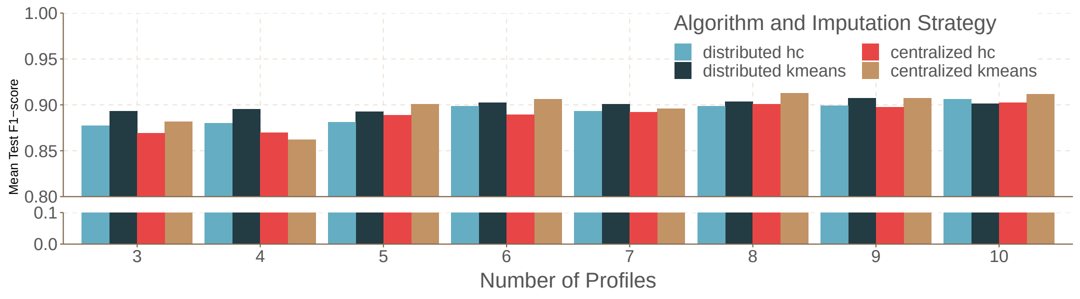
- Clustering Algorithm.
- Imputation Method.
- Number of Clusters.

The clustering algorithms consist of hierarchical clustering (**hc centralized**), privacy preserving distributed hierarchical clustering (**distributed hc**), k-means (**centralized k-means**) and efficient privacy preserving distributed k-means (**distributed k-means**). The imputation method used was the multivariate imputer, that estimates each feature from all the others. But applied in different ways, one globally, using all the users data together (not possible in a private distributed manner) and applied locally, each user applies the imputation locally to their dataset.

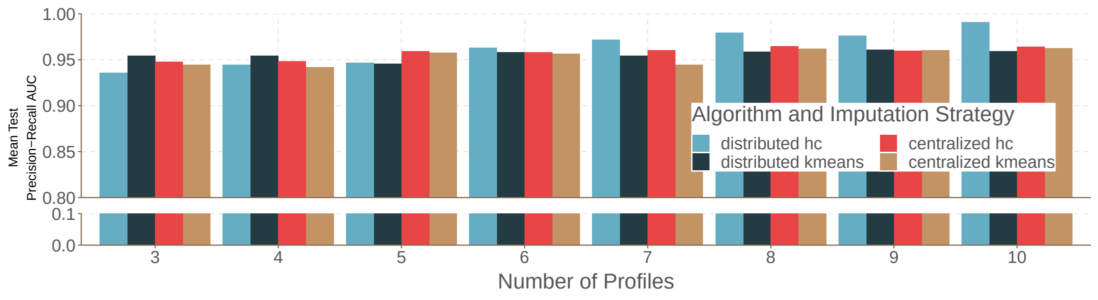
For every combination we applied a 5-fold cross validation, using 80% of the dataset, leaving 20% for testing. We used the centralized neural network and federated learning, since we want to evaluate the utility of the profiles in a secure way as well. In figures 5.23a, 5.23b and 5.23c we can observe the results obtained for three metrics, accuracy, F1-score and precision-recall AUC with centralized neural networks.



(a) Mean accuracy 5-fold cross validation results.



(b) Mean F1-score 5-fold cross validation results.



(c) Mean precision-recall AUC 5-fold cross validation results.

Figure 5.23: Results obtained for the centralized neural networks.

We can observe that the lowest achieved accuracy was 0.81 with distributed hierarchical clustering with 3 profiles, and the highest was 0.89 with k-means centralized with 8 profiles. The distributed hierarchical clustering always underperforms all the other strategies, and the best strategies are always k-means (centralized or distributed).

Looking at the F1-scores (figure 5.23b), the lowest score is 0.86 with k-means centralized using 4 profiles, and the highest one is 0.91 with k-means centralized with 8 profiles. These results are more inconsistent, with a low number of profiles (3 and 4), the distributed k-means outperforms the other strategies. However, when the number of profiles is greater than 4, k-means centralized usually outperforms the other strategies, with distributed k-means in second.

Finally, the precision-recall AUC scores present a more optimistic and consistent view, with the lowest score being 0.93 for distributed hierarchical clustering using 3 profiles, and the highest being 0.99 for distributed hierarchical clustering using 10 profiles.

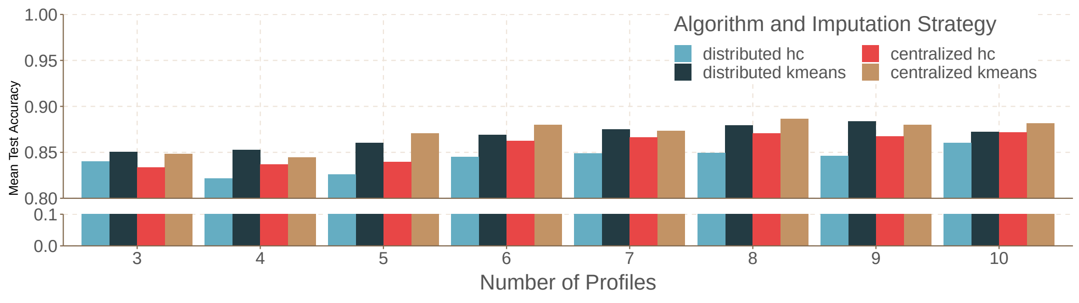
This set of metrics allow us to conclude that using privacy preserving strategies for generating privacy profiles does not affect the usefulness of the resulting profiles. With this in mind, we can use this strategies and compute their usefulness using federated learning and compare the results to understand if it is possible to generate and evaluate privacy profiles in a fully distributed scenario.

In figures 5.24a, 5.24b and 5.24c we can observe the results obtained for three metrics, accuracy, F1-score and precision-recall AUC with federated learning.

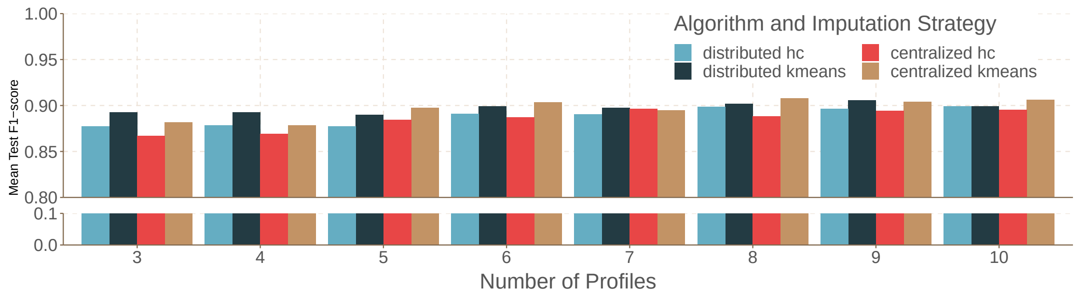
With federated learning, we are capable of achieving similar results to the ones with centralized neural networks. We can observe that the lowest accuracy is 0.82 with distributed hierarchical clustering with 4 profiles, and the highest is 0.88 with distributed k-means with 9 profiles. The distributed hierarchical clustering still underperforms almost all the other strategies, and the best strategies are always k-means (centralized or distributed) as well.

Looking at the F1-scores (figure 5.24b), the lowest score is 0.87 with distributed hierarchical clustering using 3 profiles, and the highest one is 0.91 with distributed k-means with 9 profiles. The k-means algorithm, both centralized and distributed, continue to outperform most of the strategies.

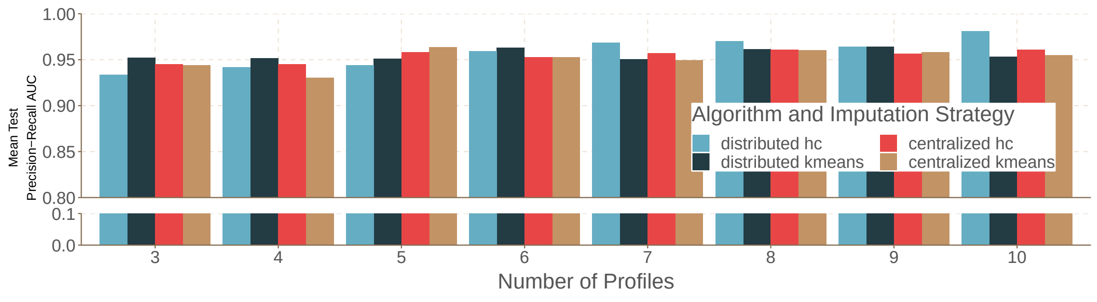
Finally, the precision-recall AUC scores (figure 5.24c) have a lowest score of 0.93 for distributed hierarchical clustering using 3 profiles, and the highest being 0.98 for distributed hierarchical clustering using 10 profiles.



(a) Mean accuracy 5-fold cross validation results with federated learning.



(b) Mean F1-score 5-fold cross validation results with federated learning.



(c) Mean precision-recall AUC 5-fold cross validation results with federated learning.

Figure 5.24: Results obtained for the federated learning approach.

Overall, and since we want to compare our results to the ones in previous works, the best secure model is the distributed k-means using 9 profiles, both for centralized neural networks and federated learning (tables 5.5 and 5.6). This demonstrates that our strategy can be used in a fully distributed scenario, where both the profiles' generation and evaluation are done in a private distributed manner.

	Centralized Neural Network					
	Accuracy		F1-Score		PR-AUC	
Best	0.89	$(k = 3)$	0.91	$(k = 8)$	0.99	$(k = 10)$
	Centralized	k-Means	Centralized	k-Means	Distributed	HC
Worst	0.87	$(k = 3)$	0.77	$(k = 5)$	0.94	$(k = 3)$
	Distributed	HC	Distributed	k-Means	Distributed	HC

Table 5.5: Best and worst results for centralized neural network.

	Federated Learning					
	Accuracy		F1-Score		PR-AUC	
Best	0.88	$(k = 9)$	0.91	$(k = 9)$	0.98	$(k = 10)$
	Distributed	k-Means	Distributed	k-Means	Distributed	HC
Worst	0.82	$(k = 4)$	0.87	$(k = 3)$	0.93	$(k = 3)$
	Distributed	HC	Distributed	HC	Distributed	HC

Table 5.6: Best and worst results for federated learning.

5.4.2 Testing

In order to test the best models found in the validation phase, each user will train the models using the data from validation, 80% of their dataset, and test it with the remaining 20%. The F1-score, accuracy and precision-recall AUC metrics for each user is presented in figures 5.25 and 5.27 and a scatter plot with metrics vs. percentage of granted permission is presented to identify any bias in the model is presented in figures 5.26 and 5.28.

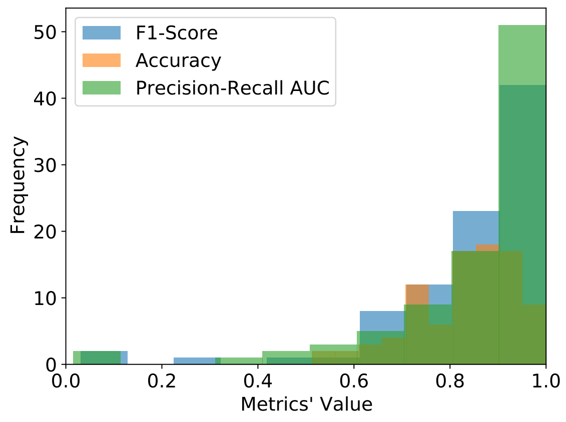


Figure 5.25: Histogram of scores for centralized neural network.

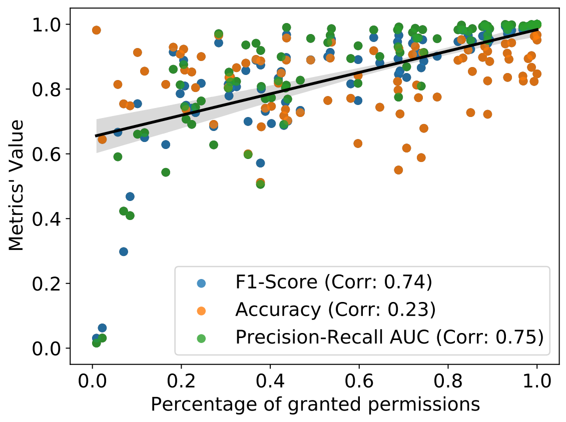


Figure 5.26: Scores vs. percentage of granted permissions for centralized neural network.

For the centralized neural network, the global F1-score was 0.91, the global accuracy was 0.90 and the global precision-recall AUC was 0.97. Figure 5.26 show us that the model is still somewhat biased, with users with a high percentage of granted permissions having consistently higher scores than the ones with a lower percentage. With the exception of a few users, where the neural network underperforms the random model, the results are good and similar to the ones obtained by Bin Liu [18], where a linear-kernel support vector machine using hierarchical clustering to generate the privacy profiles achieved a cross-validated F1-Score of 90.02%.

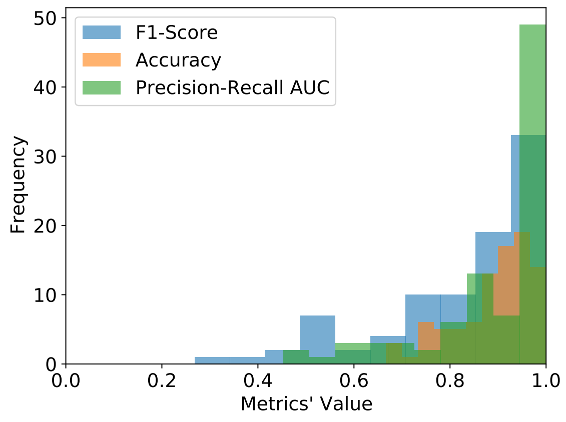


Figure 5.27: Histogram of scores for federated learning.

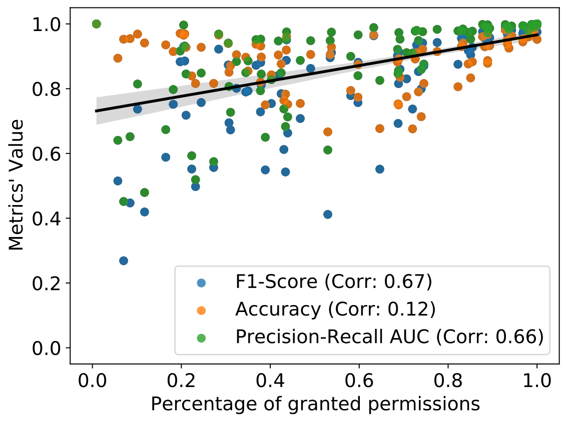


Figure 5.28: Scores vs. percentage of granted permissions for federated learning.

For the federated learning, the global F1-score was 0.90, the global accuracy was 0.88 and the global precision-recall AUC was 0.97. When comparing the results from the federated learning displayed in 5.27 and 5.28, to the centralized approach from figures 5.25 and 5.26, we can see that federated learning achieved a lower bias. This is evidenced by the more uniform distribution on the y-axis throughout the x-axis. Overall, these federated learning results are positive and comparable to the centralized results obtained by Liu, Lin and Sadeh [18], with the benefit of ensuring user privacy.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis we present methods for generating privacy profiles and using these to predict users' privacy preferences with privacy guarantees. For that, we developed a new clustering technique to generate privacy profiles using k-means in a efficient, private way that is also robust to non-IID data. For this strategy, each client computes the k-Means algorithm locally. The server will then use the resultant centroids from each client to apply the k-Means algorithm again, generating the global centroids. To maintain the client's privacy, homomorphic encryption is used to compute the distance matrix in a secure way and secure aggregation is applied to share the sum and number of points per cluster. This way, the server never sees the original data, only ciphertexts and aggregated data.

Using privacy-preserving clustering techniques, such as the one described above, we used the resulting clusters to generate privacy profiles. Using these profiles, together with context features, we can train a neural network, without sharing the clients' data, using federated learning. In each iteration, every client trains a neural network locally on their private data and only the resulting weights are shared with the central server, which, in turn, will return the weights average to the clients. This strategy can outperform the centralized hierarchical clustering approach used in the state-of-the-art strategy.

Using the resulting profiles and federated learning we were able to demonstrate the usefulness of the privacy profiles in predicting the users' grant decision in a secure fashion, *i.e.* without sharing users data. Moreover, our secure strategy achieved an F1-score of 90%, matching state-of-the-art performance for privacy decisions prediction. In conclusion, our strategy can be applied in the real world, where not only we are able to generate the profiles, but test them as well, while preserving users' privacy and maintaining a state-of-the-art performance.

6.2 Future Work

In the future, to improve the robustness and applicability of our strategy, data from broader demographics should be collected. This data would allow us to analyse more diverse responses to different app categories and permissions, thus testing the robustness of our strategy.

We presented an approach to securely generate privacy profiles and also test their usefulness, but in order to apply our strategy to real world scenarios, we need to be able to monitor and re-train the model in a secure way as well. We need to build a framework capable of monitoring the model's performance in a secure way, together with the ability to update the profiles with the new data in a secure way as well. Such framework would improve the adaptability of the model to real world scenarios, and thus incorporate potential changes in privacy preferences.

The final strategy, presented by us, to predict the users' grant decision, is complex, including two learning phases: one for privacy profile generation, another for using the profiles and context variables to predict the users' answers. For future work, a deep learning approach together with federated learning could be capable of replacing the two step process: profile generation and grant prediction, by a single one, simplifying the process and possibly obtain better performance.

Could be considered as well other neural network architectures in the context of deep learning, such as recurrent neural networks that, given the temporal characteristic of our dataset, could capture patterns in the data that traditional neural networks cannot. Testing the performance, in a real scenario, of federated learning with recurrent neural networks or other neural network architectures would also be interesting, as there is little work in this area.

Appendix A

Supplementary Information

In this appendix we present a more complete description of the data collected from each permission request in section A.1. We also provide an actual example of the data collected from a permission request in section A.2.

A.1 Description of the data collected from a permission request

In this section we provide a complete description of every data field of the data collected from a permission request.

- **permissionDialogData:** *object.* Contains the data pertaining to a permission dialog.
  - **permissionDialogDataId:** *long.* Internal identifier of the permission dialog data.
  - **requestingApplicationInfo:** *object.* Contains data about the requesting application.
    - **name:** *nullable string.* Application name as returned by the PackageManager getApplicationLabel().
    - **packageName:** *string.* Package name of the application.
    - **longVersionCode:** *nullable long.* The long version code of the application as returned by PackageInfo getLongVersionCode().
    - **flags:** *nullable integer.* Application flags as given by ApplicationInfo flags.
    - **isSystemApp:** *nullable boolean.* Whether the app is a system app or a third-party app.
    - **isForeground:** *nullable boolean.* Whether the app is currently in foreground or background.
    - **category:** *integer.* The category of the app as obtained from ApplicationInfo category.
    - **uid:** *nullable integer.* The user id assigned to the application.

    * **pid:** *nullable integer.* The process id of the running application.

- **ipcCallingApplicationInfo:** *nullable object.* Contains the data about the application that called (through interprocess communication) the permission requesting app (identified by the object in requestingApplicationInfo). This field does not exist if there was no interprocess call that lead to the permission request. Contains the same fields as requestingApplicationInfo.
- **checkedPermission:** *string.* The name of the permission being requested as declared in the manifest. Default android permissions can be found here.
- **checkedPermissionGroup:** *nullable string.* The group of the checkedPermission if any. Default android permission groups can be found here
- **method:** *string.* The calling method used to check if the app has the permission. The value is one of: checkCallingOrSelfPermission; checkCallingPermission; checkPermission; checkSelfPermission.
- **timestamp:** *long.* The epoch time of this permission request.
- **answerType:** *string.* The value can be one of: UNANSWERED, USER_ANSWERED, CACHE_ANSWERED, TIMEDOUT, DISMISSED, UNHANDLED.
- **grantResult:** *integer.* Whether the permission was granted or denied by the user. The value is either 0 if the permission is granted, which corresponds to PackageManager.PERMISSION_GRANTED, -1 if the permission is denied, which corresponds to PackageManager.PERMISSION_DENIED, or *integer.*MIN_VALUE (-2147483648) if there was no answer.
- **selectedSemanticLoc:** *string.* The user semantic location given as input by the participant. The value can be one of: Home, Work, Home, Travelling and Other.
- **wasRequestExpected:** *integer.* Participant input on whether the request was expected. Value is 2 if it was expected, 1 if it was unexpected or 0 if the participant was not sure.

- **contextData:** *nullable object.* Contains the data related to the participant and device context at the time of the permission dialog. This field only exists when answerType in permissionDialogData is USER_ANSWERED.
- **contextDataId:** *long.* Internal identifier of the context data *object.*
- **permissionDialogDataOwnerId:** *long.* The id of the permissionDialogData object that this contextData pertains to.
- **topRunningApplicationInfo:** *object.* The application that is currently being shown to the user as top application. Contains the same fields as requestingApplicationInfo.
- **foregroundRunningApplicationInfoArray:** array of *objects.* An array of the applications that are visible to the user at the time of the permission dialog. Each application has the same fields as requestingApplicationInfo.

- **foregroundRunningApplicationInfoArray:** array of *objects.* An array of the applications that are running in the background at the time of the permission dialog. Each application has the same fields as requestingApplicationInfo.
- **networkStatus:** *string.* The network status at the time of the permission dialog. The value is one of: DISCONNECTED, NOT_METERED or METERED.
- **screenIsInteractive:** *boolean.* Whether the screen is interactive (true) or not (false) as given by the PowerManager isInteractive().
- **isKeyguardLocked:** *boolean.* Whether the keyguard is locked (true) or unlocked (false) as given by the KeyguardManager isKeyguardLocked().
- **dockState:** *integer.* The dock state at the time of the dialog. It is the integer value obtained from EXTRA_DOCK_STATE.
- **callState:** *integer.* The current call state as returned by the TelephonyManager getCallState().
- **plugState:** *integer.* The current plug (charging) state as returned from the field EXTRA_PLUGGED. Value is 0 if it is not plugged, 1 if it is plugged to an AC charger, 2 if it is plugged to an USB port and 3 if the power source is wireless.
- **location:** *object.* Contains the last location reading of the user. The location can be obtained from GPS, network cell towers and wifi access points. Note that the last location does not necessarily means the current location, as the participant might have turned location off.
  - **latitude:** *double.* Latitude reading.
  - **longitude:** *double.* Longitude reading.
  - **time:** *long.* The time of this location reading in milliseconds since January 1st, 1970. Note campaign 3 is missings this field.
  - **elapsedRealtimeNanos:** *long.* Time of the location reading in nanos since system boot.
  - **horizontalAccuracyMeters:** *float.* The accuracy of the location reading in meters as returned by Location.getAccuracy().
- **wifiDevices:** *nullable* array of *objects.* Each object in the array is a wifi device in the neighbourhood of the participant. A scan is attempted every 5 minutes, but the participant may deactivate the wifi at any time. Each device has the following fields:
  - **ssid:** *string.* The network name.
  - **bssid:** *string.* The address of the access point.
  - **rssid:** *integer.* The power of the signal measured in RSSI.
  - **timestamp:** *long.* The time that this device was last see in microseconds since boot.
  - **time:** *long.* The time at which this device was last seen in milliseconds since January 1st, 1970. Note campaign 3 is missing this field.

- **bluetoothDevices:** *nullable* array of *objects.* Each object in the array is a bluetooth device in the neighbourhood of the participant. A scan is attempted every 5 minutes, but the participant may deactivate the bluetooth at any time. Each device has the following fields:
  - **name:** *string.* The name of the device as returned by BluetoothDevice.getName().
  - **address:** *string.* The hardware address of the bluetooth device.
  - **type:** *integer.* The device type. The value is 0 if the device type is not known, 1 if it is a Classic - BR/EDR, 2 if it is Low Energy - LE-only or 3 if is Dual Mode - BR/EDR/LE.
  - **bondState:** *integer.* The bond (pair) state of the remote device. Value is 10 if device is not bonded, 11 if is bonding or 12 if is bonded (paired).
  - **deviceClass:** *integer.* The major and minor device class component of the remote device. The value corresponds to one of the device classes in Bluetooth-Class.Device.
  - **rssi:** *short.* The power of the signal measured in RSSI.
  - **timestamp:** *long.* The time at which the last scan that found this device started in milliseconds since January 1, 1970.
- **calendarEvents:** *nullable* array of *objects.* Each object in the array is an instance of a calendar event that is on-going at the time of the permission dialog. Each event has the following fields:
  - **instanceId:** *long.* The identifier of the event instance.
  - **calendarId:** *long.* The identifier of the calendar that the event instance pertains to.
  - **eventId:** *long.* The identifier of the event.
  - **eventBegin:** *long.* The beginning time of the instance, in UTC milliseconds since January 1, 1970.
  - **eventEnd:** *long.* The ending time of the instance, in UTC milliseconds since January 1, 1970.
  - **eventLocation:** *string.* Where the event takes place, as inputted by the user.
  - **eventStatus:** *integer.* The event status. The value is 0 if the status is tentative, 1 if it is confirmed or 2 if it canceled.

A.2 Example of the data collected from a permission request

In this section we provide a real example of a permission request received by our server. Personal identifiable information was replaced with random data.

```
{
  contextData: {
```

```
backgroundRunningApplicationInfoArray: [
    {
       category: -1,
       flags: 550027079,
       isForeground: true,
       isSystemApp: false,
       longVersionCode: 1,
       name: Example,
       packageName: pt.uc.dei.copmode.example,
       pid: 1235,
       uid: 10000
    }
],
calendarEvents: [
    {
       calendarId: 1,
       eventBegin: 1601137396919,
       eventEnd: 1605137396919,
       eventId: 1,
       eventLocation: Example location,
       eventStatus: 1,
       instanceId: 1
    }
],
callState: 0,
contextDataId: 1,
dockState: 0,
foregroundRunningApplicationInfoArray: [
    {
       category: -1,
       flags: 550027079,
       isForeground: true,
       isSystemApp: true,
       longVersionCode: 1,
       name: CM-NPM,
       packageName: pt.uc.dei.copmode.npm,
       pid: 1234,
       uid: 10000
    }
],
isKeyguardLocked: false,
location: {
      elapsedRealtimeNanos: 123123,
      horizontalAccuracyMeters: 10,
      latitude: 60.124512,
      longitude: -60.245
    },
    networkStatus: NOT_METERED,
    permissionDialogDataOwnerId: 1,
    plugState: 0,
    screenIsInteractive: true,
    topRunningApplicationInfo: {
      category: -1,
      flags: 550027079,
      isForeground: true,
      isSystemApp: true,
      longVersionCode: 1,
      name: CM-NPM,
      packageName: pt.uc.dei.copmode.npm,
      pid: 1234,
      uid: 10000
    },
    wifiDevices: [
      {
        bssid: aa:aa:aa:aa:aa:aa,
        rssid: -20,
        ssid: WIFI_NAME,
        timestamp: 123123123
      }
    ]
  },
  permissionDialogData: {
    answerType: USER_ANSWERED,
    checkedPermission: android.permission.WRITE_EXTERNAL_STORAGE,
    checkedPermissionGroup: android.permission-group.STORAGE,
    grantResult: 0,
    method: checkSelfPermission,
    permissionDialogDataId: 1,
    requestingApplicationInfo: {
      category: -1,
      flags: 550027079,
      isSystemApp: true,
      longVersionCode: 1,
      name: CM-NPM,
            packageName: pt.uc.dei.copmode.npm,
            pid: 1234,
            uid: 10000
        },
         selectedSemanticLoc: Home,
         timestamp: 123123123,
         wasRequestExpected: 1
    }
}
```

Bibliography

[1] Wenyun Dai, Meikang Qiu, Longfei Qiu, Longbin Chen, and Ana Wu. Who moved my data? privacy protection in smartphones. *IEEE Communications Magazine*, 55:20–25, 01 2017. doi:10.1109/MCOM.2017.1600349CM.

[2] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. A conundrum of permissions: Installing applications on an android smartphone. In Jim Blyth, Sven Dietrich, and L. Jean Camp, editors, *Financial Cryptography and Data Security*, pages 68–79, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN: 978-3-642-34638-5.

[3] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 1025–1035, New York, NY, USA, 2014. Association for Computing Machinery. ISBN: 9781450327565. doi:10.1145/2568225.2568276.

[4] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, page 31–40, New York, NY, USA, 2012. Association for Computing Machinery. ISBN: 9781450313124. doi:10.1145/2420950.2420956.

[5] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, New York, NY, USA, 2012. Association for Computing Machinery. ISBN: 9781450315326. doi:10.1145/2335356.2335360.

[6] Panagiotis Andriotis, Gianluca Stringhini, and Martina Angela Sasse. Studying users' adaptation to android's run-time fine-grained access control system. *Journal of Information Security and Applications*, 40:31–43, 2018. ISSN: 2214-2126. doi:https://doi.org/10.1016/j.jisa.2018.02.004.

[7] Lena Reinfelder, Andrea Schankin, Sophie Russ, and Zinaida Benenson. An inquiry into perception and usage of smartphone permission models. In Steven Furnell, Haralambos Mouratidis, and Günther Pernul, editors, *Trust, Privacy and Security in Digital Business*, pages 9–22, Cham, 2018. Springer International Publishing. ISBN: 978-3-319-98385-1.

[8] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673.

[9] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014. ISBN: 1107057132.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Unsupervised Learning*, pages 485–585. Springer New York, New York, NY, 2009. ISBN: 978-0-387-84858-7.

[11] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[12] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[13] Christopher Ifeanyi Eke, Azah Anir Norman, Liyana Shuib, and Henry Friday Nweke. A survey of user profiling: State-of-the-art, challenges, and solutions. *IEEE Access*, 7: 144907–144924, 2019. doi:10.1109/ACCESS.2019.2944243.

[14] Mehdi Srifi, Ahmed Oussous, Ayoub Ait Lahcen, and Salma Mouline. Recommender systems based on collaborative filtering using review texts—a survey. *Information*, 11(6), 2020. ISSN: 2078-2489. doi:10.3390/info11060317.

[15] Michael J Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial intelligence review*, 13(5):393–408, 1999.

[16] Keith Bradley, Rachael Rafter, and Barry Smyth. Case-based user profiling for content personalisation. In Peter Brusilovsky, Oliviero Stock, and Carlo Strapparava, editors, *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 62–72, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN: 978-3-540-44595-1.

[17] W. Paireekreng and K.W. Wong. Client-side mobile user profile for content management using data mining techniques. In *2009 Eighth International Symposium on Natural Language Processing*, pages 96–100, 2009. doi:10.1109/SNLP.2009.5340939.

[18] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Aerin Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016)*, pages 27–41, 2016.

[19] Nicola Guarino and Pierdaniele Giaretta. Ontologies and knowledge bases: Towards a terminological clarification. In *Towards very Large Knowledge bases: Knowledge Building and Knowledge sharing*, pages 25–32. IOS Press, 1995.

[20] Xiaohui Tao, Yuefeng Li, and Ning Zhong. A personalized ontology model for web information gathering. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):496–511, 2011. doi:10.1109/TKDE.2010.145.

[21] Yuvraj Agarwal and Malcolm Hall. Protectmyprivacy: Detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, page 97–110, New York, NY, USA, 2013. Association for Computing Machinery. ISBN: 9781450316729. doi:10.1145/2462456.2464460.

[22] Bahman Rashidi, Carol Fung, and Tam Vu. Dude, ask the experts!: Android resource access permission recommendation with recdroid. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 296–304, 2015. doi:10.1109/INM.2015.7140304.

[23] Yuchen Zhao, Juan Ye, and Tristan Henderson. Privacy-aware location privacy preference recommendations. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MOBIQUITOUS '14, page 120–129, Brussels, BEL, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN: 9781631900396. doi:10.4108/icst.mobiquitous.2014.258017.

[24] PATRICIA A. NORBERG, DANIEL R. HORNE, and DAVID A. HORNE. The privacy paradox: Personal information disclosure intentions versus behaviors. *The Journal of Consumer Affairs*, 41(1):100–126, 2007. ISSN: 00220078, 17456606.

[25] Jierui Xie, Bart Piet Knijnenburg, and Hongxia Jin. Location sharing privacy preference: Analysis and personalized recommendation. In *Proceedings of the 19th International Conference on Intelligent User Interfaces*, IUI '14, page 189–198, New York, NY, USA, 2014. Association for Computing Machinery. ISBN: 9781450321846. doi:10.1145/2557500.2557504.

[26] Qatrunnada Ismail, Tousif Ahmed, Apu Kapadia, and Michael K. Reiter. *Crowdsourced Exploration of Security Configurations*, page 467–476. Association for Computing Machinery, New York, NY, USA, 2015. ISBN: 9781450331456.

[27] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Proceedings of the Tenth USENIX Conference on Usable Privacy and Security*, SOUPS '14, page 199–212, USA, 2014. USENIX Association. ISBN: 9781931971133.

[28] Bin Liu, Jialiu Lin, and Norman Sadeh. Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help? In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, page 201–212, New York, NY, USA, 2014. Association for Computing Machinery. ISBN: 9781450327442. doi:10.1145/2566486.2568035.

[29] Ramprasad Ravichandran, Michael Benisch, Patrick Gauge Kelley, and Norman Sadeh. Capturing social networking privacy preferences: Can default policies help alleviate tradeoffs between expressiveness and user burden? In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, New York, NY, USA, 2009. Association for Computing Machinery. ISBN: 9781605587363. doi:10.1145/1572532.1572587.

[30] Odnan Ref Sanchez, Ilaria Torre, Yangyang He, and Bart Knijnenburg. A recommendation approach for user privacy preferences in the fitness domain. *User Modeling and User-Adapted Interaction*, 30, 07 2020. doi:10.1007/s11257-019-09246-3.

[31] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[32] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020. doi:10.1109/MSP.2020.2975749.

[33] Ram Rajagopal Oskar J. Triebe. Federated K-Means: clustering algorithm and proof of concept. arxiv preprint. Available: https://github.com/ourownstory/federated_kmeans/blob/master/federated_kmeans_arxiv.pdf. Accessed: 2020-11-26., 2020.

[34] M Emre Celebi, Hassan A Kingravi, and Patricio A Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert systems with applications*, 40(1):200–210, 2013.

[35] Mona Hamidi, Mina Sheikhalishahi, and Fabio Martinelli. A secure distributed framework for agglomerative hierarchical clustering construction. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 430–435, 2018. doi:10.1109/PDP2018.2018.00075.

[36] Zoe L Jiang, Ning Guo, Yabin Jin, Jiazhuo Lv, Yulin Wu, Zechao Liu, Junbin Fang, Siu-Ming Yiu, and Xuan Wang. Efficient two-party privacy-preserving collaborative k-means clustering protocol supporting both storage and computation outsourcing. *Information Sciences*, 518: 168–180, 2020.

[37] Kai Xing, Chunqiang Hu, Jiguo Yu, Xiuzhen Cheng, and Fengjuan Zhang. Mutual privacy preserving k-means clustering in social participatory sensing. *IEEE Transactions on Industrial Informatics*, 13(4):2066–2076, 2017.

[38] Hui Yin, Jixin Zhang, Yinqiao Xiong, Xiaofeng Huang, and Tiantian Deng. PPK-means: Achieving privacy-preserving clustering over encrypted multi-dimensional cloud data. *Electronics*, 7(11):310, 2018.

[39] J. Yuan and Y. Tian. Practical privacy-preserving mapreduce based k-means clustering over large-scale dataset. *IEEE Transactions on Cloud Computing*, 7(2):568–579, 2019.

[40] Jialin Wang, Hanni Cheng, Minhui Xue, and Xiaojun Hei. Revisiting localization attacks in mobile app people-nearby services. In Guojun Wang, Mohammed Atiquzzaman, Zheng Yan, and Kim-Kwang Raymond Choo, editors, *Security, Privacy, and Anonymity in Computation,*

Communication, and Storage, pages 17–30, Cham, 2017. Springer International Publishing. ISBN: 978-3-319-72389-1.

[41] Vincent Schellekens, Antoine Chatalic, Florimond Houssiau, Y-A De Montjoye, Laurent Jacques, and Rémi Gribonval. Differentially private compressive k-means. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7933–7937. IEEE, 2019.

[42] Zhigang Lu and Hong Shen. A convergent differentially private k-means clustering algorithm. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 612–624. Springer, 2019.

[43] Dong Su, Jianneng Cao, Ninghui Li, Elisa Bertino, and Hongxia Jin. Differentially private k-means clustering. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 26–37. ACM, 2016.

[44] Xiaoyi Hu, Liping Lu, Dongdong Zhao, Jianwen Xiang, Xing Liu, Haiying Zhou, Shengwu Xiong, and Jing Tian. Privacy-preserving k-means clustering upon negative databases. In *International Conference on Neural Information Processing*, pages 191–204. Springer, 2018.

[45] Chris Clifton and Tamir Tassa. On syntactic anonymity and differential privacy. In *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, pages 88–93. IEEE, 2013.

[46] Tom Farrand, Fatemehsadat Mireshghallah, Sahib Singh, and Andrew Trask. Neither private nor fair: Impact of data imbalance on utility and fairness in differential privacy. In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, PPMLP'20, page 15–19, New York, NY, USA, 2020. Association for Computing Machinery. ISBN: 9781450380881. doi:10.1145/3411501.3419419.

[47] Amira Soliman, Sarunas Girdzijauskas, Mohamed-Rafik Bouguelia, Sepideh Pashami, and Slawomir Nowaczyk. Decentralized and adaptive k-means clustering for non-iid data using hyperloglog counters. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 343–355. Springer, 2020.

[48] Eshref Januzaj, Hans-Peter Kriegel, and Martin Pfeifle. Towards effective and efficient distributed clustering. In *Workshop on Clustering Large Data Sets (ICDM2003)*, volume 60, 2003.

[49] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[50] SEAL. Microsoft SEAL (release 3.5). https://github.com/Microsoft/SEAL, April 2020. Microsoft Research, Redmond, WA. Accessed: 2020-11-26.

[51] William Navidi, William S Murphy Jr, and Willy Hereman. Statistical methods in surveying by trilateration. *Computational statistics & data analysis*, 27(2):209–227, 1998.

[52] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1): 193–218, 1985.

[53] Julio-Omar Palacio-Niño and Fernando Berzal. Evaluation metrics for unsupervised learning algorithms. *CoRR*, abs/1905.05667, 2019.

[54] Chunhui Yuan and Haitao Yang. Research on k-value selection method of k-means clustering algorithm. *J—Multidisciplinary Scientific Journal*, 2(2):226–235, 2019.