Master of Science in Informatics at Grenoble
Master Informatique
Specialization Data Science and Artificial Intelligence

DRIFT : A Federated Recommender System with Implicit Feedback

Theo Nommay

August 21, 2022

Research project performed at Laboratoire d'Informatique de Grenoble

Under the supervision of:
Massih-Reza Amini
Radu Ciucanu
Marta Soare

Defended before a jury composed of:
Massih-Reza Amini
Philippe Mulhem
Georges Quénot
Marta Soare
Danielle Ziebelin

September 2022

Abstract

Nowadays there are more and more items available online, this makes it hard for users to find items that they like. Recommender systems aim to find the item that best suits the user, using his interaction history. Depending on the context, these interactions may be more or less sensitive and collecting them brings an important problem concerning users' privacy. Federated systems have shown that it is possible to make accurate and efficient recommendations without storing users' personal information. However, these systems use instantaneous feedback from the user. In this report, we propose DRIFT, a federated architecture for recommender systems, using implicit feedback. Our learning model is based on a recent algorithm for recommendation with implicit feedbacks SAROS [3] . We aim to make recommendations as precise as SAROS, without compromising the users' privacy. In this report we show that thanks to our experiments, but also thanks to a theoretical analysis on the convergence. We have shown also that the computation time has a linear complexity with respect to the number of interactions made. Finally, we have shown that our algorithm is secure, and participants in our federated system cannot guess the interactions made by the user, except DOs that have the item involved in the interaction.

Résumé

A l'heure d'aujourd'hui, il y a de plus en plus d'articles disponibles en ligne, cela rend difficile pour les utilisateurs de trouver des articles qui leur plaisent vraiment. Les systèmes de recommandation sont la pour aider l'utilisateur à trouver l'article qui lui convient le mieux, en utilisant l'historique de ses interactions. Selon le contexte, ces interactions peuvent être plus ou moins sensibles et les sauvegarder pose un problème important concernant la vie privée des utilisateurs. Les systèmes fédérés ont montré qu'il est possible de faire des recommandations précises et efficaces sans stocker les informations personnelles des utilisateurs. Cependant, ces systèmes utilisent un retour instantané de l'utilisateur. Dans ce rapport, nous proposons DRIFT, une architecture fédérée pour les systèmes de recommandation, utilisant un retour implicite des utilisateurs. Notre modèle est basé sur un algorithme récent de recommandation avec feedbacks implicites SAROS [3]. Notre objectif est de faire des recommandations aussi précises que SAROS, sans compromettre la vie privée des utilisateurs. Dans ce rapport nous le montrons que grâce à nos expériences, mais aussi grâce à une analyse théorique sur la convergence. Nous avons également montré que le temps de calcul a une complexité linéaire par rapport au nombre d'interactions effectuées. Enfin, nous avons montré que notre algorithme est sécurisé, et que les entitées de notre système fédéré ne peuvent pas deviner les interactions faites par l'utilisateur, à l'exception des DO qui ont l'objet impliqué dans l'interaction.

All symbols used in this report are available in the Figure 0.1

Acknowledgement

I would like to express my sincere gratitude to Marta Soare, Radu Ciucanu, and Massih-Reza Amini for their invaluable assistance throughout the internship.

I would also like to express my sincere thanks to Aleksandra Burashnikova, for all the help she gave me in understanding her algorithm.

Contents

I	The total number of items
U	The total number of users
i	The index of the item i
u	The index of the user u
Ψ	The set of item representation
Υ	The set of user representation
ψ_i^t	The d-dimension vector that represent the item i at timestamp t
υ_u^t	The d-dimension vector that represent the user u at timestamp t
S_u	The scores of items for user u
DO_i	The Data Owner i
K	The total number of DO
COS	The Central Orchestration Server
$C = (u, i, is\_positive)$	The triplet representing the interaction of the user u with the item i.
$B_u^{k,t}$	The block for the user u at timestamp t created by DO_k
Π_u^t	The list of positive interaction in the block for the user u at timestamp t
η_u^t	The list of negative interaction in the block for the user u at timestamp t
$l_{\mathrm{u,i,i'}}$	The loss for user u and two items i and i'
$\mathscr{L}_{B_u}$	Empirical ranking loss with respect to a block of items
σ	The sigmoid function
MAP	The Mean Average Precision
NDCG	The Normalized Discounted Cumulative Gain

Table 0.1: All the notation used

— 1 —
Introduction

We introduce this report in 3 parts, first we will present the problem statement in Section 1.1, then we will show how our method to solve this problem in Section 1.2, we will finish with a resume of all the Sections of this report in Section 1.3

1.1 Problem statement

With the increasing number of available items, recommender systems have more and more interactions to manage. These interactions are usually treated with a collaborative filtering algorithm, who needs to collect and store all the data in one central server. However, it is not always possible to collect user data because of potentially sensitive data, for example in the medical or financial context. Some organizations may profit from having access to these data, by selling them or merging them with other organizations, which may jeopardize users' privacy. To solve this problem, recent works use federated learning for their recommender system. A federated recommender system aims to update the learning model without sending the user interaction. To do that, it computes the values needed to update separately from the central server, to hide potentially sensitive information. These systems mostly take only into account a positive interaction of the user, with matrix factorisation or reinforcement learning for example. This is problematic because it does not take into account when the user has no interaction with the proposed item. Moreover, it does not keep the context of the interaction to understand the choice of the user. For example a user may choose an article among those that are proposed, but other items that are not proposed may please him even more.

1.2 Scientific Approach and Investigative Method and Results

Using implicit feedback, our objective is twofold

- We aim to build an efficient federated architecture which is able to recommend items that best suits the user. A federated architecture is based on multiple Data Owners (DOs) that are orchestrated by a Central Orchestration Server (COS). Updating a model in a federated architecture consist to hide the user's information from the central server. To do that, all the computations needed with these informations are computed locally on

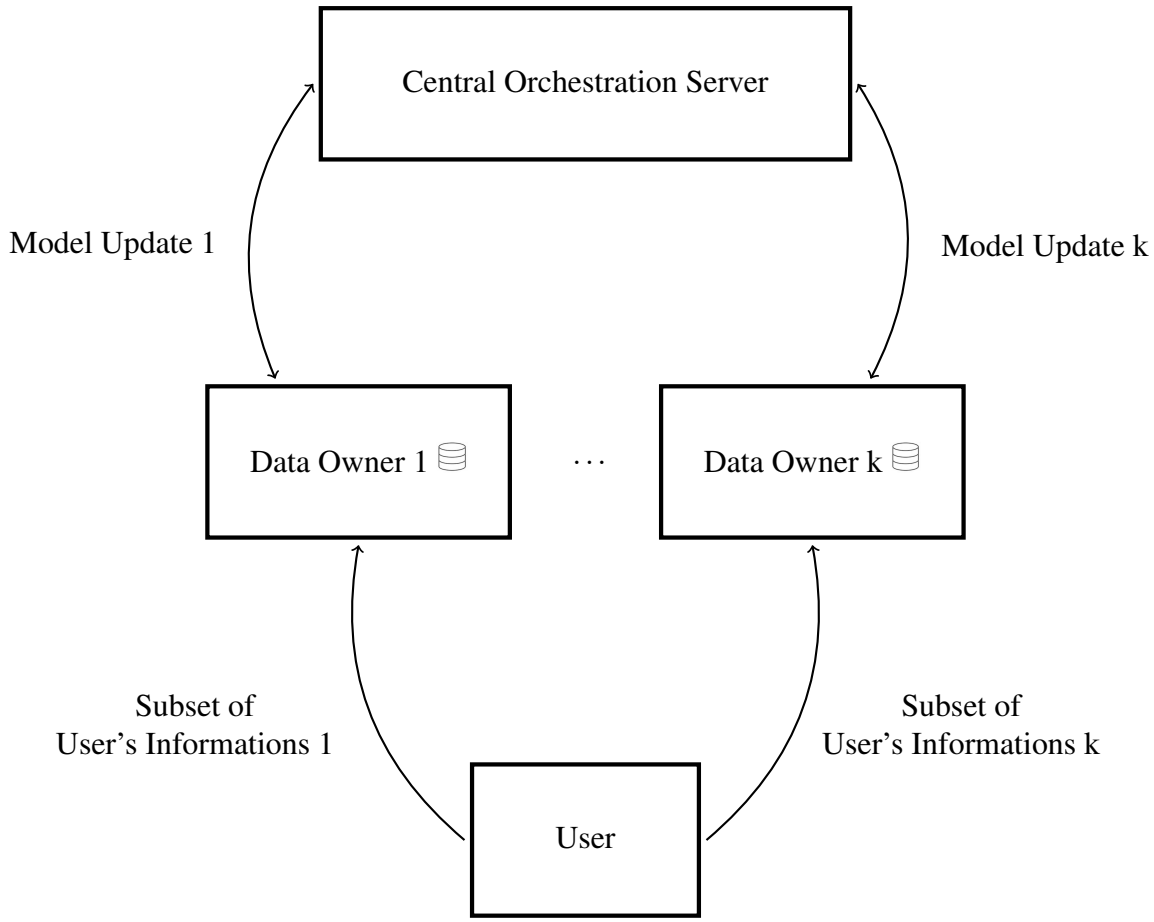

Figure 1.1: Update Model With Federated Learning

the users devices. The organizations containing the items that users interact with, are represented as different DOs. We summarize this principles in the Figure 1.1.

- We also decide to guarantee data security by avoiding any leak in the architecture, using cryptographic tools.

Our federated architecture is orchestrated by the COS. Each item is stocked into multiples DOs, who can communicate with both users and COS. Our learning algorithm is built following a recent algorithm from recommendation with implicit feedback, SAROS [3]. It consists of creating one block of items per user, when a positive interaction follows a negative one, the block is complete and the model parameters are updated. Each DO will store these blocks as local parameters, and the COS will store the learning model as global parameter. When a block is complete the DO will communicate with the COS to update the learning model.

The objective of our algorithm is threefold, we ensure relevant recommendations, we ensure a low computation time, and we ensure the security of the users. In this report :

- We analyze the precision of our recommendation, using the Mean Average Precision and the Normalized Discounted Cumulative Gain. Moreover, we provide a proof of convergence, which shows that the loss of our algorithm will converge to the same loss

as a non secured version of this algorithm, in other words, that the relevance of the recommendation will be similar to a non secured algorithm.

- We analyze the complexity of our algorithm thanks to a theoretical analysis, where we have shown that our algorithm has a linear complexity in the number of interactions.
- We provide theoretical analysis where we have shown that all the participants of our federated system are unable to guess the users' interactions, even if they intercept messages in the system.

1.3 Contents of this report

The rest of this report is organized as follows.

Chapter 2 present related works. We start by Section 2.1 where present our motivation and context. In addition we study the last recsys' 21 conference, where we saw that the privacy of the user was a crucial point for the community, and where a lot of secure recommender systems were based on federated learning. For this reason we will next focus on Federated Learning in Section 2.2. After the architecture is decided, we focus on recommender systems on Section 2.4 where we will see in detail different recent federated and non federated recommender systems with implicit and explicit feedback. As we said in Section 1.2, we set our learning algorithm following the SAROS [3] algorithm. Finally, since we decide to secure the communication between the participants, we present in Section 2.3 different encryption tools to secure communication between participants.

Chapter 3 presents our algorithm. We start by presenting what happens at the initialization of the algorithm in Section 3.1. We will then focus on the recommendation to the user in Section 3.2. Then we present the core of our algorithm in Section 3.3, where we present how DOs build the local parameters thanks to user interactions. We finish with Section 3.4, where we present how the COS updates the global parameters thanks to the local parameters built in the DOs.

Chapter 4 presents the technical choices that we made in our algorithm. We start by presenting the repartition of the task in Section 4.1 , where we explain different possibilities with their advantages and their drawbacks. We finally chose to compute gradients directly in the DO and to share them with the COS. We will then focus on the computations of these gradients in Section4.2 . We finish with the encryption of the sensitive information in Section 4.3 .

Chapter 5 presents the technical analysis that we provide. We start by showing that our algorithm can converge to the same minimizer as a non secure algorithm in Section 5.1. Then we analyze the security of DRIFT in Section 5.2. Our objective here is to show that all participants cannot guess the users' interaction with a probability better than random, even if they intercept messages in the network. We finish with Section 5.3 where we compute the complexity of DRIFT. We have shown that the complexity of our secure algorithm is linear in terms of the number of interactions.

Chapter 6 presents the experiments that we made. We aim to have recommendations as relevant to SAROS. In Section 6.1 we present the metrics that we used to compare these two

algorithms, and all the settings of our experiments In Section 6.2 we analyze the results of our recommendation, we do it in two parts. We first analyze the evolution of the loss, where we saw that DRIFT converges earlier than SAROS, then we analyze the evolution of the metrics after DRIFT converges and after SAROS converges. This part confirm the assumption of convergence made in Chapter 5

— 2 —
State-of-the-art

In this Chapter we will see the current scientific state of the art about security in recommender systems. To do that we will first focus on the motivation and context of the work in Section 2.1 . After that, we will see in detail what has been done in federated learning, in Section 2.2 , and how we can encrypt the messages in Section 2.3 . After that, we will focus on recents Recommender Systems in Section 2.4

2.1 Motivation and Context

Federated Learning is a machine learning technique that trains a model across multiple service providers while keeping the training data locally. This technique is used to avoid a centralized system, due to potential sensitive data. Moreover, as said in Introduction (Section 1.1), there is an urge to focus on security in recommender systems. This motivates us to focus on security in recommender systems, using the federated learning paradigm. Moreover, recent works as [6] shows that it is possible to have a relevant algorithm with a secure framework, thanks to federated learning.

In addition we saw in the last **Recsys21 conference**[1], that the privacy of users in recommender systems was a crucial point for the community. A lot of these works show that the collecting and handling potentially sensitive data raises serious privacy problems, as [10] which show that if there is a collaboration between different vendors to increase the size of their dataset, it poses an important privacy problem for vendors and users. To avoid having a centralized system, recents works use a federated system, as [15] who show that federated method is better suited to protect privacy. Moreover some other recent works transform the most standard recommendation algorithm into a federated one to avoid a centralized system, where a server may collect data. For example [17] who create a federated system for recommendation using collaborative filtering, or [4] who create a federated system for recommendation with matrix factorization.

2.2 Federated learning

We aim to use this technique to build a secure learning system, this is why we built DRIFT upon the typical federated learning characteristics. First, when a DO manages interactions,

[1] `https://recsys.acm.org/recsys21/`

local parameters are created. The DO keeps them locally, and each one has only access to his own data. Since the COS orchestrates the entire system, and can communicate with all DOs, it must manage the global parameters. The data of an organization will be stored into a dedicated DO. These characteristics are resume in Figure 1.1 .[8] notice that the information of the score given by a user shall also be well protected, and that the server should not know which user interacted with which item. So that COS does not know which user is updating its values, we will store the user interactions into different DOs, who can communicate with the COS. Recent works in Federated Recommender Systems, store the information of the user directly on their devices, as [16] which save the historical clic news in it. In our case, since we will need different DO to save the items, we will prefer a cross-silo architecture, where different DO will store the user information. We are positioning our architecture following a recent survey [12], where we keep some typical characteristics:

- **Data distribution.** Local parameters are generated and remain decentralized. Moreover, each DO cannot read the data of other DO
- **Data availability.** DO make few and small computations, they are almost always available.
- **Distribution scale.** We have few DO, generally less than 100. They are created for each organization in the system.
- **Addressability.** The COS can access a specific DO thanks to an id.
- **Statefulness.** Each DO maintains local variables throughout the execution of the entire algorithm.
- **Data partition.** The partition should be fixed. In our case, we assume that the number of organizations in our system is fixed, which implies that the data in each DO is fixed.
- **Incentive mechanisms.** To ensure the honest participation of each DO we need incentive mechanisms. In our case, since each DO is related to a unique organization, we assume that they are business competitors, with a monetary gain according to the number of items chosen by the user.

In our case, the knowledge of our participants is limited and totally independent, so it is not a problem if they are trying to analyze the data, but we need to be sure that the results that they send are correct. Following [7], we decide to suppose that our participants are **honest but curious**.

2.3 Encryption

Since we supposed that our participants are honest but curious, they can sniff the network and intercept messages sent in our algorithm. This must not compromise the privacy of the user, so we decide to encrypt the messages. To do that we follow the encryption of a recent federated framework [6]. As we want to have an efficient algorithm, we relies DRIFT to a NIST standard for crypto-system, AES-GCM [1]. The AES-GCM crypto-system is defined by a triplet of polynomial-time algorithms (Gen,Enc,Dec) and a security parameter λ such that

$\text{Gen}(1^\lambda)$ generates a uniformly random symmetric key, according to λ. Let $c_1 = \text{Enc}(m_1)$ be the encryption of a message m_1 and let a message $m_2 = \text{Dec}(c_1)$ be the decryption of c_1. If the symmetric key κ is the same for both operations, then $m_1 = m_2$. In our case we need to hold this assumption, so that DOs are able to use the user's interaction. We create keys at the very beginning, and we share them with each DO when they enter into the system. Then DOs and users are able to read and write encrypted information. Moreover AES-GCM is IND-CPA secure [2], so if an external attacker can read an encrypted message, he cannot guess the decrypted one with a probability better than random.

2.4 Recommender Systems

When a user interacts with a recommender system, it creates an interaction. It can be

- **Explicit**, where the user gives a score to the item.
- **Implicit**, where the feedback depends on if the user clicks or not in the item.

It is harder to manage implicit feedback, if a user does not click on an item, it does not mean that the user is not interested. On the other hand, if a user clicks on the item, it does not mean that the recommendation is pertinent, maybe she wants to click on it anyway. But since it is easier to collect implicit feedback (generally in the form of clicks), we adapt DRIFT for implicit rewards. There are multiple ways to update the model of a federated recommender system, recents works are mainly based on Matrix factorization as [14] where the rating predictions is replaced by a Multi Layer Perceptron, trained on the rates given by users. A lot of works are also based on reinforcement learning, as [13], which use the Upper Confidence Bound algorithm their federated recommender system, a reinforcement technique which aims to find the item who grants the higher rates from the user. But all these approaches mainly consider positive interactions, and the main hypothesis of [3] is that the user preference is better represented when we consider a sequence of positive and negative interactions on the items. This is why we position our algorithm following the learning characteristics of their algorithm SAROS, presented in Section 1.2.

— 3 —
Algorithm DRIFT

DRIFT has 4 different principal steps:

- The Initialization takes place before any user enter in the system, here we initialize DOs and the COS global parameters. We present it in Section 3.1
- The Recommendation takes place after a user enters in the system, here the COS, that has global parameters, will return to the user the item that is supposed to please him the most. We present it in Section 3.2
- The Core of our Algorithm takes place while a user interact with items, here we DOs are updating locals parameters. We present it in Section 3.3.
- The Update of Global Parameters takes place when a DO has complete blocks of interactions, where it will share information with the COS. We present it in Section 3.4.

3.1 Initialization

We represent each item and user as a d-dimension vector, we denote ψ_{i}^t (resp. υ_{u}^t) the value of the item (resp. user) at the index i (resp. u), at time t. When the system is initialized, the COS will create vectors $\Upsilon \in R^{d \times U}$ and $\Psi \in R^{d \times I}$, which contain all the user and item representations. At this point, the COS has no information about the users and items, so it will initialize these vectors at random. When an organization O enters in the system, it will give us all his items Ψ_{O}, we associate a new DO to this organization. When a user wants to interact with items of an organization, he will interact with the associate DO. To redirect the user to the right DO, the COS has a table that maps items to DOs that contain them.

3.2 Recommandation

The main objective is to make recommendations. To do that we need to learn which items are preferred depending on his interactions. The score of an item i for the user u will be equal to the dot product between the two representation vectors. To recommend item to a user, the COS will first create the score vector $S_{\mathsf{u}} \in R^{|\Psi|} = \Psi \times \Upsilon[\mathsf{u}] = \Psi \times \upsilon_{\mathsf{u}}^t$, which contains the score of all the items for the user u. The COS will then return to the user the item with the highest score.

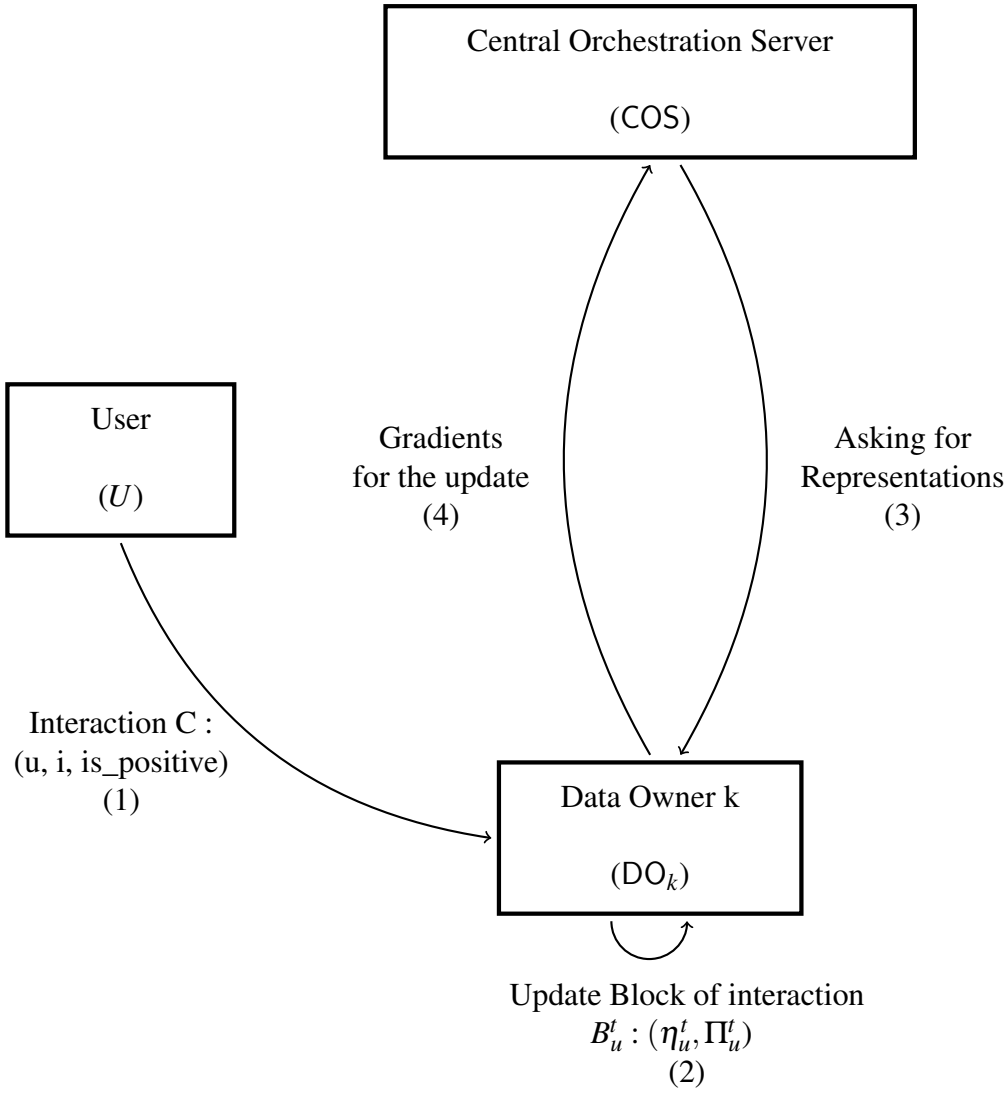

Figure 3.1: The Workflow of the Core of the Algorithm

3.3 Core of the Algorithm

The core of our algorithm is composed of 4 steps:

- Manage the user interaction
- Update the local parameters
- Asking representations for computing the gradients
- Update the global parameters

We schematized the workflow of DRIFT in Figure 3.1. We assume that the COS and DO are well initialized, following the steps described in Section 3.1.

The step (1) begins after a user interacts with an item. This interaction is represented as C, who contain the user id, the item id and the type of interaction (since we are in implicit context,

it can be positive or not). Thanks to his table, the COS can find all the DO where the item is contained. These DO receive the C, and create the user block $B_{\mathsf{u}}^{k,t}$. This block is composed of 2 vectors of items, the positive interactions Π and negative interactions η.

The step (2) begins after a DO receives an interaction, and has to update his block. There are 3 state possible:

1. When it receives a positive interaction, it closes the vector η_{u}^t, and adds it to Π_{u}^t.
2. When it receives a negative interaction, it adds it to the vector η_{u}^t.
3. When it receives a negative interaction while the vector η_{u}^t is close, it considers the block as complete, and starts filling a new block with this interaction.

The principal advantage of this block is that it keeps the context of the interactions, by saving all negative interactions before the positive one.

The steps (3) and (4) begin after a block is completed. Here we need to update the global parameters. To do that the DO asks the COS the representation of the user and the items in $B_{\mathsf{u}}^{k,t}$. Thanks to these representations, the participants are able to update the global parameters. We give more details below.

3.4 Update of Global Parameters

We aim to make recommendations by keeping the context of the interactions. If the user u chooses the item i over i′ the score of i should be higher than the score of i′ for this user. The goal is to avoid misranking over all the items and the set of users, in other words we want to avoid that $S_{\mathsf{u}}[\mathsf{i}] < S_{\mathsf{u}}[\mathsf{i}'] \Leftrightarrow \Psi[\mathsf{i}] \times \Upsilon[\mathsf{u}] < \Psi[\mathsf{i}'] \times \Upsilon[\mathsf{u}] \Leftrightarrow \Upsilon[\mathsf{u}] \times (\Psi[\mathsf{i}] - \Psi[\mathsf{i}']) < 0 \Leftrightarrow \upsilon_u \times (\psi_i - \psi_{i'}) < 0$. We denote $w = \upsilon_u \times (\psi_i - \psi_{i'})$ and $l_{\mathsf{u},\mathsf{i},\mathsf{i}',}(w)$ the ranking loss for the items i, i′ and the user u. The result of the loss should be low when w is high, and high when w is low, so we aim to minimize this loss. To do that we will use the gradient descent principle. The idea is to update the global parameters with the opposite of the loss gradient, this will make the loss tend towards 0. It is possible if and only if the loss is differentiable (it is possible to derive it at any point of the domain). In our case the loss depends on 3 representations, the user, the item i and the item i′. The gradient will be the vector

$$\nabla l_{\mathsf{u},\mathsf{i},\mathsf{i}'}(w) = \begin{bmatrix} \frac{\partial l_{\mathsf{u},\mathsf{i},\mathsf{i}',}}{\partial \upsilon_u}(w) \\ \\ \frac{\partial l_{\mathsf{u},\mathsf{i},\mathsf{i}',}}{\partial \psi_i}(w) \\ \\ \frac{\partial l_{\mathsf{u},\mathsf{i},\mathsf{i}'m,}}{\partial \psi_{i'}}(w) \end{bmatrix}$$

As shown in Section 3.3, the update of the model takes place after completing a local block on interaction. We need to loop over all negative items $\mathsf{i}' \in \eta$ and positive ones $\mathsf{i} \in \Pi$ in the block and compute the ranking loss. This will result to the pairwise ranking loss, with respect

to a block of item. We denote it $\mathscr{L}_{B_u}(w) = \frac{1}{|\eta_u||\Pi_u|} \sum_{i' \in \eta_u} \sum_{i \in \Pi_u} l_{u,i,i'},(w)$. Applying the gradient descent algorithm on our values will result to :

$$\begin{bmatrix} \upsilon_u^{t+1} \\ \psi_i^{t+1} \\ \psi_{i'}^{t+1} \end{bmatrix} = \begin{bmatrix} \upsilon_u^{t} \\ \psi_i^{t} \\ \psi_{i'}^{t} \end{bmatrix} - \alpha \times \nabla \mathscr{L}_{B_u}(w)$$

with $\nabla \mathscr{L}_{B_u}(w) = \nabla \frac{1}{|\eta_u||\Pi_u|} \sum_{i' \in \eta_u} \sum_{i \in \Pi_u} l_{u,i,i'},(w) = \frac{1}{|\eta_u||\Pi_u|} \sum_{i' \in \eta_u} \sum_{i \in \Pi_u} \nabla l_{u,i,i'},(w)$ and $\alpha \in [0,1]$ the learning rate.

— 4 —
Practical implementation

Here we will focus on the practical implementation of DRIFT. We will first focus on the repartition of the different values that it needs to compute in Section 4.1, then we will see how the gradients are computed in Section 4.2. We will finish by seeing how the interactions are secure in Section 4.2. We provide a sequence diagram in Figure 4.1 which resumes all the steps of DRIFT.

4.1 Repartition of the Task

Now that we know exactly how to compute our values, we need to disturb the computations among the participants. Recall that we have 2 major steps for updating the model, compute the loss, and update the global parameters.

Since these global parameters are contained in the COS, our first idea was to send directly the completed block of interaction from each DO to the COS. The principal problem with this method is that the COS can have access to all the interactions of the user, and this does not solve the problem of privacy. We need to hide these blocks from the COS.

To solve that our second idea was to send directly the two vectors Ψ and Υ to the DO when a block is finish. The principal problem with this method is that sending too much, and potentially useless, information could take a lot of time, and if multiples DO finish at the same it can create a bottleneck. Another problem with that method is that if our system is composed of only 2 DO each one is able guess which items are contained, and update in the other DO, so this method does not solve the problem of privacy either.

To solve that our third idea was to ask the COS to send only the representation of all items in blocks and the representation of the user. Each DO will compute directly the gradient following the gradient descent algorithm presented in Section 3.4. Since the gradient value is lighter than the entire representation, the DO will send this result to the COS, who will make the update on the global parameters. This method solves the problem of time complexity by sending only the information needed. Moreover it improves privacy by sending only needed information, and hides to the DO the information it doesn't need. But we still have a problem. The COS can save the old score vector for the user u, and compare it with the updated one. Thanks to these informations, after updating the model for a user, the COS can see if the items scores are increasing, decreasing or staying the same, and it can guess interactions present in the block of interaction. To solve this problem we add a threshold Θ, such that the DO ask for an update if

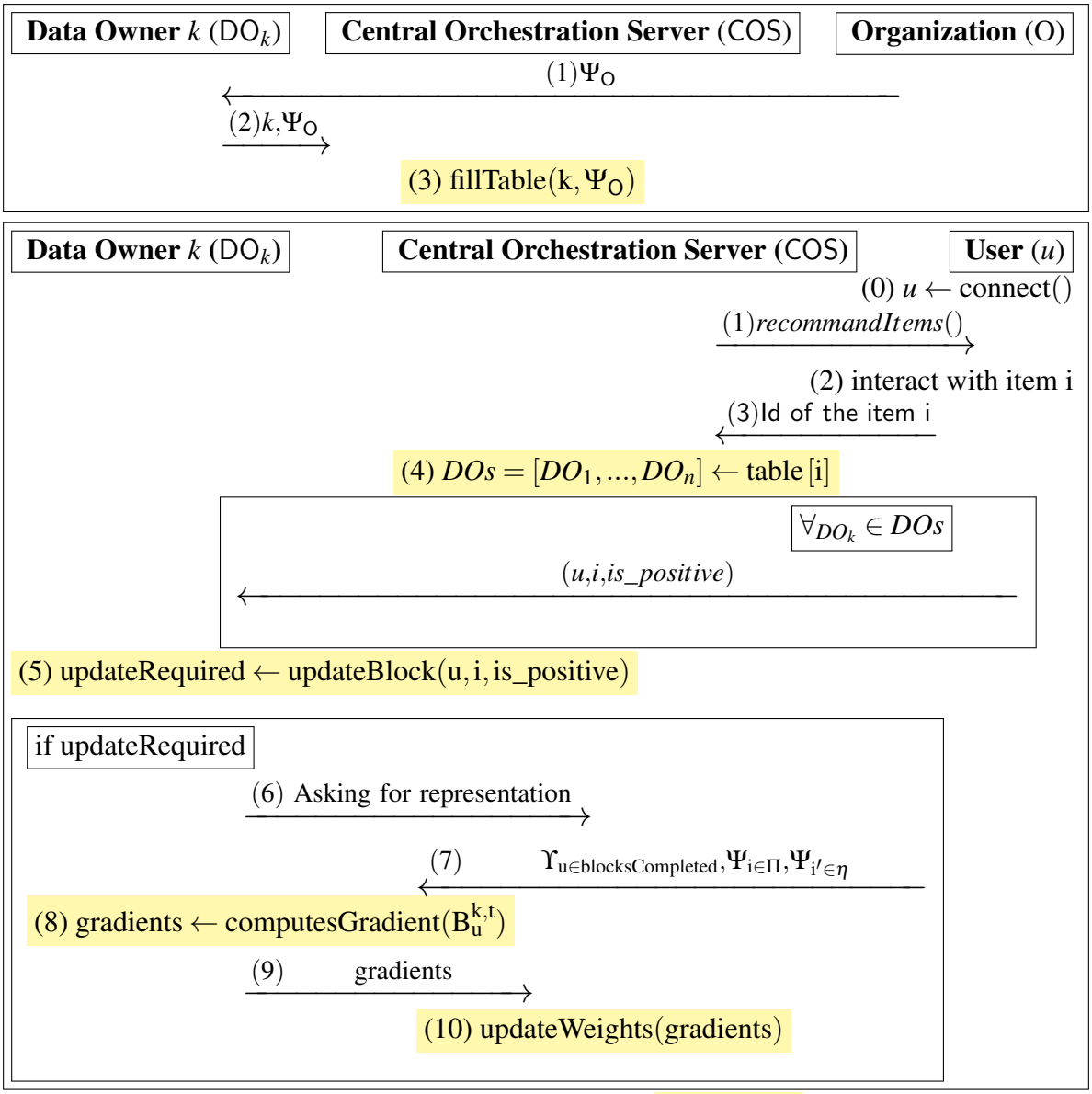

Figure 4.1: Workflow of DRIFT. The first diagram shows the preprocessing of the data, and the second one shows the core of the algorithm
The pseudo code of the algorithm fillTable, updateBlock, computesGradient and updateWeights are available in Appendix 1, 2, 3 and 4 respectively

and only if the number of block completed is greater or equals than Θ. We set $\Theta = 2$, thanks to that, the score of all the items with which users have completed blocks will be updated, and the COS can not conclude anything about the interaction with a specific user.

4.2 Computation of the Gradients

Now that we know that the DO will compute and send the gradient of the loss function, we need to find a loss function and compute its gradients. In our experiments we will use the binary cross entropy loss = $-\frac{1}{N}\sum_{i=1}^{N} y_i log(p(y_i)) + (1-y_i) log(1-p(y_i))$, with y_i the label of the target, and $p(y_i)$ the probability that the target has the right label. In our case we will evaluate the loss between items in the completed block, so we already know which item is preferred, so we have a unique label equal to 1. We need to estimate the probability that the items are well classified, depending on w, as shown in Section 3.4. We will compute this probability thanks to the sigmoid function on the score ($\sigma(w) = \frac{1}{1+e^{-w}}$). We can conclude that in our case, we will have :

$$\nabla l_{\mathrm{u,i,i'}}(w) = \nabla - log(\sigma(w)) = -\frac{\nabla\sigma(w)}{\sigma(w)}$$

With :

$$\nabla\sigma(w) = \nabla\frac{1}{1+e^{-w}} = (\frac{1}{1+e^{-w}})^2\nabla(1+e^{-w}) = -\frac{1}{1+e^{-w}}\frac{e^{-w}}{1+e^{-w}}(\nabla w) = -\sigma(w)(1-\sigma(w))(\nabla w)$$

With

$$(\nabla w) = \begin{bmatrix} \frac{\partial \upsilon \times (\psi - \psi')}{\partial \upsilon} \\ \frac{\partial \upsilon \times (\psi - \psi')}{\partial \psi} \\ \frac{\partial \upsilon \times (\psi - \psi')}{\partial \psi'} \end{bmatrix} = \begin{bmatrix} (\psi - \psi') \\ \upsilon \\ -\upsilon \end{bmatrix}$$

This result to

$$\nabla l_{\mathrm{u,i,i'}}(w) = \begin{bmatrix} (1-\sigma(w))(\psi - \psi') \\ (1-\sigma(w))\upsilon \\ (1-\sigma(w))(-\upsilon) \end{bmatrix}$$

These 3 values will be computed by the DO and send to the COS who will update the global parameters, following the method described in Section 3.4.

4.3 Securing Interactions

Now that we know which data will be computed and sent, we need to know how to secure them. We will focus on the encryption of the triplets C. Recall that this one contains 3 informations, the user id, the item id, and a boolean saying if interaction is positive or not. This triplet will be shared between the user and several DO that have the item. Since this interaction can share sensitive information, and since another participant can intercept this message, we need to secure this exchange to avoid harming the user's privacy. To do that we will use AES-GCM, presented in the state of the art (Section 2.3). At the reception of an interaction the DO will decrypt it, thanks to the key share when it was created. To optimize our architecture we will not encrypt all the triplets, but only the user id and the item id, as a unique tuple. We computes

to the benchmark of [5], we saw that this optimization divides by two the computation time, and since a lot of interaction will be made this gain is significant.

— 5 —
Theoretical Analysis

We analyze the convergence of DRIFT in Section 5.1, the security of DRIFT in Section 5.2, and the complexity of DRIFT in Section 5.3.

5.1 Convergence

We recall that the preprocessing of the data consists of splitting the item into K DO, then each item of a dataset S is present in at least one DO, so at each step, at least one block will be updated. All the interaction between a user and an item will be done directly in the corresponding DO. We can conclude that all the interactions of a dataset S, will be split into smaller datasets $(S_1, S_2, ..., S_K)$. Each of them will participate in the update of the global parameters, by adding his computed loss, we can conclude that the final loss of our algorithm will be : $\mathscr{L}(w) = \sum_{i=1}^{M} \mathscr{L}_i(w)$, with $\mathscr{L}_i(w)$ the total loss created by the DO_i. We know that the time taken by the DO to compute the gradient, depend on a finite block of interaction, so we know that there exist $D < \infty$, such that for any DO_i, the delay $d_k^i < D$. Assume that each $\mathscr{L}_i$ is differentiable and that $\nabla\mathscr{L}_i$ is $\frac{1}{L} - cococercive$. But since the delay is bounded and assumptions on the loss function hold, the sequence produced by our Distributed Gradient update rule converges to the unique minimizer. (Theorem 1 [11]). Moreover, in our experiments, we will use the same loss function as SAROS, we can conclude that both algorithms should converge to the same minimizer.

5.2 Security

The safety characteristics of DRIFT are summarized in Figure 5.1. We will show that all participants cannot guess the user interacts with a probability better than random.

Security of the user with respect to DO. Thanks to the architecture of DRIFT, each DO knows data about its items, both items and blocks are private, so they cannot have any information about another DO. This means that at a time step each DO cannot guess the interactions a user has made with another DO. Moreover, since each DO has a different key for decrypting the interactions, they cannot guess interactions that do not concern it with a probability better than random.

Data \ Participant	DO_i	User	COS	Ext
Interaction	X*	X		Enc
Block of interaction	X*			
Items and Users Weight at time step t	X*		X	

Figure 5.1: *Security properties of* DRIFT. The X means that the participant can see in clear the informations. X* means that the DO_i can see in clear the informations only if it have the items involved. Ext means an external observer having access to all messages exchanged between participants. Enc means that the data are visible but encrypted with AES-GCM. A grayed cell means that the participant cannot see the information.

Security of the user with respect to the COS**.** Thanks to the architecture of DRIFT, the COS only knows about the global parameters. It will receive the gradient to update the representation of several items and users, depending on the hyper-parameter Θ. Moreover, unlike most recent works where the users send their gradients to the server, our algorithm shares the gradients from the DO to the COS, so even if the COS has access to the scores of items for users, the COS is unable to conclude which interaction was made by which user. We can conclude that the COS cannot guess any user interactions, with a probability better than random.

Security of the user with respect to an External observer**.** Thanks to the architecture of DRIFT, there is only 4 information that can be sent on the network, interactions, item representations, user's representations and the gradients computed by DOs.
Interactions are encrypted thanks to AES-GCM, which is IND-CPA secure, so an external observer cannot guess the interaction with a probability better than random.
But other values are not encrypted, so the external observer will have access to items representations, users representations and the gradients computed by DOs. We can conclude that it does not have more information than the COS. Following the same principles as 5.2, we can conclude that an external observer cannot guess the interactions made by users, with a probability better than random.

5.3 Complexity

We will study here the complexity of DRIFT. Since the preprocessing is done once at the beginning, the time spent in this part is constant, we will not take it into account in our results. At each time step 3 operations can be done. We need first to secure and send the interaction, then to share the parameters, and update the model. We will first assume that at each time step, only one DO participate.
First, we will focus on the sending of the user interaction, this is our only cryptographic interaction. Denote A, the time complexity to encrypt and decrypt a tuple composed of the user id and the item id, with AES-GCM. DRIFT requires O(N) encryption, which means that the time to share the information between the user and the DO is on O(NA). Sharing the parameter consists of sharing raw information between the DO and the COS. Since we share only information for a block of interaction, the size of these two messages is bounded by this block's

size. We assume that there is no perturbation in the network, so we assume that the time spent for these communications is negligible.
The last operation made by our algorithm is the update of the global parameters. Each value will be updated after receiving the gradients, the total complexity of this operation will be on O(N).
The total complexity of our algorithm will then be on O(NA + N). Suppose now that our system is composed of K DO, and that at each time step several DO participate. Since they make the same operation, we can conclude that the total complexity of DRIFT will be on O(K(NA + N)) = O(NA), which is linear in N.

— 6 —
Evaluation and Validation of DRIFT

In this Chapter we will present the results of our experiments. We present first the experimental evaluation (Section 6.1), where we present the metrics used, and the experimental setting. Then we analyze the quality of the recommendations (Section 6.2). We will finish with an analysis of the repartition of the time during the training of DRIFT. (Section 6.3).

6.1 Experimental Evaluation

We will evaluate our system thanks to the evolution of the loss during the training, and we validate it thanks to two metrics presented in Section 6.1.1. Before analyzing the results, we present the setting of our experiments in Section 6.1.2.

6.1.1 Metrics

We will focus on 2 metrics based on the precision of our model, in other words, based on the number of relevant items recommended. First the MAP as $\frac{1}{N}\sum_{i=1}^{K} AP_k(u)$ with $AP_k(u)$ The average precision for the user u.
Then the NDCG = $\frac{DCG}{IDCG}$. Denote the DCG the Discounted Cumulative Gain, as $\sum_{i=1}^{K} \frac{2^{rel_i}-1}{log_2(i+1)}$, with rel_i equals to 1 if the item is relevant, 0 otherwise. Denote the IDCG the Ideal Discounted Cumulative Gain, which correspond to the maximum possible value for the DCG, when all the items are pertinent, $\sum_{i=1}^{K} \frac{1}{log_2(i+1)}$.

6.1.2 Experimental Setting

We aim to compare our algorithm with SAROS [3], to see how both algorithms evolve. As we said in Section 5.1, we will use the same loss function for both algorithms. [3]

Dataset. This comparaison will be made using a dataset that contains user rating for movies i.e. using MovieLens [9]. This dataset is composed of 1 million interactions between 6040 users and 3900 items, all of these interactions are dated by the timestamp. Each item has one or multiple genre(s) among the 20 genres in the dataset. We supposed that each genre is representing an organization, so our experiments are composed of 20 DOs. We represent each participant as a unique object, all of them are running in the same machine. As future work, we intend to run DRIFT in a real federated setting. We will next focus on the interaction of

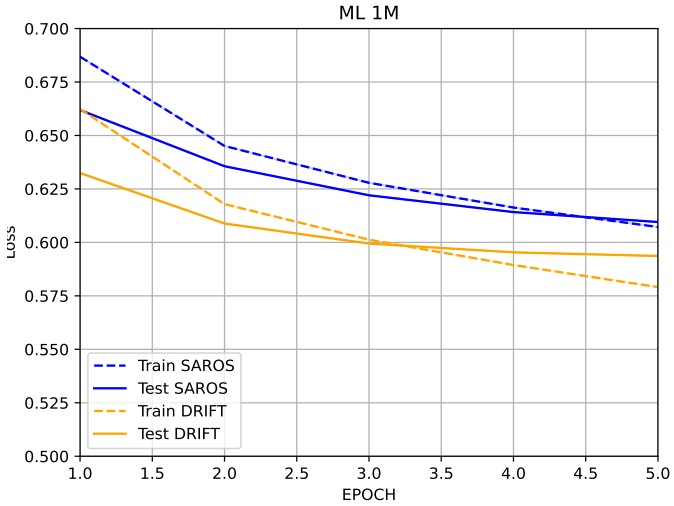

Figure 6.1: Evolution of the loss during the training and testing of SAROS and DRIFT

users. As we need to have implicit interaction, and since the dataset gives us grades on the movie (from 1 to 5), we have considered that the interaction is negative if the rate is lower than 3, otherwise we have considered it as positive. We also need to keep the context to build the blocks of interaction, to do that we sort the dataset with respect to the timestamp. Then we keep 80% first user interactions for the training part, and we will try to guess his 20% remaining interactions, which represent our testing part.

We did our experiments on a virtual machine running Ubuntu, located in a server with a 32 cores Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz and a nVidia RTX A6000. Since we also need to focus on the efficiency of our algorithm, we launched SAROS [3] and DRIFT at the same time to avoid any perturbation of the server. Moreover to have a more efficient algorithm we will also use tensorflow for the update and for the recommendation. Each DO will compute the loss and the gradient of this one, thanks to the gradients computed in Section 4.2. We also add a ridge regression to our loss function, which makes it strongly-convex, and respect all the assumption presented for the correctness, shown in Section 5.1

6.2 Analysis of the Recommendation

We will now focus on the recommendation of DRIFT, compared to the SAROS [3] ones. First, we will focus on the evolution of the loss during the training (Section 6.2.1), then we will focus on the result of our metrics, presented in the Section 6.1.1 (Section 6.2.2).

6.2.1 Evolution of the Loss

We will now focus on the evolution of the loss during the training part and as the testing part. The result of the experiment is available on Figure 6.1. As we can see the evolution of the

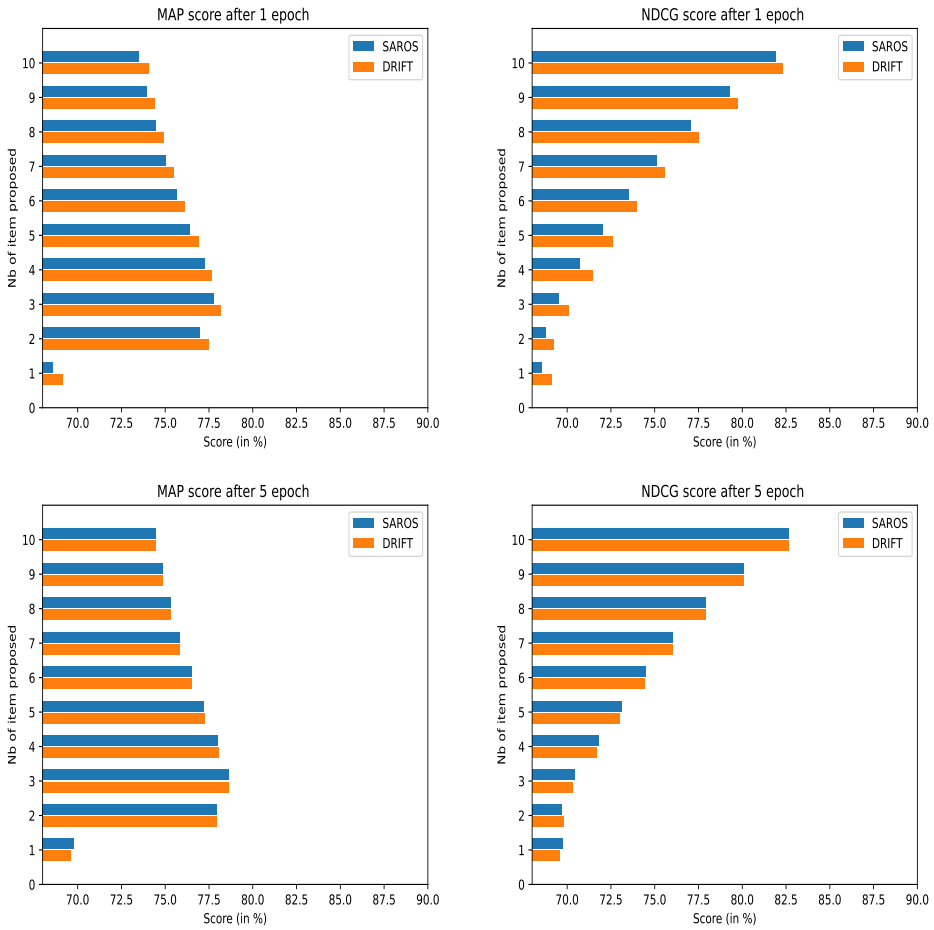

Figure 6.2: Evolution of the Recommendation

algorithm is very similar. But we can see that at the beginning DRIFT has a lower loss than SAROS, but converges after a few epochs.

6.2.2 Evolution of the Metrics

Thanks to the observation of the loss made in Section 6.2.1, we will focus on the recommendation at the very beginning, and at the end of the training. The results of our metrics are available in Figure 6.2. We observe that after one epoch, our architecture outperforms SAROS.

We think that this is due to the preprocessing. In our dataset there is an average of 3 genres per movie, so we can conclude that each block created in SAROS will result in an average of 3 blocks in DRIFT. These blocks will contain less negative items, and will update multiple times the score of the items, this can explain why our recommendations are more precise. To continue to observe this phenomenon, we also compare the results after training. We observe that after the training our architecture does not evolve a lot, unlike SAROS who evolves a lot. This is due to the high number of blocks in DRIFT that made it converge earlier than SAROS. However we observe that at the end, the scores of our metrics are very close between SAROS and DRIFT, which confirm the convergence of the algorithm, shown also theoretically in Section 5.1.

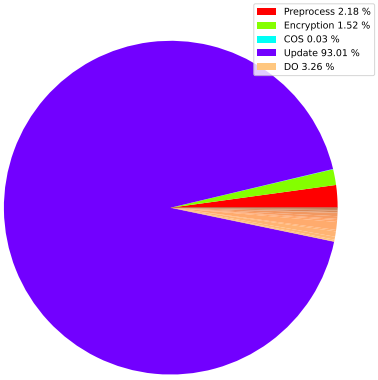

Figure 6.3: Evolution of the time during the running of one epoch. The non label part representing one Data Owner. The legend gives the sum of time spent into the different Data Owners.

6.3 Analysis of the Time

We provide the details of the total time of computation in Figure 6.3. As we can see, we spend the most of the time in the update parts. Less than 10% of the time is used for the security layer of the algorithm, which is reasonable, especially when some part, such as the preprocessing, are constant. Thanks to the fact that we optimize the number of values encrypted, and thanks to the efficiency of AES-GCM, the encryption takes only 1.52% of the total time.

— 7 —

Summary of results, Conclusions, Expected Impact

In this report we aim to solve the problem of security in recommender systems, with a federated recommender system algorithm using implicit feedback. The main piece of our architecture are the DOs which manage the interaction of the user, and participate in the update of the model by communicating with the COS. We proposed DRIFT, a federated recommender system with implicit feedback.

Thanks to a theoretical analysis of the security, where we demonstrated that each participant cannot guess the interactions of the users with a high probability, we showed that the privacy is preserved. We also provide a proof of convergence that shows that our algorithm has the potential to learn as well as a non secure version. Moreover, thanks to our experiments, we can see that our algorithm is shown to be as precise as a non secure version, on MAP and NDCG measure.

Future work As future work we aim to test DRIFT in real federated settings, with multiple machines, each representing one DO and the COS. We also aim to dynamically manage the number of DOs and users in the system, to see how the recommendation will evolve over time. It will be also needed to focus on the minimal number of completed blocks before update, focusing on our hyper-parameter Θ, if this one is too small, there will be a lot of communication, who can bring some bottleneck, If this one is too large the global parameters will be updated only slightly, which may cause bad recommendations for users.
We also aim to test DRIFT on different datasets to complete our results. The main problem with the training is the repartition of data into different datasets.

— A —
Appendix

Algorithm 1 fillTable
Used by the COS during the preprocessing at step 3

Require: **int** k, **List[int]** items
  for item *in* items **do**
    if *not* $self.table.contains(item)$ **then**
      $self.table[item] \leftarrow [\,]$
    end if
    $self.table[item].append(k)$
  end for

Algorithm 2 updateBlock
Used by the DO to update his blocks at step 6

Require: **int** u, **int** i, **bool** $is\_positive$,
  $isFull \leftarrow False$
  $B_u^{k,t} \leftarrow self.getBlockForUser(u)$
  $\eta_u^t \leftarrow B_u^{k,t}.negativeInteractions$
  $\Pi_u^t \leftarrow B_u^{k,t}.positiveInteractions$
  if is_positive **then**
    $\Pi_u^t.add(i)$
  else if $\Pi_u^t.isEmpty()$ **then**
    $\eta_u^t.add(i)$
  else
    self.save[u].append($B_u^{k,t}$)
    $B_u^{k,t+1} \leftarrow newBlock$
    $B_u^{k,t+1}.negativeInteractions.add(i)$
    $updateRequires \leftarrow self.save.size \geq \Theta$
  end if
  return $updateRequires$

Algorithm 3 computesGradient
Used by the DO to computes the gradient
thanks to the representation of the items at step 8

Require: **List[List[double]]** Ψ^{+}, **List[List[double]]** Ψ^{-}, **List[List[double]]** Υ
  for u **in** self.save **do**
    $b \leftarrow self.save[u]$
    for n **in** $b.negativeInteractions$ **do**
      for p **in** $b.postiveInteractions$ **do**
        $d\_u, d\_ipos, d\_ineg \leftarrow \nabla LossFunction(\upsilon_u^t, \psi_n^t, \psi_p^t)$
        $gradients\_items.append((p, d\_ipos))$
        $gradients\_items.append((n, d\_ineg))$
        $gradients\_user.append((u, d\_u))$
      end for
    end for
  end for
  $self.save.reset()$
  return $gradients$

Algorithm 4 updateWeights
Used by the DO to update the weight matrix thanks to the blocks at step 8

Require: **List** $gradients\_user$**List** $gradients\_item$
  for $u, gradient$ **in** $gradients\_user$ **do**
    $\upsilon_u^{t+1} \leftarrow \upsilon_u^t - \alpha \times gradient$
  end for
  for $i, gradient$ **in** $gradients\_item$ **do**
    $\psi_i^{t+1} \leftarrow \psi_i^t - \alpha \times gradient$
  end for

Bibliography

[1] Advanced Encryption Standard (AES). `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf`, 2001. FIPS Publication 197.

[2] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 1997.

[3] A. Burashnikova, Y. Maximov, M. Clausel, C. Laclau, F. Iutzeler, and M. Amini. Learning over no-preferred and preferred sequence of items for robust recommendation. *J. Artif. Intell. Res.*, 71:121–142, 2021.

[4] D. Chai, L. Wang, K. Chen, and Q. Yang. Secure federated matrix factorization. *IEEE Intell. Syst.*, 36(5):11–20, 2021.

[5] R. Ciucanu, P. Lafourcade, M. Lombard-Platet, and M. Soare. Secure Protocols for Cumulative Reward Maximization in Stochastic Multi-Armed Bandits. *Journal of Computer Security (JCS)*, 2022. Accepted, to appear.

[6] R. Ciucanu, P. Lafourcade, G. Marcadet, and M. Soare. SAMBA: a generic framework for secure federated multi-armed bandits. In J. Artif. Intell. Res., 2022.

[7] O. Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[8] L. Guanyu, L. Feng, P. Weike, and M. Zhong. Fedrec: Federated recommendation with explicit feedback. *IEEE Intell. Syst.*, pages 21–30, 2021.

[9] F. M. Harper and J. A. Konstan. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19:1–19:19, 2016.

[10] A. Ben Horin and T. Tassa. Privacy preserving collaborative filtering by distributed mediation. In *RecSys '21: Fifteenth ACM Conference on Recommender Systems, Amsterdam, The Netherlands, 27 September 2021 - 1 October 2021*, pages 332–341, 2021.

[11] B. Joshi, F. Iutzeler, and M. Amini. Large-scale asynchronous distributed learning based on parameter exchanges. *Int. J. Data Sci. Anal.*, pages 222–232, 2018.

[12] P. Kairouz and H. B. McMahan. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.

[13] T. Li, L. Song, and C. Fragouli. Federated recommendation system via differential privacy. In *IEEE International Symposium on Information Theory, ISIT 2020, Los Angeles, CA, USA, June 21-26, 2020*, pages 2592–2597. IEEE, 2020.

[14] Y. Lin, P. Ren, Z. Chen, Z. Ren, D. Yu, J. Ma, M. de Rijke, and X. Cheng. Meta matrix factorization for federated rating predictions. In Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 981–990. ACM, 2020.

[15] L. Minto, M. Haller, B. Livshits, and H. Haddadi. Stronger privacy for federated collaborative filtering with implicit feedback. In *RecSys '21: Fifteenth ACM Conference on Recommender Systems, Amsterdam, The Netherlands, 27 September 2021 - 1 October 2021*, pages 342–350, 2021.

[16] T. Qi, F. Wu, C. Wu, Y. Huang, and X. Xie. Privacy-preserving news recommendation model learning. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1423–1432. Association for Computational Linguistics, 2020.

[17] L. Wang, Z. Huang, Q. Pei, and S. Wang. Federated CF: privacy-preserving collaborative filtering cross multiple datasets. In *2020 IEEE International Conference on Communications, ICC 2020, Dublin, Ireland, June 7-11, 2020*, pages 1–6. IEEE, 2020.