User privacy in collaborative filtering systems

Antoine Rault

ANNÉE 2016

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale Matisse

présentée par

Antoine Rault

préparée à l'unité de recherche Inria Rennes - Bretagne Atlantique
Institut national de recherche en informatique et en automatique
ISTIC

User Privacy in Collaborative Filtering Systems

Thèse soutenue à Rennes
le 23 juin 2016

devant le jury composé de :

Pascal FELBER
Professeur à l'Université de Neuchâtel / rapporteur

Arnaud LEGOUT
Chercheur à INRIA Sophia Antipolis / rapporteur

Ludovic MÉ
Professeur à CentraleSupélec / examinateur

Sébastien GAMBS
Professeur à l'Université du Québec à Montréal / examinateur

Patrick LOISEAU
Maître de conférences à EURECOM / examinateur

Anne-Marie KERMARREC
Directrice de recherche à INRIA Rennes / directrice de thèse

Davide FREY
Chercheur à INRIA Rennes / co-directeur de thèse

Contents

Chapter 1

Introduction

Recommendation systems (RSs) were first introduced in the 1990s as a solution to information overload. With the democratization of personal Internet access, and the beginning of e-commerce came an important surge in the amount of available content/information. This rendered any single individual unable to process information as fast as it is produced, thus prompting the creation of systems able to automatically filter information, such as recommendation systems. Nowadays, recommendation systems are not only ubiquitous in information systems, notably on the world wide web, but they are also a core mechanism in the way many companies do business [44]. For instance, Netflix heavily relies on recommendation to drive what the majority of its users watch since 2013 at least [112]. Similarly, Amazon saw a 29% increase in its sales after integrating a recommendation system [61]. Facebook uses several recommendation systems, their News Feed itself is a RS, and their App Center was built with personalization in mind, around another RS [2].

The two main reasons why many websites feature a recommendation system are: first, a recommendation system performs information filtering, thus enabling the visitors of its accompanying website to find relevant content faster. This is what appeals to users of recommendation systems. Second, a recommendation system increases user engagement through a personalized user experience of its accompanying website. More specifically, [101] says that a recommendation system increases the sales of an e-commerce website by converting people who just browse into buyers, by increasing cross-sell, *e.g.* when recommending items at checkout time based on the current shopping cart, and by increasing customer loyalty. Indeed, a customer who interacted extensively with an e-commerce's recommendation system will be more likely to come back to that same e-commerce because its RS knows the customer's interests. This is what appeals to website owners.

Generally, the task of a recommendation system is defined as follows: the system must automatically select a handful of items among a large pool of them. This selection is intended for a given user, and is constituted using the predicted relevance of selected items according to this user, based on her preferences. Prediction of items' relevance depends on the way the system infers users' preferences, and on the recommendation technique it uses.

Nowadays, the main recommendation techniques in use are collaborative filtering (CF), content-based, and hybrid ones, while the accepted categorization [93]

identifies three more types: demographic, knowledge-based, and social-based. CF techniques use the preferences of users with similar interests in order to make recommendations. Content-based systems extract semantic features from the description of items, then recommends items based on features for which users have expressed preferences. Hybrid techniques combine several of techniques to leverage the benefits and/or overcome the drawbacks of each individual technique. Demographic techniques are a rudimentary ancestor of recommendation systems which provide the same recommendations to all users within one demographic category. Knowledge-based techniques have users explicitly and formally describe their preferences, then compute which elements in their domain-specific knowledge base (*e.g.* traveling or automobile) are the best fitted. Social techniques leverage the preferences of the users' social network to provide recommendations.

Given that this thesis focuses on CF systems, for reasons detailed later in Section 1.1, we describe this type of techniques in more depth. At the core of collaborative filtering techniques is this rationale: like-minded people share similar preferences about items. It means that if users A and B share the same opinion on many items, it is likely that A will like an item i, unknown to her but liked by B. CF techniques leverage this rationale by basing their recommendations on the ratings given by other users. These techniques can be further divided in two categories: memory-based techniques, which keep the ratings as is, in memory, and model-based techniques, which summarize the ratings with a generative model.

On the one hand, recommendation systems obviously offer advantages to users and content providers alike. On the other hand, they are potential threats to the privacy of users, where some threats are to be expected while some others are more insidious. We will see that these are inherent threats, whatever the scale of RSs, and that the ubiquity of these systems exacerbates their privacy threats.

Any recommendation system inherently poses a potential privacy threat to its users because it aggregates preference data about each user. These data are private because they capture the interests of the corresponding user. It is obvious that preferences regarding religion, sexuality, or health issues are private and sensitive data in and of themselves. Less obviously so, preferences regarding any topic may also be considered as private data. Even the most innocuous-looking preferences like jokes or games are private data because they act as a quasi-identifier. A quasi-identifier [113] is a feature of an individual which, on its own, does not uniquely identify her, although when combined with other quasi-identifiers or public information, can yield a unique identifier. Thus the collection of a user's preferences, her profile, is increasingly likely to be a unique identifier as she incorporates more preferences. Moreover, it is not generally unachievable to guarantee the non-uniqueness of any subset of a profile because a typical recommendation system user's profile contains many preferences. For instance, even anonymization by generalization of preferences, *e.g.* converting 5-star ratings to like/dislike, is not not enough, if such a subset is revealed [3]. An illustration of an attack using quasi-identifiers is the famous re-identification of supposedly anonymized Netflix users by Narayanan and Shmatikov [81]. They use a dataset of anonymous Netflix users' movie ratings, and auxiliary information from the Internet Movie Database to re-identify some users as well as private information such as their political preferences.

The pervasiveness of recommendation systems exacerbates this inherent privacy threat they bring about. Individuals leave a few pieces of private information on each website or online service featuring a recommendation system they use.

An online service using a RS leads to privacy risks for users whether it uses its own RS or a third-party one, although the risks are different in the former and the latter case. When most online services implement their own separate recommendation system, users' information is scattered across a myriad of small-scale RSs. Although this situation has benefits such as compartmentalization of preferences by the topics covered by each service, it also means different RSs are diversely secured. Thus, the higher the number of small-scale recommendation systems containing information about a user, the higher the probability that one of them could suffer a privacy breach. When such a breach happens, users of the breached RS suffer from an increased privacy risk regardless of the confidentiality of the revealed information. Indeed, even if the revealed information is not sensitive, the adversary can use it as auxiliary information to bootstrap a targeted attack on the same users but in a different information system, *e.g.* another RS.

When most online services use a few large-scale third-party recommendation system providers, privacy risks for users do not automatically decrease because it concentrates private information. Despite the fact that large-scale RSs are probably better secured than small-scale ones, the reduced number of providers makes each one of them more attractive for adversaries because they concentrate much more private information.

1.1 Focus of this Thesis

We saw previously that recommendation systems, which can be implemented by different techniques, offer useful services, and at the same time, represent threats to privacy. Privacy issues in RSs being a vast domain, we focus in this thesis on user privacy aspects in collaborative-filtering systems, and notably the memory-based ones, for several reasons.

First, CF is the most popular technique to provide recommendations these days. It can recommend any type of content, its recommendations can be serendipitous, and they offer the highest precision of predicted ratings. Collaborative filtering can be divided in two subsets: memory-based techniques, and model-based ones. Although model-based techniques have the highest prediction precision, memory-based ones are simpler to implement and tweak, and they can easily provide explanations for the recommended items, which contribute more to users' satisfaction than greater precision of predicted ratings. Moreover, memory-based CF is particularly fit for decentralized architectures, which is a good point for user-privacy because it prevents threats by "Big Brother" adversaries.

Second, collaborative-filtering systems make for a privileged target for adversaries who want to obtain private data about users. Mathematically, since CF is the most widely used technique in commercial recommendation systems, this kind of systems is the most efficient choice for the adversaries we are concerned about. Furthermore, the collaborative aspect of CF makes it especially attractive for the

adversaries. Indeed, because the very rationale of collaborative filtering consists in exploiting some users' preferences in order to provide recommendations to another user, it means many parts of the CF process can potentially leak private user data. In comparison, the fact that user A gets recommended item i by a content-based recommendation system only tells her that the keywords or concepts extracted of i's description by the system match her interests. While if you consider the same fact but in a collaborative-filtering system, user A can potentially deduce some information about other users because such a system cannot recommend an item about which no one expressed any opinion.

We first discuss the threats (or attacks) from "Big Brother" adversaries in centralized systems, and those from adversaries we call "Little Brothers" in decentralized systems. Then, we discuss some other attacks of RSs, regardless of their architecture.

1.2 User Privacy in P2P User-based CF

Today, we observe that the overwhelming majority of commercial recommendation systems are centralized. This observation tells us that privacy of users of centralized RSs is under the threat of at least one type of adversary: "Big Brother" ones. Because the operator of a centralized RS is in control of users' data and/or the recommendation algorithms, users have no way to easily check that the operator operates her system as she claim she does, especially regarding potential privacy-preserving mechanisms. Moreover, even if the operator is honest and implements strong privacy-preserving mechanisms, she can be coerced into disabling or weakening them by entities such as intelligence agencies of the country they are incorporated in. Essentially, users of centralized RSs must put blind faith in these systems' operators, or stop using them altogether when it comes to privacy concerns.

One approach to avoid the threats of "Big Brother" adversaries is to use decentralized recommendation systems [29]. In decentralized RSs, the common feature is that no single entity in control of the system, but beyond that, the extent of decentralization can vary. For instance, system maintenance can be assigned to a small group of independent trusted entities, while the majority of participants (a.k.a. peers) simply run the protocol, as in the Tor network[1]. Alternatively, the system can be fully decentralized (a.k.a. peer-to-peer), meaning that all peers of the system are equal and that every peer is partly responsible for the system's maintenance. Consequently, peers usually manage their own data and their own computations. Decentralization not only makes "Big Brother" adversaries irrelevant, but it also makes it harder for a third-party adversary (*i.e.* either a user or an operator of the RS) to discover the private data of many users. memory-based collaborative filtering techniques, and particularly user-based ones lend themselves well to decentralized architectures because they generate recommendations using information which is local to the current user.

Recent years saw the emergence of peer-to-peer (P2P) recommendation systems based on fully-decentralized CF algorithms [16, 6]. These systems not only solve the "Big Brother" privacy threat as we just said, but they are also scalable, resilient

[1] `https://www.torproject.org`

to censorship, and fault-tolerant. P2P systems are scalable because distributing computations across peers makes it possible to compute recommendations without requiring huge servers or data centers. Furthermore, the more users, the more peers, which implies that the computational power of the system automatically grows with its number of users. P2P systems are resilient to censorship thanks to the combination of this two properties: anyone can join such a system, and no single entity controls an entire decentralized system. Although these systems are not entirely impossible to censor, when considering a powerful country-level adversary for instance, they are much harder to censor than centralized equivalents. Finally, P2P systems are fault-tolerant because they are designed from the ground-up on the assumption that any peer can leave the system or fail, at any time.

Although “Big Brother" adversaries do not exist in decentralized systems, performing collaborative filtering in a decentralized fashion is not free of privacy threats for users. Indeed, users must cooperate with each other to run the CF algorithm, but trusting all other users by default would be a mistake because anyone can participate in a decentralized system, including ill-intended users. We name “Little Brothers" this type of adversary, made of ill-intended but otherwise regular users of the system, leading to a prevalent threat to user privacy. Thus, it is necessary to adapt CF algorithms so that they strike a delicate trade-off between cooperating with others, and holding back some information to avoid unnecessarily leaking of users' data.

The main elements of decentralized user-based CF requiring specific care to avoid leakage of a user's privacy are her profile and her neighbors. Obviously, a user should reveal as little as possible of her profile, because it captures her interests which is private data. Regarding the neighbors of user A, they should not be freely available to any user, because when the neighbors of A are selected based on the similarity of their profiles, as is the case with user-based CF, knowing which users are similar to target user A enables an adversary to indirectly learn A's profile via the profiles of A's neighbors.

Preserving users' privacy implies making changes at several levels of the memory-based CF process. We divide this process into two logical steps. The first one consists in finding the neighbor users (respectively items) which are most similar to the current user (resp. item), a.k.a. the current user's K-Nearest-Neighbors (KNN). The second step consists in selecting the items which are unknown and the most relevant to the current user, among the candidate items from the KNN computed at the previous step. Privacy-preserving mechanisms addressing issues within each logical step deserve discussion on their own given the importance of both. In Chapter 3, we focus on privacy aspects at the first logical step, and propose our first contribution: *Hide & Share*, a similarity computation protocol which preserves the privacy of P2P CF users by avoiding that they reveal their profile, directly or indirectly via their neighbors.

1.3 Architecture-independent Attacks on CF Systems

We discussed above possible attacks on user privacy which affect CF systems depending on their architecture: attacks by “Big Brother" adversaries for centralized

systems, and attacks by "Little Brothers" adversaries for decentralized ones. But there also is a class of attacks on user privacy which is independent of systems architectures. We address in this thesis one type of attack from this class: Sybil attacks using auxiliary information. We study this type of attack because it is difficult to avoid since it exploits the very principle of CF, and because it was not yet well studied in the literature. Before describing this kind of Sybil attack, we describe other attacks from the same class which, although being beyond the scope of this thesis, are worth considering so that the reader may have a broader view of this class of architecture-independent attacks.

We illustrate how the feasibility of exploiting RSs for nefarious purposes is often under-estimated with the following attack. Calandrino *et al.* propose in [21] a passive attack on collaborative-filtering systems which show publicly lists of items related to the current item. The attack leverages these related-items lists, and some auxiliary information about a target user to guess whether changes over time within some related-items lists mean that the target user added or removed items in her profile. This attack is particularly insidious because it does not requires the adversary to directly interact with the collaborative-filtering system. Moreover, the only output of the collaborative-filtering system this attack uses is the public related-items lists of some items, which at first looks innocuous user-privacy-wise given that such lists only indicate that some items are correlated.

Alternatively, one can identify users interested in a specific topic by publishing a trap item in the recommendation system. Here, the adversary can be a user or a content provider depending on the system's policy regarding who can add new items. The goal of the adversary is to obtain an identifier of users (*e.g.* an IP address) who are interested in some topic (*e.g.* environmental activism). She can do so by adding to the system an item designed to be relevant to the target interest community, and by bundling with it a trap mechanism leaking identifying information. For instance, it could be a surreptitiously unique URL pointing to an adversary-controlled website within the item's description. Thus, if the target user clicks on this URL, the adversary can find the target's IP address by looking the unique URL up in the website's access log. The trap mechanism could also be a polyglot file, a file which is valid according to more than one file format such as an image one and an executable one, used as the item's picture. Let this polyglot picture be a valid executable too, it would include tracking code triggered when the picture is displayed. For instance, it can be an SVG file containing fingerprinting JavaScript code which a browser runs when trying to render the image. Both centralized and decentralized recommendation systems are vulnerable to this type of attack.

We now describe the Sybil attack which we address in this thesis. Sybil attacks consist in a single entity (*e.g.* an individual or a criminal organization) controlling several fake identities, each appearing as a different user to the attacked system. An adversary playing the role of a user in a collaborative-filtering system can mount a Sybil attack for different purposes. Since we are concerned with privacy-preservation in this thesis, we discuss a Sybil attack from [21] with the goal of tricking the recommendation algorithm into revealing the items from the profile of a target user. This type of attack requires the adversary to have some auxiliary information about the target user's interests, *i.e.* some items, and their ratings possibly, from the

target's profile, found using auxiliary channels. In the context of CF systems, the adversary may use auxiliary channels such as e-commerce reviews, or consumer protection organizations' forums. The adversary then creates just the right number of Sybil users, using the auxiliary information as their profile, so that, with high probability, the users who contribute recommendations for each Sybil are either another Sybil or the target user. Therefore, when this condition is met, any new item recommended to a Sybil comes from the target's profile. Because such an attack exploits the rationale of CF itself, it applies to centralized and decentralized systems alike.

A variation of the above attack is using colluding users instead of Sybil ones. Collusion attacks in general are similar to Sybil attacks, except that, instead of a single entity, several are in control of the different identities (users) in the system. This makes collusion attacks more robust because the colluding users can be made to look more like real users, thanks to the human beings behind the users. This kind of attack is also more complicated to foil than Sybil ones. For instance, let us consider a simple mitigation consisting in requiring a new user to pay a small amount of money in order to register to the system. The rationale is that registering one user is cheap, but registering many costs enough to deter attacks using many users. This works reasonably well against Sybil attacks because the small registration cost becomes more sizeable when multiplied by the number of Sybils, and this global cost is supported by the only entity behind the attack. However, this mitigation is less effective in deterring a collusion attack, especially one in which there is a different entity behind each colluding user, because the cost is spread between colluding entities. The main downside of collusion attacks is their increased complexity in comparison to Sybil attacks, due to involvement of entities and human beings with potentially different agendas, thus requiring more management and bargaining.

1.4 Contributions

1.4.1 *Hide & Share*

Our first contribution is *Hide & Share* (*H&S*), a novel mechanism for similarity computation which offers a reasonable level of privacy for users' profile. The problem *H&S* aims to solve is finding one's most similar users in a collaborative filtering peer-to-peer system without revealing too much of one's profile. This is a tricky issue because, on the one hand, users must share some information regarding their profile to assess their similarity. On the other hand, since anyone can join a P2P system, users desire to hide their profile from any unknown user. Moreover, the P2P setting leads a user to compute her similarity with many users before finding the most similar ones, so the similarity computation mechanism used needs to be lightweight.

H&S makes it possible to compute the KNN graph without requiring users to share their profile with anyone else, thanks to a simple observation: user-centric KNN applications such as recommendation do not require perfect knowledge. Based on this observation *H&S* trades off precision in the computation of similarity for privacy. This allows it to gain significant protection in terms of privacy with a minimal impact on applications like recommendation.

H&S's key contribution lies in a novel *landmark-based* approximation technique as well as in a fair landmark-generation protocol. The landmarks of our solution allow two users to indirectly measure their similarity by comparing their own profiles with a set of randomly generated profiles (the landmarks). The similarity between a user's profile and a landmark acts as a coordinate in a coordinates system. Users then exchange vectors of coordinates and compute an approximation of their actual similarity. This preserves user privacy as users do not exchange their full profiles and landmark coordinates only reveal a limited amount of information about a user.

Let A and B be users who compute their similarity for the first time via *H&S*. Before beginning the protocol, they each compute on their own a compact version of their profile. This compact profile is a Bloom filter, a probabilistic representation of a set which trades accuracy for memory-efficient, using a bit vector. Such a filter can return false positives, *i.e.* the filter can say an element is in the represented set while it actually is not, with a small customizable probability, but no false negatives, *i.e.* when a filter says it does not contain an element, it is always true. Then, A and B begin a series of bit-commitment schemes so as to jointly generate a random seed in a fair fashion. This seed and some publicly available parameters enable each user to independently compute the same set of landmarks, that is random bit vectors of the same size as compact profiles. In turn, A (B respectively) computes on her own the *Cosine* similarity of her compact profile with each landmark, and stores them in a vector of such similarity values, that we call her landmark coordinates. Finally, A and B exchange their respective landmark coordinates, enabling them to compute their final approximated similarity, which is the *Cosine* similarity of the two landmark coordinates vectors.

We evaluate *H&S* using real data traces in terms of recommendation quality, overhead, and empirical privacy protection. We also demonstrate formally its privacy guarantees by computing an upper bound on the amount of information leaked by *H&S*'s similarity approximation. The adversary model we consider when evaluating *H&S*'s privacy protection is a class of "Little Brothers" adversaries based on the semi-honest a.k.a. honest-but-curious model, augmented by some specific active abilities (*e.g.* attempt to bias multi-party computations) but unable to collude with other users or to create Sybil peers. Our results show that *H&S*'s KNN provides a reasonable trade-off between privacy and utility. *H&S* disturbs similarity values but it does not significantly hamper the quality of the resulting recommendations. Approximate similarity values constitute instead an asset towards privacy preservation as they effectively prevent adversaries from performing profile reconstruction attacks.

1.4.2 Sybil Attack Analysis & $2-step$

Collaborative filtering (CF) leverages the preferences of other users with similar interests to generate recommendations. In the case of user-based CF, recommended items are even directly taken from the profile of those similar users. Therefore, an adversary wishing to learn the content of a target user's profile can try to influence the recommendation system (RS) so that the recommendations she gets come mostly from the target's profile. We study a kind of attack which works as we just described,

and which influences the RS by creating several fake users (Sybils) sharing some common interests with the target. Given the feasibility of this kind of attack, and its applicability RSs regardless of their architecture, we propose a countermeasure: the $2-step$ similarity metric which makes the attack's success conditions harder to reach, while preserving recommendation quality for genuine users. In summary, our contribution is two-fold: we study the actual behavior of a Sybil attack on user privacy in CF, and we propose $2-step$ to mitigate this attack.

1.4.2.1 Sybil Attack Analysis

Although the Sybil attack which we consider (see Section 1.3 above for the precise description) was first conjectured in [21], the authors just describe it briefly and informally without evaluating it. Consequently, the first part of our contribution is a proper evaluation of this attack, showing that, if in theory the attack should be very successful, its actual performance strongly depends on the user population, and on the system's similarity measure. Our results, obtained using a state-of-the-art recommendation framework on three real datasets, show a strong correlation between the ability of a similarity measure to provide good recommendations, and its vulnerability to the attack.

During the evaluation, we notably observe that *Cos-overlap*, a variant of the *Cosine* similarity, significantly reduces the attack's success rate. This similarity measure considers that two users are perfectly similar as soon as they gave identical ratings to their common items, *i.e.* the items present in both profiles. Thus, according to this metric, two users whose profiles share only one item may appear perfectly similar simply because they have given the same rating to this common item. This low discriminatory power makes it difficult for Sybils to distinguish the target, and the other Sybils, from the target's perfectly similar alter egos. Therefore, the ability to prevent Sybil nodes from isolating their target is key in protecting the privacy of users.

While the above property constitutes an advantage for Sybil resilience, it clearly hinders the system's ability to provide good recommendations. Let us consider a user A, and its two perfectly similar alter egos B and C. A and B share the same rating for their only common item. However, A and C share the same rating for several common items. Clearly, C would be a better candidate than B to provide recommendations for A. But *Cos-overlap* considers B and C as equally good.

1.4.2.2 $2-step$

This observation allows us to propose the second part of our contribution: $2-step$, a new composite measure combining the recommendation quality of *Cosine* similarity with the resilience to the attack of *Cos-overlap*. Our results with $2-step$ combine a good $RMSE$ score, *i.e.* genuine users still get good recommendations, with a low success rate for the Sybil attackers, it mitigates the attack.

Let U and N be a user and a potential neighbor respectively. The first step uses a standard similarity measure with a good recommendation quality, *e.g. Cosine*, then modifies the output so that the potential neighbors with a standard similarity value above a given threshold appear identical for U. Contrarily to *Cos-overlap*, using a

threshold enables 2−*step* to coalesce the users who would probably provide recommendations of similar quality, making it hard for a Sybil to obtain a neighborhood containing the target user.

The second step of our measure attempts to distinguish the best potential neighbors among those with a standard similarity above the threshold, *i.e.* which ones would be the most suited to generate recommendations for U. The recommendation process consisting in finding relevant items in the profiles of U's neighbors, it implies that a neighbor without any item which does not already appear in U's profile does not bring anything. Rather, a good neighbor should have at least a few items which do not appear in U's profile. However, she should not have too many: users with too large a profile tend to provide less precise suggestions.

Thus, the second step differentiates between potential neighbors who exceed the threshold by taking into account the number of items in their profiles which do not appear in U's. Since the system generates neighborhoods for all users including Sybils, this heuristic has the beneficial effect of discouraging the presence of other Sybils in the neighborhood of a Sybil, thus making the attack even more difficult.

In a nutshell, the threshold makes it difficult for a Sybil to differentiate the target or another Sybil from other very similar users. The second step complements this feature by preferring genuine users to Sybils.

1.5 Thesis Outline

The remainder of this thesis is structured as follows: in Chapter 2, we discuss some of the relevant literature on recommendation techniques and architectures of recommendation systems, and on the possible attacks against user privacy, as well as privacy-preserving mechanisms in such systems. Then, we describe and evaluate *Hide & Share*, our first contribution in Chapter 3, and 2−*step*, our second contribution in Chapter 4. Finally, Chapter 5 summarizes our contributions while discussing the limitations of our approach, before discussing some opening questions.

Chapter 2

Privacy in Recommendation Systems

In this chapter, we first review recommendation systems (RSs) from the point of view of their technique, and their architecture. Then, we discuss the relatively small literature regarding privacy attacks on RSs, before covering the larger body of literature on privacy-preserving RSs.

2.1 Recommendation Systems

In this section, we give an overview of recommendation techniques before discussing more in depth collaborative filtering, the most successful technique nowadays, and we finish with RSs' architectures, that is centralized and decentralized systems mainly.

2.1.1 Recommendation Techniques

First, we touch on the whole spectrum of recommendation techniques, to provide the reader with a sense of the other available recommendation approaches beyond those we address in this thesis.

The accepted categorization of recommendation techniques [93] identifies six broad types: collaborative filtering, content-based, demographic, knowledge-based, social-based, and hybrid. Collaborative filtering (CF) techniques use the preferences of users with similar interests in order to make recommendations. Content-based systems extract semantic features from items then recommend items based on features for which users have expressed preferences. Demographic systems are a rudimentary ancestor of recommendation systems (RSs) which provide the same recommendations to all users of the same demographic category. Knowledge-based RSs have users explicitly express descriptions of their preferences, then compute which elements in their domain-specific knowledge base are the best fitted. Social RSs leverage the preferences of the users's social network to provide recommendations. Finally, hybrid systems combine several of the aforementioned techniques to leverage the benefits and/or overcome the drawbacks of each individual technique.

Nowadays, the main recommendation techniques in use are the content-based, collaborative filtering, and hybrid ones. We will therefore expand on these techniques in the following, after introducing the standard model of recommendation systems, and some notations.

2.1.1.1 Model & Notations

We describe below the standard way in which the literature models RSs: roles, items, ratings, user-item matrix.

The different generic roles in such systems are: user, operator, and content-provider. A user is the individual whose preferences are analyzed by the system, and who in consequence receives recommendations. An operator is an individual or entity who operates the system, and provides recommendations to users. A content-provider is an individual or entity who creates or introduce items in the system. Depending on the system, an individual may have one or more of these roles.

Item is the generic name for the elements being recommended by a RS. It may be a physical object (*e.g.* products sold by an e-commerce website), a place (*e.g.* restaurants), a webpage (*e.g.* news articles), or even a person (*e.g.* potential acquaintances on a social network).

A rating is the opinion of a user regarding an item, as understood by the recommendation system. It may consider various interactions made by a user as a rating. Obviously, one such interaction is when a user sends explicit feedback to the system, such as giving a 5-star rating. The other main interaction is implicit feedback such as when a user accesses or views an item, the system may infer that the user is interested in this item.

One last common term is the user-item rating matrix. This is the set of all ratings by any users on any item in the system, represented as a (usually sparse) matrix. Each entry $r_{A,i}$ is either empty, when user A did not rate item i yet, or contains the value of A's rating for i. Usually, a row corresponds to all the ratings of a user, while a column corresponds to all the ratings for an item.

2.1.1.2 Content-based Techniques

Content-based recommendation systems focus on describing items by their semantic features, then matching items with users' preferences for different features. These systems offer the advantages of directly explaining why they recommend a given item through its semantic features, and of being able to recommend new items which no one rated yet. However, they tend to lack serendipity, *i.e.* enjoyable and unexpected discoveries, and need items' description to be automatically parsable. This is not always reliably feasible *e.g.* when items are videos.

We can identify two types of content-based techniques, distinguished by their approach in semantic features extraction: the classic keyword-based one and the more recent concept-based one.

Keyword-based The classic approach to content-based recommendation relies on dividing the often unstructured textual description of an item into keywords, which are then weighted to determine the most significant ones in order to use them as the item's features. Recommending items to a user consists in matching the features she is interested in, with the items which fit best her set of features. A common framework [84] to compare the different keyword-based techniques is to look at (1) how they represent items, (2) how they represent users' interests (a.k.a. profile), and (3) how they match items and profiles.

The most convenient item representation is when items are stored as structured data, such as in a database. The attributes of a table naturally become the features of items. However, the most common situation is that items are usually accompanied only with unstructured data, typically text data *e.g.* a textual book review. The classic solution is to create a structured representation by dividing the text into words or short word sequences (terms), then counting all occurrences of words sharing the same root towards one term (stemming), and assigning weights to each stemmed term. The main function used to weigh the significance of a term within an item's description is the term frequency-inverse document frequency ($tf - idf$) measure [90].

The profile of a user, *i.e.* the data structure capturing her interests, is typically implemented using three types of representations. One type of representation is a history of the user's interactions with the system, *e.g.* the items she rated or the search queries she performed. Another type of representation is a list of features characterizing the items liked by the user. The last type of representation is a function which computes the probability that the user will like an item, based on a global description of that user's interests.

In the following, we introduce some of the three most popular types of recommendation generation algorithms.

Relevance feedback algorithms consist in incrementally refining the user's query, based on her feedback on the returned items. Queries and items are represented in a high dimensional vector space. Rocchio's algorithm [95] is one of the most widely used algorithm. These algorithms have the best classification performance compared to the two other, despite their lack of underlying theoretical model.

Nearest neighbors algorithms consist in classifying an item without rating by using the classes of its k nearest neighbors. Typically, these algorithms use the Euclidean distance or the *Cosine* similarity (which we define later, in Section 2.1.2.1) to determine neighbors. [12] uses such an algorithm to determine a user's short term interests then recommend news. The classification performance of these algorithms is only slightly lower than relevance feedback ones, but their main drawback is their long classification time.

Finally, the most computationally efficient type of algorithm is the Naïve Bayesian classifier. Despite not being as performant as the previous two types of algorithm in the general case, this classifier demonstrates good classification performance when probabilities are low. [83] applies them to recommendation of web sites.

Note that other types of algorithm exist such as rule-based ones [28], decision trees [60], or linear classifiers [71].

Semantic concept-based Recently, research focused on new approaches to overcome the limitations of classic keyword-based ones, such a inadequacy to some domains like poems. These new approaches are called concept-based as they extract semantic concepts from an item's description [35], while classic approaches use a syntactical analysis, extracting keywords. Concept-based approaches use techniques, notably from natural language processing, which allow them to use deep domain knowledge so as to gain in serendipity, and to improve their recommendation capabilities for specific domains such as jokes or poems. These approaches fit in either

the top-down or the bottom-up category.

Top-down approaches consist in supplementing the recommendation process with external structured knowledge, such as taxonomies, thesauri, or ontologies, to add linguistic and cultural knowledge when treating items' description. This allows such approaches to rebuild concepts associated with extracted keywords. Top-down approaches have the advantage of being transparent, meaning that the concepts used in such system are explicitly defined, thus allowing semantic similarity comparison. See [79] for instance.

Bottom-up approaches (a.k.a. distributional models) consist in representing words and items in a high-dimensional vector space, where a vector is learnt based on the context in which words are used. They learn the semantics of a word using unsupervised machine learning on large sets of texts. These approaches use the hypothesis that words occurring in the same context often have the same meaning. Bottom-up approaches do not require any human intervention, contrarily to top-down ones of which ontologies must be maintained. See [78] for example.

The advantages of content-based techniques are that they can generate recommendations independently for each user, they can recommend new items which no one rated yet, and their recommendations are directly explainable by the semantic features which triggered their recommendation. However, these techniques have troubles dealing with some domains like microblogs, jokes or poems, because their text is often short, and due to their nature, importance of words is not easily relatable to occurrence frequency. On a related note, content-based techniques have issues processing item descriptions which are not textual, *e.g.* audio or video. Moreover, they tend to lack serendipity. The above two limitations are in part mitigated by the semantic concept-based approaches. Finally, content-based recommendations tend to be unreliable when users have few ratings.

2.1.1.3 Collaborative Filtering Techniques

The rationale behind collaborative filtering (CF) lies on the following hypothesis: like-minded people share similar preferences about similar items, thus if they liked similar items in the past, they are also likely to like similar items in the future. In consequence, a CF system recommends items to user A based on what items users with interests close to A's liked. For instance, an item unknown to A, within those highly-rated by users with interests close to A's makes a good candidate for a relevant recommendation to A, if the hypothesis holds.

CF has the benefits of being content-agnostic because such systems do not require any knowledge about items themselves. Moreover, these techniques are naturally serendipitous because their recommendations are not based on any categorization of items, and they do not ask users to explicitly characterize their interests. Yet, CF suffers from the cold-start problem, *i.e.* it cannot recommend new items without ratings or make recommendations for new users who did not rate any items.

There are three broad families of CF techniques [37, 107]: model-based a.k.a. latent semantic analysis, memory-based a.k.a. neighborhood-based, and techniques mixing the two previous ones. We now give an overview of the available techniques in each family, and later, in Section 2.1.2, we will come back in details on the most

important techniques.

Model-based CF Model-based techniques, a.k.a. latent semantic analysis ones [37], use the initial user-item matrix of ratings only to build a model of users' behavior, which then predicts ratings. In layman's terms, latent semantic analysis consists in automatically inferring latent features about both items and users, from observed ratings. Such a system builds its model by learning the value of each parameter in the model's formula, by training itself using the user-item matrix.

Model-based CF is very scalable thanks to its dimensionality reduction aspect, and can generate recommendations quickly once the model is trained. Moreover, the most accurate of all recommendation techniques, in terms of predicted rating, are model-based CF techniques. They can also detect and represent very specific relationships between items, which would probably be overlooked by a non-expert human. However, the mathematics underlying these techniques are complex, which makes the tuning of the model's parameters tricky, and its building phase computationally expensive.

There are four categories of techniques implementing latent semantic analysis: matrix factorization ones, fully probabilistic ones, probabilistic matrix factorization ones, and other techniques from the machine learning and artificial intelligence (AI) communities.

Matrix factorization techniques are the most popular model-based ones. They look for hidden relationships between items in order to explain observed ratings. They use linear algebraic transformations to factor the user-item rating matrix into compact predictive models of user preferences. These models represent both users and items in the same lower dimensionality space of latent factors. We cover these techniques more deeply in Section 2.1.2.2.

Fully probabilistic techniques consider the prediction of a user's rating for an item as a probability problem. Therefore, they build various models based on estimating conditional probabilities such as $P(i|A)$, the probability that item i is liked by user A. Cross-sell [65] and Personality Diagnosis [85] are examples of such techniques.

Probabilistic matrix factorization techniques (a.k.a. PLSA for probabilistic latent semantic analysis) are similar to matrix factorization except for the underlying mathematical theory: the former is based on probability theory while the latter is based on linear algebra. In PLSA, ratings are modeled as probabilities like $P(i|A)$, which are defined using latent factors f:

$$P(i|A) = \sum_f P(i|f) \times P(f|A)$$

Examples of probabilistic matrix factorization techniques include latent Dirichlet allocation [32] or factor analysis [22].

Other techniques coming from the machine learning or AI communities include artificial neural networks [96], Markov decision processes [50], Bayesian networks [108], and cluster models [25].

Memory-based CF Memory-based techniques were the first implementations of collaborative filtering in the 1990s [45, 92]. They are also called neighorhood-based

techniques because they gather users or items which are closest in terms of similarity, thus the term "neighbors", into groups (neighborhoods). Memory-based CF techniques keep the user-item matrix as-is, in memory.

These techniques are simple to implement, more serendipitous than model-based ones, and can easily provide explanations for their recommendations which is often more important to end-users than predicted rating accuracy [111]. However, centralized systems using memory-based CF have troubles scaling because the memory and computational requirements grow linearly with the number of users and items. That being said, the user-based techniques are some of the few recommendation techniques to be easily decentralizable, therefore alleviating the previous limitation.

Memory-based CF systems generate recommendations in two main logical steps: (1) find the most similar users/items, (2) select recommendations from the candidate set of items derived from the most similar users/items. Each step is implemented slightly differently depending on whether the particular technique in use is user-based or item-based.

The first logical step in memory-based CF consists in finding the users (resp. items) which are most similar to the current user (resp. item). This is the classic K-Nearest-Neighbors (KNN) search, that is to find the k closest elements to a given element (*i.e.* a user or an item) according to a similarity metric, among all the elements in the system. Regarding the similarity metric, we discuss commonly used ones in Section 2.1.2.1. The brute-force approach to solving a KNN search is to exhaustively compare the similarity of the current element with every other element in the system. This approach gives the best possible set of k nearest neighbors but does not scale well with increasing number of elements. For obvious complexity reasons, the search can be enhanced with Approximate Nearest Neighbor (ANN) search, *i.e.* compare the current element with only a sample of all the elements, or with periodical offline updates, *i.e.* instead of doing a KNN search every time a recommendation must be generated (online), maintain a table of KNN for each element which is only updated periodically (offline).

The second logical step in memory-based collaborative filtering is to select the items which are unknown and the most relevant to the current user, among the candidate items from the KNN computed at the previous logical step. Which items qualify as candidate items depends on whether the RS uses an item-based or a user-based KNN. In the item-based case, candidate items are the neighbor items of each item from the current user's profile. In the user-based case, candidate items are the items from the profile of each neighbor user of the current user. Then, the main way to select the most relevant items is top-N recommendation, that is ranking the candidate items, then recommending the N highest ranked items. Ranking is done by computation of a score for each item, such as the average of the item's ratings, weighted by the similarity value of the users/items associated with the ratings.

There also exist a few works fusing user-based and item-based approaches [116, 114].

Memory-and-model-based CF Given that memory-based and model-based techniques have different benefits/limitations, it is only logical that researchers propose systems relying on both types of techniques to leverage complementary advantages

or to mitigate the drawbacks of each.

We review below examples of such systems.

In [66], based on the observation that memory-based and matrix factorization techniques leverage different types of information from the user-item matrix, Koren proposes a system combing a neighborhood-based technique reformulated as a formal model, with the SVD++ matrix factorization technique. The resulting technique considers information from neighborhoods and latent factors of equal importance, rather than post-processing the result of one source with the other's. In order to fuse both base techniques, Koren expresses the memory-based one in a formalism similar to the one used for matrix factorization. A notable limitation of this system is that the reformulated of the memory-based approach makes it loose some of its benefits, such as explainable recommendations, or its applicability to decentralized architectures.

In the case of the RS of Google News [31], the main motivations for using a memory-and-model-based collaborative-filtering system are the scale of their system (millions of users and items), a high rate of item churn, and a strict requirement on the time it takes to generate recommendations (a few hundreds of milliseconds). For these reasons, the authors combine two model-based techniques (probabilistic clustering, namely MinHash, and probabilistic latent semantic analysis) for long-term trends, and a memory-based technique (item covisitation) for short-term trends. Google News' recommendation system runs independently each of the three techniques then ranks the candidates items via a linear combination of the scores assigned by each technique.

Regarding the three base techniques, the authors adapt the probabilistic latent semantic analysis one from [54] to cope with datasets as dynamic as theirs. They use MinHash as a Locality Sensitive Hashing (LSH) technique to quickly compute the similarity of two users. Finally, item covisitation is a graph of items, where two items (nodes) are linked by an edge when at least one user accessed both items within a given timeframe.

The main limitation of this RS is that it does not merge memory-based and model-based approaches into a single algorithm. Therefore, the weights assigned to each technique is determined empirically, and cannot be explained.

2.1.1.4 Hybrid Recommendation Techniques

Hybrid recommendation systems use complementary techniques, primarily collaborative filtering and content-based ones, in order to provide better recommendations. We now discuss three such systems.

[4] describes the RS of TiVo, which runs two recommendation techniques independently, a CF one and a content-based one, then merges their output in a single ordered list of recommended items (TV shows). The CF technique is a classic item-based one, and the content-based one is a Bayesian classifier using shows' genre and cast. The authors use the content-based technique only to compensate for the other's cold-start issue, as they observed CF provides better recommendations n other situations. The output of each technique is a list of items accompanied by a predicted rating, and a confidence score in $[0, 255]$ for the predicted rating. The

CF technique produces confidence scores in the upper half of the range, while the content-based one produces confidence scores in the lower half of the range. Both outputs are merged and ordered by decreasing predicted rating, then by decreasing confidence score.

Fab [5] proposes an hybrid RS mixing content-based and CF techniques too. Contrarily to [4], Fab's algorithm is a merger of the two techniques. User profiles are represented using a content-based approach, that is a collection of terms weighted according to the current user's interest in them. These profiles serve for content-based recommendation, by matching terms and weights between profiles and items' description. They are also used for similarity computation in selecting the user's neighbor, as in user-based CF. Finally, users get the best recommendations, independently of the technique which generated them.

Finally, the authors of [74] blend CF and social recommendation in order to overcome typical weaknesses of the former such as cold-start and data sparsity issues. They propose a classic user-based, memory-based collaborative-filtering system, except that neighborhoods are not determined using a usual similarity metric. Instead, the system uses an existing Web of Trust, *i.e.* a directed graph linking users with a trust value, and a trust propagation algorithm. This introduces a trade-off between accuracy and coverage of recommendations, depending on the trust propagation horizon. Additionally, the proposed system is resilient to Sybil attacks thanks to its use of trust relationships.

2.1.2 Collaborative Filtering in Details

Collaborative filtering (CF) being the most prominent family of recommendation techniques in both academia and industry, we explain further in details the three most popular techniques: matrix factorization, item-based approaches, and user-based ones.

2.1.2.1 Preliminaries

We first introduce some similarity and recommendation quality metrics which are commonly used throughout the literature.

Similarity metrics Similarity metrics are an important element of RSs, especially in CF ones as their recommendations are based on most similar users or items, in terms of interests. We cover four classic metrics: the well-known *Cosine* similarity metric, the *Jaccard* index, the *Pearson* correlation coefficient [110], and the *Spearman*'s rank correlation coefficient.

Originating from the field of information retrieval [97], the *Cosine* similarity reflects the similarity between two users A and N by measuring the *Cosine* of the angle between their profiles r_A and r_N (i.e. vectors of tuples representing the item and its associated rating).

$$Cosine(A, N) = \frac{r_A \cdot r_N}{\|r_A\| \|r_N\|}$$

The *Jaccard* index, in turn, is defined as the size of the intersection divided by the size of the union of two profiles, regardless of the rating associated to items, making this metric computationally efficient.

$$Jaccard(A, N) = \frac{|r_A \cap r_N|}{|r_A \cup r_N|}$$

The *Pearson* correlation coefficient consists of the covariance of the two profiles divided by the product of their standard deviations. It is equivalent to the *Cosine* similarity applied to the profiles obtained after centering the ratings around their average.

$$Pearson(A, N) = \frac{cov(r_A, r_N)}{\sigma_A \times \sigma_N} = \frac{\sum\limits_{i \in I_{AN}} (r_{A,i} - \overline{r_A})(r_{N,i} - \overline{r_N})}{\sqrt{\sum\limits_{i \in I_{AN}} (r_{A,i} - \overline{r_A})^2 \sum\limits_{i \in I_{AN}} (r_{N,i} - \overline{r_N})^2}}$$

where $\overline{r_X}$ is the average rating of user X over all the items she rated, and I_{AN} is the set of items rated by both A and N.

Spearman's rank correlation coefficient is equivalent to *Pearson* applied to the ascending rank of ratings R, instead of the ratings directly.

$$Spearman(A, N) = \frac{\sum_i (R_{A,i} - \overline{R_A})(R_{N,i} - \overline{R_N})}{\sigma_A \times \sigma_N}$$

where $R_{X,i}$ is the rank of $r_{X,i}$ among all ratings in X's profile.

The *Pearson* and *Spearman* coefficients produce outcomes in $[-1, 1]$, while the other metrics give values in $[0, 1]$. There is not a single best metric for any situation, rather the similarity metric which models best users' relationship depends on the dataset considered.

Recommendation quality metrics Another equally important element of a RS's design is the metrics used to evaluate the quality of generated recommendations. We present the most popular ones, namely $RMSE$, MAE, *Precision*, *Recall*, $F1$-*measure*, *Coverage*, *ROC* curve, and *AUC*.

One of the most often used metrics is Root Mean Square Error ($RMSE$). It measures how close predicted ratings for recommended items, computed using a training subset of a dataset, are to the actual ratings of items in the remaining testing subset of the same dataset. It is defined as below for each user A:

$$RMSE(A) = \sqrt{\frac{\sum_{i=1}^{n} (pred_{A,i} - r_{A,i})^2}{n}}$$

where, n is the number of items rated by A in the testing set, $pred_{A,i}$ is the predicted value of user A's rating for item i, and $r_{A,i}$ refers to A's actual rating on item i.

A related metric is Mean Absolute Error (MAE). It is similar to $RMSE$, except that it does not square differences between predicted and actual ratings, thus

penalizing less harshly large prediction error, *e.g.* predicting a rating of 5 while the actual one is 1. MAE of user A is defined as follows:

$$MAE(A) = \frac{\sum_{i=1}^{n} |pred_{A,i} - r_{A,i}|}{n}$$

Precision and *Recall* [94] are complementary metrics evaluating the quality of a set of recommendations. The former evaluates whether the recipient user likes the recommendations. The latter evaluates if the recommendations cover all the interests of the recipient, expressed by the ground truth ratings in the testing set. Here are the definitions of *Precision* and *Recall* of the set of recommendations IS for user A:

$$Precision(A, IS) = \frac{|IS \cap I_A|}{|IS|}$$

$$Recall(A, IS) = \frac{|IS \cap I_A|}{|I_A|}$$

where I_A is the set of items liked by A in the testing set. When ratings are not binary, these metrics require to define a threshold value determining which ratings to consider as a like.

$F1$-*measure* is the harmonic mean of *Precision* and *Recall*. It summarizes in a single value the quality of a set of recommendations. The $F1$-*measure* of the set of recommendations IS for user A is:

$$F1\text{-}measure(A, IS) = \frac{2 \times Precision_{A,IS} \times Recall_{A,IS}}{Precision_{A,IS} + Recall_{A,IS}}$$

Coverage is the percentage of all the items that a recommendation system can recommend. It is measured by the percentage of items for which the system can predict a rating when requested by a user.

The *Receiver Operating Characteristic* (*ROC*) curve and the related *Area Under the (ROC) Curve* (*AUC*) [39] evaluate recommendation quality by measuring the evolution of the true positive and the false positive rates depending on the threshold score above which an item gets recommended. A recommendation is a true positive when the recipient likes it, else it is a false positive. The *ROC* curve plots the true positive rate on the Y-axis, and the false positive rate on the X-axis. The *AUC* sums up this curve into a numerical value: the fraction of the area of the unit square which is under the *ROC* curve. Therefore, recommending random items yields an *AUC* of 0.5, while a perfect recommendation system would yield an *AUC* of 1.

2.1.2.2 Matrix Factorization

Matrix factorization techniques, a.k.a. dimensionality reduction ones, are the most prominent model-based ones nowadays. This is due to their accuracy and scalability. This is especially true since the start of the Netflix prize competition in 2006 which brought a lot of attention to research on RSs. We concisely introduce the rationale of these techniques before explaining the two main ones.

Matrix factorization techniques look for hidden relationships between items in order to explain observed ratings. They represent both users and items with a vector

m in the same lower dimensionality space of latent factors. Such vector means either how much each factor characterizes the represented item, or how much each factor is interesting to the represented user. In consequence, a rating in this space is the dot product of the vectors of a user and an item:

$$m_{A,i} = m_A \cdot m_i$$

Factors can sometimes be mapped to semantic features, but it is sometimes impossible as some mathematical relationships have no corresponding semantic interpretation.

We describe further how singular value decomposition (SVD) and principal component analysis (PCA), two popular matrix factorization techniques, are applied to collaborative filtering.

Singular value decomposition The first application of SVD to CF is [99] in 2000. However, the original formulation of SVD in linear algebra is undefined on a matrix with missing entries such as the user-item matrix at hand. The authors of [99] work around this issue by filling the missing entries with naive non-personalized predicted ratings. In this case, they use the average rating of an item i after each user X's rating $r_{X,i}$ have been normalized by X's average rating $\overline{r_X}$.

A more scalable and more accurate approach is to keep the user-item matrix as is, and use regularization when solving the associated least squares problem. This is the approach taken in [66], in which the predictor of A's rating for i $pred_{A,i}$ is:

$$pred_{A,i} = \overline{r} + bias_i + bias_A + m_A \cdot m_i$$

where $\overline{r}$ is the average rating over all users and items, $bias_i$ and $bias_A$ are scalar parameters describing the observed biases in use of the rating scale for item i and user A respectively. In order to determine the value of $bias_i$, $bias_A$, m_A, and m_i, one solves the following least squares problem:

$$\min_{bias_*, m_*} \sum_{r_{A,i}} (r_{A,i} - \overline{r} - bias_i - bias_A - m_A \cdot m_i)^2 + \lambda(bias_i^2 + bias_A^2 + \|m_A\|^2 + \|m_i\|^2)$$

where the last term is for regularization so as to avoid overfitting the parameters to the training data. Parameter λ controls the extent of this regularization and its value is determined empirically.

Note that it is possible to obtain even better results by using more regularization parameters, *e.g.* one parameter for biases and another one for factor vectors, or one per parameter to be learned. However, doing so also increases the cost of the training phase, and the space of parameters to explore in order to fine-tune such RS.

Least squares problems are usually solved by stochastic gradient descent or alternating least squares. The former approximates a gradient descent by solving the problem with some initial parameter values, then iterating over it with small parameters changes until it converges. The latter works by dividing the minimization problem into two efficiently solvable ones. In the context of a RS, it alternates between fixing the value of user profiles m_A to solve the item profiles m_i, and fixing the opposite problem. Although alternating least squares yields an optimal

solution, stochastic gradient descent leads to the same solution faster. The most popular stochastic gradient descent method is from [105]. It is an iterative algorithm which makes predictions for all known rating, starting with arbitrary values for all parameters. Then, at each iteration, the value of each parameter is updated depending on the prediction error and the learning rate, a meta-parameter typically taking small values, *e.g.* in $[0, 0.1]$. This process goes on until it converges and the prediction error becomes stable.

The aforementioned model can be enhanced further by integrating other aspects, such as which items users rated (SVD++), or temporal changes in items popularity and user preferences [67].

Principal component analysis Another commonly used matrix factorization technique is principal component analysis (PCA), which computes principal components (eigenvectors) representing users in a lower dimensional vector space than the original user-item matrix. PCA is related to SVD as their formulae can be written so as to highlight the fact that some eigenvector are equal to some singular vectors.

[46] proposes the Eigentaste algorithm. It does a PCA on the gauge set, a small, normalized (z-scores) subset of the user-item matrix made dense by requiring each user to rate all items in the gauge set. It then projects users in the vector space defined by the top k eigenvectors, and divide this space into clusters. Thus, all users projected within the same cluster are considered as neighbors, which enables to recommend items to a user based on the items rated by her neighbors.

SVD and PCA being mathematically related, they suffer from the same types of limitations. Namely, they are less serendipitous than memory-based technique, they may miss hyper-local relationships between some items, and most importantly, they require a lot of time and computations to train.

2.1.2.3 Item-based CF

We now turn to the memory-based techniques which rely on neighborhoods of items, a.k.a. item-based collaborative filtering ones.

In the late 1990's and early 2000's, academia [63, 100] and industry (Amazon [73]) pioneered item-based CF systems, a variant of memory-based CF, which rely at their core on similarity between items rather than between users, as it was the norm until then. This increases scalability because it shifts the bottleneck to the growth in number of items, and item similarities tend to be more stable in time than user ones.

Generally speaking, item-based CF works as follows. Such systems maintain two data structures in memory: the usual user-item matrix, and an item-KNN table which associates an item with the list of its k most similar items.

It is the maintenance of this second data structure which implements the first logical step of memory-based CF, *i.e.* forming item neighborhoods. Like user similarity, the similarity of two items is determined with a similarity metric and their profile, the collection of ratings r_i for item i by any user. In other words, user profiles are the rows of the user-item matrix, while item profiles are its columns. While the RS updates the user-item matrix online, it builds and updates the item-KNN table

only periodically, because it is the most computationally expensive phase. This does not significantly hamper recommendation quality thanks to the stability over time of item similarities, which yields quite static KNN lists for most items [100].

In the second logical step of memory-based CF, the system gathers the candidate items by aggregating the KNN items associated with each item from the current user's profile. This is efficiently done online because most users' profile contain few rated items.

[100] evaluates item-based CF with several similarity metrics, and rating predictors. They found that the best performing similarity metric is *AdjCosine*, a variant of *Cosine* with normalization of raw ratings using deviation-from-mean as in *Pearson*. The best performing rating predictor consists in computing the score of an item i for user A, using the sum of ratings of i's KNN by A, weighted by the similarity between i and each of its neighbor items.

$$pred_{A,i} = \frac{\sum_{j \in \text{KNN}_i} W_{i,j} \times r_{A,j}}{\sum_{j \in \text{KNN}_i} |W_{i,j}|}$$

where KNN_i is the set of the k most similar items to i, and $W_{i,j}$ is the similarity between items i and j.

Alternatively, [63] proposes to use an asymmetric similarity metric based on conditional-probability, when ratings are binary. This metric corresponds to the probability of liking item j given that item i is already rated, while avoiding over-emphasis on popular items, and balancing the contribution of user who rated many items because their ratings are less reliable than focused users with few ratings.

The author reports that *SimProba* consistently yields a 3% to 5% higher *Recall* than *Cosine*, for 10 recommendations, across all the five datasets used. A limitation of this evaluation is that *Precision* is not reported though.

Later, Deshpande and Karypis propose in [33] some item-based techniques described as widely used ten years later [114]. The main contribution of [33] compared to [63] is an interpolated higher-order item-based CF technique. Its rationale is to account for the probability of several items being rated in a same user profile, in addition to considering each item from the profile on its own when building item-neighborhoods.

In practical terms, this technique is similar to the generic one described above, except that, during the first logical step of CF, it also computes a KNN item-list for some combination of items (called an itemset) from the current user's profile. To avoid computing all possible itemsets, this technique limits itself to those smaller than a parameter-fixed size, and those appearing in at least a threshold percentage of all user profiles. Then in the second logical step of CF, this technique performs an interpolation of the item recommended for each size of itemset.

The authors find that the higher-order technique performs as well, or better than the generic item-based one, and a user-based one, in terms of recommendation quality measured by hit-rate and average reciprocal hit-rank.

2.1.2.4 User-based CF

Finally, we discuss user-based collaborative filtering, the first and most straightforward type of memory-based CF techniques. Indeed, it is an immediate implementation of CF's rationale: recommend items to user A based on what items users with interests close to A's liked. Therefore, user-based techniques find A's neighbors, pick candidate items from those neighbors' profile, and recommend some of these items to A, based on the ranking produced by a predictor. A predictor is a function which attribute a score to a candidate item i, usually thanks to the similarity values between A and her neighbors N, as well as the rating of i by each N.

Basic technique User-based CF was introduced with the GroupLens distributed Usenet RS [92] in 1994. It provides recommendations by adding recommendation servers (called "Better Bit Bureaus" in the paper) to Usenet's architecture. Recommendation servers store users' ratings about news, and predict missing ratings, though ratings from one newsgroup do not contribute to predictions for other newsgroups.

To predict missing ratings within a newsgroup for A, a recommendation server computes the *Pearson* coefficient $Pearson_{A,N}$ (although other similarity metrics can be used) between A and each other user N based on their rating histories, a.k.a. her profile (first step of memory-based CF). The server then uses these coefficients as weights for the other users' ratings in the following predictor function, predicting here the rating of i by A:

$$r_{A,i} = \overline{r_A} + \frac{\sum_{N \in \text{users}} (r_{N,i} - \overline{r_N}) \times Pearson_{A,N}}{\sum_{N \in \text{users}} |Pearson_{A,N}|}$$

where $\overline{r_X}$ is the mean rating of user X across all the items of her profile.

Basically, the predictor takes the mean rating of A, and adds the sum of ratings for i by each N, after taking into account N's use of the rating scale as well as weighing her contribution accordingly to the correlation of her profile with A's. Although GroupLens does not maintain a proper KNN structure, its use of *Pearson* coefficients in the predictor is similar to a KNN where k is equal to the total number of users.

Advanced predictor The authors of [19] propose a predictor based on the previous one, using more advanced weighting of other users' ratings. The three weighting differences of the proposed predictor are: increased significance of the similarity of profiles with few common items, reduced contribution of universally liked items, and increased importance given to closest neighbors.

In order to increase the significance of the similarity of two profiles sharing few items, the authors assign a default rating to items rated in only one of the two profiles. This notably enables a metric like *Pearson* to give more discriminative similarity values when ratings are binary.

Universally liked items are not useful to provide personalized recommendations, thus their contribution to two users' similarity should be small. The authors suggest

to adapt metrics, such as *Cosine* or *Pearson*, by incorporating the Inverse User Frequency (IUF), adapted from information retrieval's Inverse Document Frequency (IDF).

Increasing the importance given to ratings from the closest neighbors of the current user is done by exponentiating of a factor ρ their weight, *i.e.* their similarity regardless of the metric used.

The resulting predictor, illustrated with A's rating for i, is:

$$r_{A,i} = \overline{r_A} + \frac{\sum_{N \in \text{users}} (r_{N,i} - \overline{r_N}) \times W'_{A,N}}{\sum_{N \in \text{users}} |W'_{A,N}|}$$

where $W'_{A,N}$ is the exponentiated similarity of users A and N, defined as:

$$W'_{A,N} = \begin{cases} W^{\rho}_{A,N} & \text{if } W_{A,N} \geq 0 \\ -(-W^{\rho}_{A,N}) & \text{if } W_{A,N} < 0 \end{cases}$$

where, in turn, $W_{A,N}$ is a similarity metric incorporating IUF. For instance, $Pearson_{IUF}$ is defined as:

$$Pearson_{IUF}(A,N) = \frac{1}{\sigma_{IUF}(r_A) \times \sigma_{IUF}(r_N)} \times \sum_i IUF_i \times \left(\sum_i IUF_i \times r_{A,i} \times r_{N,i} - (\sum_i IUF_i \times r_{A,i})(\sum_i IUF_i \times r_{N,i}) \right)$$

Components of user-based CF In [52], the GroupLens research group proposes a decomposition of user-based CF into five components, and empirically evaluates the approaches for each component, in order to provide guidelines for the design of CF systems.

The five identified components are: user-similarity metrics, user-similarity significance weighting, item rating variance weighting, neighbor selection, and rating normalization.

The first component, *similarity metrics*, defines how one quantifies the extent of two users' similarity. The authors consider classic metrics (*Pearson*, *Spearman*, and *Cosine*) and less common ones: the uncertainty coefficient [118], and the mean-squared difference (MSD) dissimilarity metric used in the Ringo system [104]. The experiments show that the metrics producing the best recommendation quality are *Spearman* and *Pearson* (equally perfomant), although the former requires less assumptions on the data.

The second component, *significance weighting*, associates a weight in $[0, 1]$ to a similarity value in order to quantify the representativity of this similarity. The insight behind weighting of similarity significance is that high similarity values for users sharing a small set of common rated items is not representative. The authors suggest to lessen the contribution of similarity values for users who share less than $thresh$ rated items. The experiments show a significant improvement of recommendation quality with $thresh = 50$.

The third component, *variance weighting*, modifies the impact of ratings for items about which almost all users agree upon, regardless of whether they like these items or not. The insight behind weighting of item rating variance is that the agreement of users A and N about item i's rating does not help asserting A and N have similar interests in general. The authors evaluate one variance weighting technique by incorporating item rating variance in the *Pearson* formula, and representing ratings as standard scores (a.k.a. z-scores) $z_{A,i} = \frac{r_{A,i}-\mu}{\sigma}$ of mean $\mu = 0$, and standard deviation $\sigma = 1$. However, the experiments show that this particular technique has no significant effect on recommendation quality.

The fourth component, *neighbor selection*, defines the criterion to decide which users to use as neighbors to generate recommendations. The considered techniques are: (1) selecting all users above a given similarity threshold, (2) top-N, *i.e.* selecting the N most similar users, and (3) a combination of both. The top-N is the best technique with excellent coverage, and good recommendation accuracy.

The last component, *rating normalization*, defines how to put every neighbor's ratings on a common ground when using these ratings to predict an item's score. The considered techniques are: (1) no normalization, just a similarity-weighted average of neighors' ratings, (2) deviation-from-mean, to account for users' different uses of the rating scale as in [92], and (3) z-score-based weighted average, to account for users' different spread of ratings. The experiments show that deviation-from-mean is the way to go, because it significantly improves upon no normalization, and the z-scores-based technique performs as well as deviation-from-mean but is not as simple.

Overall, the authors of [52] give the following design guidelines for user-based collaborative-filtering systems. Regarding the similarity metric component, the choice depends on the data's rating scale. For short & discrete rating scales (*e.g.* up to $[1, 20]$), they recommend using *Spearman*, else they recommend using *Pearson* for continuous rating scales. As for the rest, they recommend the best performing techniques for each component, namely: significance weighting, no variance weighting, top-N neighbor selection, and deviation-from-mean rating normalization.

Alternative approaches We presented above the dominant approaches for user-based CF, but other approaches exist. For example, [98] proposes other techniques for each logical step of memory-based CF. For the first logical step, the authors propose a centroids-based method to form user neighborhoods. For the second logical step, they propose to use association rules to select candidate and recommended items. However, evaluation shows that these two new techniques perform similarly or worse than traditional alternatives.

User-based CF is simple to implement, including with a decentralized architecture, and is computationally efficient. It also is the most serendipitous type of memory-based technique. However, ratings predicted by user-based techniques are not as accurate as item-based ones or as matrix factorization. User-based techniques also less able to recommend some items (*i.e.* smaller coverage) than the two aforementioned ones because some items may have too few ratings.

2.1.3 Architectures of Recommendation Systems

In the previous section, we covered recommendation systems from the point of view of their recommendation technique. In this section, we look at the architecture of these systems, and especially at the different approaches to implement distributed RSs.

The main dichotomy in terms of architecture is centralized versus distributed RSs. The overwhelming majority of industrial-grade RSs are centralized because their operators have virtually no economic incentive to use distributed ones. Distributed RSs improve scalability, and make it easier to include some privacy-preserving mechanisms for users. Despite these scalability and privacy benefits, RSs operators still prefer centralized solutions because they require less engineering, provide more accurate recommendations, and because they gain no business advantage from relinquishing some control over users' data.

We cover briefly centralized systems as well as systems using architectures in-between centralized and distributed ones. Then, we discuss in more details distributed systems.

2.1.3.1 Centralized Systems

We consider centralized any system running either on a single powerful computer or on several machines within a datacenter because, even though they are built upon decentralized tools such as MapReduce, the recommendation algorithm itself is not significantly different from a desktop implementation. In centralized systems, there is a central server controlled by the RS operator, and clients which may be users, or content providers, or both. A standard centralized RS only asks users and content providers for their inputs, *i.e.* ratings of the former and items of the latter, then takes care of all the recommendation process as well as of storage of user preferences. See [47, 73, 31] for examples.

In terms of user privacy, centralized systems are far from ideal. First of all, they often store user preferences without other privacy guarantees than legal ones, which vary widely across countries. One of the many partial solutions available is for preferences to be encrypted, but this implies that either the operator has the decryption key, or that homomorphic encryption is used. In the former case, this only shift the problem to the trust users put in the RS operator, while the latter case is unlikely to be chosen for commercial systems given that current homomorphic cryptosystems are very computationally expensive. Finally, whatever the centralized systems, there always is the threat of the "Big Brother" adversary. Because the RS operator fully controls the system, users can only hope that she is not a "Big Brother", or that she is not coerced into revealing the system's private data.

2.1.3.2 Hybrid Architectures

Even though RSs are typically either centralized or distributed, there is a few systems using a hybrid architecture. The extent of hybridization can vary, but the common idea is to keep a centralized architecture while distributing some parts of the recommendation process, typically the most computationally-intensive tasks.

[17] proposes HyRec, a user-based CF system which offloads to its users' browser the tasks of selecting their KNN, and of computing their recommendations.

An example of a different extent of hybridization is [4], in which Ali and van Stam propose an item-based CF system which distributes the storage of users profiles, and the computation of recommendations, but keeps centralized the computation of item similarities.

Pistis [72] also uses a hybrid architecture, though the purpose of this choice has more to do with user privacy-preservation than scalability. Basically, in Pistis, there is a central server which only coordinates computations and host public contents. Users are responsible for most computations, including selecting recommendations, and storage of their profile, much like decentralized systems. This way, neither the server or other users are never in possession of a user's private ratings, even under encrypted form.

In conclusion, RSs using a hybrid architecture can lead to improvements of users' privacy compared to centralized systems, if they are designed towards this purpose. That being said, even if hybrid architectures reduce the likeliness and impact of threats from "Big Brother" adversaries, they do not completely prevent them. Moreover, the central server still represents a single point of failure, so hybrid systems are not more resilient to censorship than centralized ones. While censorship is not directly a threat to privacy, one can imagine that a powerful adversary could shutdown a hybrid system by taking down its central server, to force its users to migrate to other less privacy-friendly RSs.

2.1.3.3 Distributed Systems

In distributed RSs, a.k.a. decentralized ones, there is more than one operator, thus control of the system is distributed among them. Not all systems are distributed in the same way. Some of them retain an architecture with many clients receiving recommendations on one side, and several servers producing recommendations on the other side. We call federated this type of systems in the remainder of this document. Alternatively, some systems are fully distributed (a.k.a. peer-to-peer), meaning that each peer is as much a client, *i.e.* it receives recommendations, as it is a server, *i.e.* it helps other peers to compute their recommendations.

Regarding privacy of users, distributed systems offer inherent advantages such as making irrelevant "Big Brother" adversaries, or leaving users in control of their profile in peer-to-peer (P2P) systems. However, this lack of central authority also introduces new challenges, notably in terms of trustworthiness for private data management of the participants, that is servers in federated systems, or peers in P2P ones. Usually, one wants a distributed RS to be open so as to have as many participants as possible, because its scalability grows with its size. Yet, this openness implies that anyone can participate, including dishonest or ill-intended people. Therefore, the main issue is that participants have to cooperate for the system to be operational, but, in doing so, they also need to be wary of other participants in order to protect users' privacy.

We now review some federated CF systems, before covering more in depth fully distributed (a.k.a. P2P) ones, as it is the context of Chapter 3.

Federated systems A federated distributed CF system is akin to a centralized one where the central server has been split into several servers performing the same function. Two common policies regarding who can operate a recommendation server within the federation are : open to anyone, or restricted to some types of entities, *e.g.* professionals within one economic sector. Clients can be users or content providers or both, similarly to centralized systems.

Federated systems facilitate privacy-preservation of user data, but does not automatically yield strong privacy. Although this architecture automatically protects users from "Big Brother" adversaries, simply switching from a centralized non-private system to such architecture is not enough. Doing so makes it hard for one operator to access all users' profiles, but it also increases the probability that at least one operator is malicious, considering that all operators have the same small probability of being malicious.

For instance, GroupLens [92] is a federated CF system which is not designed for privacy, thus user ratings are actually less protected than in a centralized system. It provides recommendations of Usenet messages by adding recommendation servers to Usenet's distributed architecture. These servers store their local users' ratings, and predict missing ratings for the same users. Predictions are done naively using the basic predictor from Section 2.1.2.4, so when a server computes a prediction, it needs to have locally all the ratings for the considered items. In order to enable every server to make predictions for any item, servers propagate their local ratings to all the others, piggybacking over the standard Usenet message propagation protocol. The facts that anyone can run a recommendation server, and that rating are propagated to all such servers, make GroupLens actually reveal all user's preferences to anyone.

In opposition, when one uses a federated architecture with privacy in mind, this helps ensuring users' privacy. It is the case in [53], which proposes a federated system to recommend the best physicians for treating some illnesses. Given the sensitivity of the data manipulated by the system, it is designed so that (1) servers cannot access the individual ratings they store, (2) several servers need to cooperate to generate cleartext aggregate statistics, and (3) the illnesses for which a user requests recommendations remain unknown to the servers. These properties are achieved through the combination of users' ratings and requests being sent encrypted to one server, and servers performing computations using secure multi-party computation (SMC). Moreover, the authors advise that servers are run by distrustful entities, such as competing insurance companies or hospitals, to ensure that SMC is not diverted to reveal private data.

Fully distributed systems Fully distributed or peer-to-peer (P2P) architectures are the most popular implementations of distributed RSs. In such systems, each user is represented by a peer storing her profile, and running a protocol to compute her recommendations. There is no server or super-peer coordinating the system, so each peer only has a limited knowledge of the system, depending on which peers it has connected with. The fact that peers only have local knowledge makes user-based CF techniques the best fitted ones because they only require to know a user's KNN to compute her recommendations.

In terms of privacy, P2P systems make "Big Brother" adversaries completely ir-

relevant, and have users keep their profile under their control. This shifts privacy concerns towards performing collaborative filtering while releasing as little private information as possible. Indeed, CF requires one to share some preferences to get recommendations, but the P2P architecture exacerbates this by leading one to computer her similarity with many others, including potentially malicious users, to find her KNN. Therefore, fully distributed systems raise new threats such as what we call "Little Brothers" adversaries, *i.e.* malicious users using it to hoard private information. Additionally, P2P systems have good properties such as resilience to censorship, scalability.

We now review different P2P CF systems depending on the way they connect peers together.

Gossip-based overlays The main approach on which P2P RSs rely is gossip-based overlays. In a gossip (a.k.a. epidemic) protocol, at each asynchronous round, each peer randomly select one or more peers it is connected to, and exchange information. The set of all peers running the same gossip protocol form an overlay network. The set of peers to which a peer is directly connected is called its view of the overlay. Despite their simplicity, gossip-based overlays have good properties such as accommodating unpredictable message delays, peer failures [58, 59].

In CF systems, gossip-based overlays allow users to quickly find their KNN, and to always have neighbors despite peers leaving and joining (peer churn). If a gossip protocol is used to disseminate items within the system, the epidemic aspect also ensures that they reach users interested in them.

We discuss some examples of gossip-based CF systems in the following.

WhatsUp [16] applies a P2P user-based CF system to news recommendation. It relies on two gossip protocols: WUP to generate recommendations, and BEEP to disseminate news items within the system.

WUP enables each user to find her KNN, the main building block for user-based CF. It consists in two gossip-based overlays in which peers exchange lists of peers they are connected to within the same overlay. The random-peer sampling (RPS) overlay is a continuously changing graph ensuring that the whole network stays connected. In this RPS protocol, a peer periodically exchange half of its view, with the oldest peer in its view, *i.e.* the peer which was contacted last. The clustering overlay connects peers based on their similarity, thus a peer's clustering view constitutes its KNN after some time. In this clustering protocol, a peer periodically exchange its whole view with the oldest peer in its view, then computes its similarity with peers in its current view and the received view, before updating its view by keeping the k most similar peers.

BEEP disseminates items in an heterogeneous fashion, depending on the user's opinion on the item to be disseminated, by orientation and amplification mechanisms. Whenever peer p receives a new item i, it asks its user A to rate it (like or dislike), then forwards it to other peers (push strategy). Orientation affects which peers p forwards i to. If A likes i, p forwards it to random peers in its clustering view. Else, if i has not been disliked too many times, p forwards i to the peers in its RPS view which are the most similar to i's profile, which is an aggregate profile of ratings from the profiles of previous users who liked i on its dissemination path.

Amplification affects the number of peers to which p forwards i. If A likes i, p forwards it to half of its clustering view, else p forwards i to one peer in its RPS view.

WhatsUp outperforms a gossip-based P2P CF without orientation and amplification, in terms of $F1$-*measure*, while being only 5% behind a centralized equivalent of WhatsUp, using the same measure. WhatsUp bandwidth consumption is dominated by the number of news items disseminated, and the bandwidth fraction dedicated to overlay management is between 0.4 and 4 kbps depending on the gossip frequency. However, there is no privacy-preserving mechanism, which leads WhatsUp to systematically revealing users' profile.

Tribler [88] proposes a P2P file-sharing system integrating the Buddycast gossip protocol in order to perform user-based CF. Similarly to WhatsUp's WUP protocol, Tribler relies on one similarity-based overlay, and one overlay of random topology. At each round, a peer contacts one peer from either of its views, and they exchange a Buddycast message consisting in 20 peers, 10 from each overlay, from their respective views, as well as a subset of these peers' profile. The choice of which overlay to contact a peer in is controlled by a fixed ratio. Regarding recommendation itself, the authors use a standard user-based CF technique from [19], but they do not evaluate the recommendation performance of their CF module as it is not the main focus of the paper. Also similar to WhatsUp, Tribler's recommendation module does not include any other form of privacy-preservation than the possibility of disabling recommendation altogether.

[6] proposes a P2P user-based CF system combining implicit and explicit membership to interest communities. A user joins implicit interest communities defined by her KNN, which is computed via the classic combination of a RPS gossip-based overlay and a similarity-based one. Additionally, this system creates one gossip-based overlay per public interest community, which a user can then explicitly join depending on the interests relevant to her. The interest category of such a community is explicitly defined by the profile of one of its members, elected as representative of that interest category by the community members. The election protocol is a gossip one too.

Regarding the recommendation process, peers can either use a push or a pull strategy. In the pull one, a peer requests recommendations to one or more of its KNN, then each contacted neighbor answer with all the items from her profile, which are more similar to the requesting peer than a threshold, using an undisclosed metric. In the push one, whenever a peer receives a new item, it determines the interest category it belongs to, then send it to the peers of the corresponding community, who are more similar than a threshold to the new item. This last strategy leverages public interest communities to automatically push recommendations efficiently because recipients are likely to enjoy it, given their explicit membership to the community. Moreover, assuming a way to automatically determine an item's category, this strategy allows faster dissemination than WhatsUp because a peer does not require its user's opinion to push an item.

This paper is even less respectful of user privacy than the aforementioned ones because, in addition of automatically revealing user profiles to neighbors, it may elect a user as representative of a public community without her explicit consent,

thus revealing her profile to any user in the system.

Other approaches Although gossip protocols are a popular approach to P2P RSs, some systems use alternative approaches such as a Distributed Hash Table (DHT), an ad hoc protocol, or a Publish-Subscribe (PubSub) system.

[51] proposes PipeCF, a P2P user-based CF system based on a DHT. In PipeCF, storage of ratings is distributed across all peers regardless of who issued each rating. Indeed, rating storage is managed by a DHT where a key is a <item ID-rating value> tuple, and a value is a list of users whose profile contain the corresponding tuple. Therefore, when a user issues a rating, she sends a copy in the DHT to the user responsible for the corresponding key.

In PipeCF, a user generates her recommendations using the basic predictor of user-based CF (see 2.1.2.4), and selecting her neighbors from user lists returned by querying the DHT for each <item ID-rating value> tuple from her profile. Han *et al.* ensure this process is performant by significance refinement for scalability, and unanimous amplification for recommendation quality. Significance refinement consists in limiting the number of users returned by a DHT lookup, while unanimous amplification, inspired from [19], increases the contribution to predicted ratings of users sharing at least some common ratings with the current user.

PipeCF does not consider privacy matters at all, and is even worse in this regard than centralized systems because it sends a copy of each rating to whoever is responsible for some partition of the DHT's keyspace. A malicious user only have to participate in the DHT, and wait to receive user ratings to learn private data. Moreover, a similar but active adversary could perform a denial of service on any user responsible for a specific item, until she become responsible for it, thus allowing her to discover every user rating this item.

PEOR [64] is a P2P user-based CF system applied to picture recommendation. Each peer maintains connections to its KNN peers, and uses an ad hoc strategy to manage its KNN. When a new peer joins, its KNN is initialized with popular peers (found by an undisclosed mechanism), then the peer updates its KNN by keeping the most similar peers among its current neighbors and his neighbors' neighbors. A peer updates its KNN whenever its user updates her preferences.

Peers disseminate items automatically by pushing those liked by their user, to their neighbors. Peers store pushed items, up to a limit amount, then recommend to their user the highest scoring items, using the basic predictor (see 2.1.2.4).

PEOR does not include any privacy-preserving mechanism, and user profiles are freely available to neighbor peers.

Alternatively, [56] proposes a P2P PubSub system using matrix factorization (MF) to predict ratings for all the items received by subscribers. In this system, users may be content providers, or consumers, or both. Consumers explicitly specify (subscribe) from which providers they wish to receive content, while providers send (publish) their content to all or some of their subscribers as soon as they produce it. The authors propose a fully distributed gradient descent algorithm to solve a MF formulation which predicts distribution of ratings rather than ratings directly. A rating is then predicted by combining each possible rating value weighted by its probability.

The authors' choice of a decentralized architecture being motivated by privacy concerns, this system is more careful than the aforementioned ones in preserving its users' profile. Indeed, the gradient descent algorithm is specifically designed so that two users disclose their profiles to each other only if they have a producer-consumer relationship. Although this offers better privacy-preservation than most P2P RSs, it relies on the assumption that a consumer systematically trust the producers it subscribes to, and conversely. This is strong assumption, especially regarding producers who might want to publish their content, to a possibly large number of subscribers, without revealing their profile.

In summary, we saw that recommendation systems (RSs) can be classified according to their recommendation technique and their architecture. The most important type of technique by far being collaborative filtering (CF), we explained in more details the main CF techniques, and notably user-based ones which are simple yet provide good recommendations. Regarding the architectures of RSs, centralized ones are the dominant ones in the industry, yet there is a large body of academic literature on distributed systems, especially fully distributed (a.k.a. P2P) ones.

When considering user privacy, centralized systems pose a fundamental threat caused by "Big Brother" adversaries. Therefore, P2P systems are one way to alleviate this threat. The most natural approach to provide privacy-preserving recommendation in a P2P way consists in implementing a user-based CF technique with gossip protocols. However, this approach raises other threats to users' privacy, thus requiring careful design and dedicated privacy-preserving mechanisms.

2.2 Privacy Attacks on Recommendation Systems

The majority of attacks against recommendation systems studied in the literature focus on the robustness of the recommendation process rather than than on privacy. Profile injection (a.k.a. Shilling) attacks [82, 49] target collaborative-filtering systems, and consist in creating fake user profile which are biased regarding some specific items. When the goal of this type of attack is to increase the likelihood of an item to be recommended to genuine users, it is called a push attack. Conversely, a nuke attack aims to decrease the probability of an item to be recommended. These attacks are quite effective in memory-based systems, especially user-based ones, while model-based ones are more robust [77], especially PLSA because it finds some "authoritative" users to who it gives more importance when computing recommendations. However, Power User attacks [103] are able to manipulate even model-based CF systems by creating influential users, *i.e.* users who can affect the recommendations for the largest group of users, according to measures of influence such as user indegree. The same authors also propose complementary Power Item attacks [102] which are effective against item-based systems, while Power User attacks are not.

That being said, there is some literature about attacking recommendation systems with the goal of breaching the privacy of various actors, and mainly users'

privacy. We focus on attacks on CF systems which target user privacy, among which we can distinguish between passive and active attacks. In the former, the adversary operates normally according to her role in the system, be it a user, an operator, a content provider or a third-party, and simply tries to learn information about users through legitimate means. In the latter, the adversary carries out operations that go outside the standard behavior of her role for example by introducing fake identities, or fake items. We discuss passive attacks below, and active ones later, going each time from the strongest to the weakest requirements on the considered adversary.

2.2.1 Passive Attacks

We review four passive attacks on user privacy. The first two require the adversary to have the role of RS operator, while the third attack requires her to be an operator or a third-party (*i.e.* no role in the RS). Finally, the last attack has the lightest requirement since the adversary needs only be a third-party.

BlurMe [117] and [11] by the same authors, present passive attacks that extract demographic information such as the ethnicity or gender of users from the ratings of items in a matrix factorization-based collaborative-filtering system. In both papers, the adversary takes the role of an operator of the RS.

In BlurMe, although the attack aims to discover a user's gender from her ratings, it works similarly for other categorical demographic information. This gender inference uses a classifier which the authors train using two datasets of traces from real RSs (MovieLens and Flixster) containing demographic information for at least some users. They study classifiers based on Bayesian probability, support vector machines (SVM), and logistic regression, of which the last two perform best. The authors evaluate their attack using AUC (Area Under the Curve, Receiver Operating Characteristic (ROC) curve implied), *Precision*, as well as *Recall*, and find the gender inference's *Precision* and *Recall* to be between 70% and 80%. Moreover, they note that the information that a user rated an item is sufficient for the attack to work. Granting the additional knowledge of rating values to the adversary only increases the attack's performance by 2% or less for any of the above measure.

In [11], the adversary takes advantage from the fact that CF systems usually circumvent their cold-start problem by asking news users to rate some items, to select the items to be rated so that they maximize her confidence in the inferred users' private information she targets. Unlike the previous attack, this one is designed to infer binary-only private information, such as gender or whether a user is an adult, which the authors call types. Another difference with the authors' previous work is that they consider a system using Bayesian matrix factorization, a Probabilistic Latent Semantic Analysis (PLSA) technique, instead of a "regular" LSA one *i.e.* based on linear algebra.

In this attack, the adversary first computes off-line latent factor profiles for users and items, as a normal operator would do. This step also yields a type-dependent bias for each item because ratings as modelled in the paper include a bias which depends on the type of the user who issued the rating. Then, when the adversary asks a new user to rate a given number of items, she simply select the items with the largest difference between the two type-dependent bias, *e.g.* the items for which

the difference of rating bias between men and women is the largest. Once that new user rated said items, the adversary uses the proposed Factor-Based Classifier, which leverages the same latent factors used by the recommendation algorithm, to perform her type inference. Indeed, the adversary infers a private information, a.k.a. type, $\widehat{t}$ by computing the conditional probability of the two possible values given a set of ratings r:

$$\widehat{t}(r) = \arg\max_{t \in \{-1,+1\}} P(t|r)$$

Yet, one can compute $\widehat{t}(r)$ as soon as one knows for each item in the dataset, its latent factor profile, and its type-dependent rating bias. The authors find that the efficiency of their passive attack is high, with AUC values from 0.70 to 0.80 when asking users to rate 10 items. Furthermore, $RMSE$ is almost identical to that of non-malicious strategies against cold-start, such as selecting items with the most polarized ratings, except in one out of six combinations of dataset and type.

The third attack [24] shows that targeted and personalized advertisements (ads) can be harmful to users' privacy and contain valuable information that allows accurate reconstruction of users' interest profiles. Although the context of this paper is personalized online advertising, the attack it presents can be adapted to other personalized services including recommendation. This is a profile reconstruction attack, that is an attack where the adversary's goal is to discover some private information allowing her to reconstruct the private profile of the target user. The attack is passive because the adversary only needs to observe the ads served to the target user, meaning she can take the role of an operator (of an advertising network here) or the role of a third-party within the same local network as the target because ads are more often than not served over unencrypted channels. Basically, the attack works in two stages: (1) the adversary collects the ads served to the target, and keep only the personalized ones ; (2) the adversary determines the interest categories each ad pertains to, and form a reconstructed profile with all these categories. During the first stage, the adversary collects the URL of each advertisement as well as the tracking cookie used by the ad network to identify the different visitors. Tracking cookies are sent during the second stage to the ad network's preferences page in order to obtain the interest categories associated with each ad. The authors evaluate the quality of the reconstructed profiles with *Precision* and *Recall*, reaching up to 79% for the former, and 58% for the latter.

Finally, [21] analyzes instead how auxiliary information, mostly obtained from external sources, makes it possible to extract individual user preferences from temporal changes in otherwise aggregate information such as related-items lists or item-covariance matrices. The adversary can take the role of a third-party as she does not need to interact with the CF system because she only watches some related-items lists, which are made public by the system. Indeed, this profile reconstruction attack affects the subset of item-based collaborative-filtering systems which make freely available for each item either a list of the most correlated other items, or the underlying matrix of covariance between all items. Some systems release this kind of information because, on the face of it, it does not seem susceptible to reveal private information of individuals.

The attack's rationale is that if one knows a combination of items which is distinctive enough of a target user (*i.e.* a quasi-identifier), observing the appearance a new

item in the related-items lists of several of the quasi-identifying items, one can infer that the target user added the new item in her profile with reasonable confidence. These quasi-identifying items present in the target user profile are what the authors call auxiliary information. The adversary may obtain auxiliary information about a target from the recommendation system itself, such as Amazon's "verified purchase" feature, or from third-party services, such as online social networks or general product reviewing websites. The authors evaluate their attack on three real-world CF systems, measuring the attack's success rate with $Yield$, the number of inferences, and *Accuracy*, the percentage of correct inferences. Because there is an inherent trade-off between $Yield$ and *Accuracy*, they achieve results from 58 inferences per user with 50% *Accuracy*, to 6 inferences per user with 90% *Accuracy*.

2.2.2 Active Attacks

We now look at active attacks on user privacy, that is attacks where the adversary performs operations that go outside the standard behavior of her role in the recommendation system. We start with an attack where the adversary must be a RSs operator, then in the two following attacks, she is only required to be a user of the system.

[11] also develops an active variant of the previous gender-inference attack that seeks to reach a high inference confidence as quickly as possible by selecting the next item to be rated after each rating provided the new user. This means that in this attack, the adversary asks a new user to rate one selected item at a time, showing the next one only after the user provided a rating. Specifically, once the user provided a new rating, the adversary updates its knowledge accordingly, and selects the item to be rated which minimizes the classifier's expected probability of incorrect inference. The authors evaluation shows that this active attack performs similarly or better than its passive variant, in terms of AUC and $RMSE$ alike. It also outperforms the passive attack from BlurMe [117] with a greater AUC by 10% to 30%. Finally, with a time to select the next item from 0.15 sec. to 0.4 sec., this active attack runs fast enough for new users to be oblivious to the adversary's underhandedness. This attack is relevant as long as one considers centralized collaborative-filtering systems or partially distributed ones as there must be a single operator in charge of choosing which items to rate for users.

Calandrino *et al.* also introduce [21] an active Sybil attack which we evaluate in Chapter 4. A Sybil attack consists in the adversary assuming control over several fake identities, each appearing as a different user to the attacked system. In this case, the attack affects user-based CF systems, and the adversary's goal is to discover new items from the profile of a target user. The adversary does so by tricking the CF algorithm into revealing by its recommendations items from the target user's profile in certain conditions. In order to meet these conditions, the authors assume that the adversary knows the algorithm's parameters, especially k (as in KNN), and that, similarly to the authors' passive attack, she has some auxiliary information about the target user. The adversary begins by creating k Sybil users using the auxiliary information as their profile. Then with high probability, the system should attribute to each Sybil a neighborhood composed of the $k-1$ other Sybils, since they all have

identical profiles, and the target user, since the Sybils' profile is a subset of hers. This composition of Sybils' neighborhoods is the success condition of the attack. When this condition is met, any item recommended to a Sybil user comes from the profile of the target user, because user-based CF does not recommend items already rated by the recipient, and because these algorithms draw the candidate items from the profile of the recipient's neighbors. Because such an attack exploits the rationale of CF itself, it applies to centralized and decentralized systems alike.

This active Sybil attack is not evaluated in [21] because the main focus of the paper is on the previously described passive attack. The results of our evaluation in Chapter 4 partially confirm the intuition of Calandrino *et al.* that the attack should be effective, but they also show that the attack's performance depends both on the similarity metric and on the user population of the considered RS.

Pistis [72] considers an active attack somewhat similar to the one described in [21]. Indeed, the attack from [72] tries to trick the RS into revealing private information of the target user, but it needs only one fake identity. This paper considers online social communities featuring a centralized hybrid CF system akin to Google News' [31], where users can have private interactions with items (access an item's content) as well as public interactions (posting an item or commenting one). Therefore, in this type of system, ratings are implicit and user profiles have a public part, containing items with which the user had public interactions, as well as a private part, containing items with which the user had private interactions and which do not belong to any interest group in common with her public items. Each item belongs to one or more interest group which are themselves determined by a k-centroids clustering algorithm [70].

Regarding the attack itself, the adversary assumes the role of a regular user. Instead of information about the target obtained through auxiliary channels like Calandrino *et al.*'s Sybil attack, the adversary fills the profile of her fake user with information obtained directly from the attacked system, which is the whole public profile of the target user in this case. Then, she requests recommendations and adds to her user's profile the top-n recommendations, the value of n or a method to choose it being unspecified by the authors. She repeats this step an undisclosed number of times before making a guess that some of the items learnt by requesting recommendations are part of the target user's private profile. The attack achieves a precision in $[0.6, 0.75]$ combined to a recall in $[0.1, 0.4]$ when the adversary makes few guesses, and a precision in $[0.2, 0.45]$ combined to a recall of 1 when she makes more guesses. Although this attack is applied to a centralized RS in this paper, it is also relevant in a decentralized system as long as the adversary is able to find the public profile of her target.

Despite the aforementioned attacks from the academic literature, it is currently unknown whether any commercial recommendation system suffered large-scale attacks [35]. The few public reports of attacks on a recommendation system only mention targeted attacks on a specific item [106]. However, this may be due to passive targeted attacks being difficult to detect, and/or to big players (Amazon, Netflix, ...) protecting their reputation by keeping reports private. Even if no large-scale attack was ever performed, privacy issues of recommendation systems should be addressed before they degenerate into a serious scandal.

In summary, we saw that academic literature studies more closely the attacks aiming to bias RSs than those aiming at breaching users' privacy. That being said, the latter is not completely under-explored. There is a number of attacks, either passive or active, which target CF system given that their very nature make them susceptible to reveal private information. One of these attacks is especially worth noting: the active Sybil attack suggested by Calandrino *et al.* because it can exploit any user-based CF system, regardless of its architecture, as long as the adversary is able to obtain a small amount of knowledge about her target's preferences.

2.3 Privacy-preserving Recommendation Systems

There are several approaches to privacy-preservation in RSs and we look at those based on encryption, data obfuscation, and differential privacy.

2.3.1 Privacy by Encryption

We first cover encryption techniques as protections of users' privacy. Indeed, encryption provides confidentiality because encrypted data is indistinguishable from random data for anyone else than the owners of the encryption keys. This property holds under some computational assumptions which make brute force guessing of keys impractical with any regular (*i.e.* non-quantum) computer. Most privacy-preserving RSs based on encryption rely on asymmetric cryptography, and more specifically on homomorphic encryption which allows one to perform some operation on ciphertexts and get the result when decrypting the ciphertexts. Partially homomorphic cryptosystems support only some operations, *e.g.* additions, multiplications, while fully homomorphic ones support any operations. Although encryption provides strong privacy and does not hamper recommendation quality, it is computationally expensive, thus making it somewhat impractical for large scale RSs.

In this section, we discuss homomorphic encryption, and secure multi-party computation for recommendation.

In [23], Canny proposes a peer-to-peer model-based collaborative-filtering system which preserves the privacy of users' profile by using homomorphic encryption to compute an aggregate model of users' profile. Once this aggregate model is computed, it is made public so that any user can locally compute recommendations for herself based on it. Canny uses a partial singular value decomposition to create the aggregate model because it can be computed using an iterative conjugate gradient, which consists only of additions. This allows his system to use the ElGamal cryptosystem, an additive homomorphic one, which has computational requirements low enough to make the system practical.

The aggregate model is computed in a P2P fashion, and is guaranteed to be correct if less than 50% of peers are malicious. This is made possible by having peers choose one or both the roles of users or talliers. The former ones are the

only ones to have shares of the decryption key, while the latter ones compute the encrypted aggregate model.

[109] proposes a classic centralized item-based CF system with the addition of an additive homomorphic cryptosystem (Paillier's in this case). The authors claim that users' privacy is preserved, and that their system minimizes the computational overhead incurred by the cryptosystem because it is only used during off-line precomputations. The recommendation process consists in two steps: the precomputation one, then the recommendation generation one. During the precomputation step, all users encrypt their profiles using a common public key and make the ciphertexts available publicly. Then, one user computes all similarities between items as well as each item's mean rating, thanks to the homomorphic property of the encryption used. Finally, this user hands the resulting ciphertexts to a set of trusted third-parties who share parts of the private key corresponding to the public one which was used for encryption. If enough third-parties collaborate, they are able to decrypt and publish publicly the resulting item similarities and item mean ratings. This step is only run periodically and off-line. The second step simply consist in each user computing her own recommendations using a classic item-based predictor (see Section 2.1.2.3), thanks to the data published by the third-parties. This step is performed online, and efficiently as it does not require encryption or decryption.

The authors say that users' privacy is ensured because profiles are kept encrypted, and the data published regarding items is an aggregate, so it does not reveal private information. However, they do not describe any mechanism for the third-parties to check that the ciphertext they are asked to decrypt is what they expect. If one considers a malicious user, she could instead send for decryption a piece of another user's profile. Moreover, the assertion that making public item similarities and item mean ratings does not affect users' privacy can be doubted given the passive attack described in [21].

We now look at systems employing secure multi-party computation (SMC) to ensure users' privacy.

[53] proposes a distributed system in which users submit their encrypted ratings (for physicians and illnesses) to several servers computing the recommendations by SMC. Whether SMC is implemented through homomorphic or threshold cryptosystems, it requires that a large enough subset of all the servers collaborate to perform the operation. This assumes that confidentiality of the operation is preserved if less than the threshold number of servers can collude, which is realistic given the application domain. For instance, the different servers could be run by competing entities such as competing physicians or hospitals. Besides collusion, the authors consider a honest-but-curious (a.k.a. semi-honest) adversary operating a server.

The recommendation process is slightly different from traditional CF systems. Indeed, users do not have a profile of historical ratings as they just sent their ratings (a rating concerning both a physician and an illness) to a set of servers. These servers maintain a matrix of encrypted scores for each combination of physician and illness, which they update with user-sent ratings. Periodically, they perform SMC to produce ranked physician lists for each illness, which they jointly decrypt and make publicly available. This does not yield directly personalized recommendations on purpose in order to protect users' privacy, while still allowing them to locally per-

sonalize the physician lists by assigning personal weights to some illnesses. Moreover, the system's robustness is enforced by requiring zero-knowledge proofs from users submitting ratings so as to avoid manipulation of scores by malicious users.

Another approach to privacy-preservation using SMC is Pistis [72] It proposes a collaborative-filtering system with a hybrid architecture and privacy-preserving properties, including resilience to a kind of active attack we discuss in Section 2.2. The type of CF system considered in this paper allows user to make public or private some of their ratings (like/dislike). The goal of Pistis is to prevent adversaries from discovering the private part of users' profile. It achieves this goal thanks to (1) the introduction of interest groups as well as (2) the combination of a hybrid architecture and a secure multi-party computation (SMC) algorithm.

Interest groups prevent correlation between private and public items of a user A by forbidding that a private item of A belongs to one or more interest group also containing a public item of A. Pistis defines a fixed number of interest groups, determined by a k-centroids clustering algorithm [70].

In Pistis' architecture, the main purpose of the central server is coordination: it initiates the k-centroids clustering, maintains the resulting interest groups, and coordinates the weighting of items within a group when asked so by a user so that she may compute recommendations for herself. Privacy of users' private profile is ensured because profile storage and recommendation computation are done on users' machine, and because the algorithms which create interest groups, determine the groups an item belongs to, and which compute an item's intra-group weight all rely on the *SecureSum* primitive. *SecureSum* is a SMC protocol which compute the sum of n parties' value, while keeping each value private under a honest-but-curious (a.k.a. semi-honest) adversary model. Therefore, *SecureSum* enables to compute the *Jaccard* index in a private and distributed fashion, which in turn enable users to run the aforementioned distributed algorithms and preserve the privacy of their private profile.

The authors evaluate Pistis with a three-weeks long deployment on the Fudan BBSi[1]. They compare it to a centralized model-based (MinHash) CF system, a decentralized privacy-preserving model-based (SVD) CF one, and a baseline memory-based CF one. The recommendation quality, measured with *Precision* and *Recall*, of Pistis is superior by 47% to 164% on average, depending on the competitor. They evaluate privacy by observing the success of an attack (see Section 2.2), measured with the adversary's precision and recall when making guesses. Pistis reduces the attack' success by 139% to 157% on average.

2.3.2 Privacy by Data Obfuscation

Obfuscation consists in altering data so that it is difficult for an adversary to reconstruct the original data from the obfuscated ones. In consequence, proper obfuscation of users' data yields privacy because, even if the adversary gets access to users'data, she cannot use it to reliably learn private information about them. However, obfuscated data should still retain some utility in order to allow legitimate

[1] `http://bbs.fudan.edu.cn`

uses such as recommendation. So there is a trade-off between privacy and recommendation quality when using obfuscation. In this section, we discuss three different approaches to data obfuscation for RSs.

The first approach consists in coarsening ratings, and mixing different users' ratings. [38] proposes a centralized model-based collaborative-filtering system with a two-stage obfuscation algorithm which preserves the distribution of ratings. This algorithm preserves users' privacy by hiding users' true ratings from an honest-but-curious (a.k.a. semi-honest) adversary acting as the central server's operator, who may collude with some users to try and discover the target user's true ratings. In the considered system, when a user asks for recommendations, she sends the obfuscated version of the true profile resulting from the two-stage algorithm to the central server.

We detail below how the proposed obfuscation algorithm works with the example of user A asking for recommendations. The first stage starts with A locally applying the Blind Tree Algorithm (BTA) which partitions her ratings into several clusters (the tree's leaves) using the Local Learning Analysis (LLA) clustering algorithm, and replaces each rating value within a same cluster by the average value of ratings in that cluster. A finishes this stage by asking whether other users want her to forward to the central server their requests for recommendations at the same time as hers. This mutualisation of recommendation requests increases users' privacy because, in the event of the central server operator breaching the BTA obfuscation, she does not know to whom the profiles belong.

Additionally, A may further apply a second stage of obfuscation depending on the number of other users replying to her. If less than a threshold number of users answer positively by sending A their BTA-obfuscated profiles, A sends as-is the collected profiles with hers to the central server. Else, she performs the second stage that is applying the Enhanced Perturbation (EP) algorithm to her profile and the collected ones, because otherwise the adversary might be able to infer information if she receives enough independently BTA-obfuscated profiles. This algorithm starts by clustering the different ratings using the LLA algorithm. Then, the EP algorithm generates two different matrices using principal component analysis, one using only the highest scoring points in every cluster according to a field function, the other using the covariance matrices of each cluster. Finally, the EP algorithm finishes by projecting the former on the latter.

Unfortunately, the authors do not provide clear intuition regarding what the purpose of each stage of the obfuscation algorithm is. They evaluate the recommendation quality of their system with MAE, and its privacy using an unspecified measure, but they do not provide comparisons with other systems.

Another approach to data obfuscation is add, remove, or reorder some ratings or items. Usually, the choice of which transformation to perform, and on which ratings/items depends on the particular type of attack on user privacy considered. In the case of [26], the authors consider the passive attack based on temporal changes in public related-items lists (RILs) from [21]. The main privacy breach exploited in this attack comes from a common item appearing or being ranked higher within several RILs that the adversary knows to be associated with the target user. Therefore, when a user action, *e.g.* a liking item i, would lead to such changes in some RILs, the authors of [26] propose to either remove i or rerank i lower within the RILs.

More precisely, they determine which pairs of item and RIL require this obfuscation using the *δ-bound*. This bound is respected when letting item i enter or rank higher in a RIL within some time window, does not causes the adversary's confidence in guessing that item i is in the target's profile, to surpass the threshold value δ, for any amount of auxiliary information.

Alternatively, BlurMe [117] proposes an obfuscation technique tailored to passive attacks where the adversary operates the CF system to discover users' demographic information. This technique adds a number of ratings in a user's profile before sending it to the centralized system. These additional ratings are for items chosen to mislead the adversary's inference algorithm, applied to gender in this paper. Thus, this technique obfuscates a female user's profile by adding ratings for some items found in the list of items which are the most correlated with male users. The values of these additional ratings can be either the average rating of these items, or the predicted rating by the current user for items.

One last approach to data obfuscation is Random Perturbation Techniques (RPT). Such techniques protect users' privacy by perturbing the content their profile before using it in the RS. Perturbing A's profile consists in adding a random value (a.k.a. random noise) taken from a specific distribution to some or all of the original rating values, including items which A did not originally rate. It is possible to draw random values from different distributions as long as they preserve to some extent the distribution of aggregate data, so that predicted recommendations are still useful.

For instance, [91] proposes to apply RPT to a hybrid memory-based/model-based centralized collaborative-filtering system.This paper suggests that users only send an obfuscated version of their profile to the centralized system, in order to prevent its operator (the adversary) from learning users' original ratings. The obfuscation process is as follows: user A picks a distribution and its parameters, she selects for which items i to modify their rating $r_{A,i}$, then A adds random values drawn from the chosen distribution to each $r_{A,i}$ previously selected. This process is common for all users, although each one is free to choose her distribution independently from the choices of the other users. Users may choose either a uniform distribution in range $[-\alpha, \alpha]$, where $\alpha = \sqrt{3} \times \sigma$, or a Gaussian distribution with mean $\mu = 0$. In both cases, users individually set the desired privacy level by choosing the distribution's standard deviation σ (higher σ values meaning higher privacy). The complementary parameter for each user to define her privacy level is the choice of which ratings to obfuscate. She may obfuscate any item's rating, even for items not found in her original profile, thus hiding which items she actually rated. This RPT still allows to compute reasonably accurate recommendations thanks to the distributions used having a mean of 0. Therefore, when aggregating many ratings, the expected value of obfuscated aggregate ratings converges towards the expected value of original aggregate ratings.

[86] and [87] use RPT in a similar way to preserve users' privacy in a centralized CF system. In the former, users apply random noise drawn from a uniform distribution like described above, to all their ratings, but each user can choose the range of the distribution. In the latter, the central server decides the distribution (uniform or Gaussian) and its parameters, and users turn their ratings into z-scores before applying random noise. Additionally, users also fill their missing ratings with

their average rating, before applying the same process as they apply to their actual ratings.

However, some papers [55, 62] point out that Random Perturbation Techniques as used previously are not always fit for user privacy-preservation. [55] proposes two techniques which exploit correlations between items to reconstruct original data from obfuscated data. One technique uses principal component analysis (PCA), which computes a list of eigenvectors summarizing more compactly the full obfuscated rating matrix, to filter random noise from original ratings by projecting the obfuscated matrix using the first eigenvectors. Given that (1) PCA-produced eigenvectors are ranked by decreasing variance of the ratings they represent, and that (2) random noise values are independent from ratings *i.e.* their variance is spread among all ratings which entails that they are represented by lower-ranked eigenvectors, then the aforementioned projection should retain most of the information from the original ratings while removing a fair amount of random noise. The other technique uses Bayes estimate which consists in guessing that the rating r maximizing the probability $P(r|r')$ given the observed obfuscated data r' is close to the original data. This Bayes estimate yields a formula to reconstruct the original ratings based on the expected value of the obfuscated ratings, and on the covariance matrix of the original ratings, which can be estimated from the obfuscated data by using the reasonable assumption that original ratings follow a multivariate Gaussian distribution. The two previous techniques differ in that the PCA-based technique is more easily understandable while the one based on Bayes estimate performs better when data are less strongly correlated. Given the effectiveness of their proposed reconstruction techniques, the authors of [55] also propose a RPT which generates random noises using the same covariance matrix as the one describing the original ratings. This yields correlated noise values with a distribution similar to that of the original ratings, which in turn makes PCA represent most of the amount of noise with the first few eigenvectors. Consequently, discarding the lower-ranked eigenvectors does not remove much noise anymore.

2.3.3 Differential Privacy-based Techniques

We now discuss a more formal way to strengthen the guarantees offered by random obfuscation techniques: guaranteeing that they satisfy differential privacy. Differential privacy [36] is a formal definition of privacy requiring that the output of a computation, for any two input datasets only differing in one value, does not reveal enough information for an adversary to learn something regarding the one difference between the two inputs. Specifically, differential privacy has one primary parameter ϵ, which bounds the maximum authorized difference of probability regarding the presence or absence of a value in the input dataset, given the observed output. The classic way to make a computation differentially private is to add a carefully tuned amount of random noise drawn from the Laplace distribution, to its output.

In this section, we discuss different propositions to make a collaborative-filtering system differentially private.

[76] proposes to generate a differentially private item covariance matrix which can then be used by standard techniques such as KNN or matrix factorization to

generate recommendations. This paper considers that the adversary is a user, and that the CF system is centralized and trusted. Whenever noise is added to a value, it is taken from a Laplace distribution of σ defined as in [36]. To obtain the covariance matrix, the authors first alter individual ratings in two step to make them more private.

In the first step, they compute the noisy sum of all ratings, and the noisy total number of ratings, where noisy means that random noise was added to each data point. These statistics yield the differentially private global average rating G. Similarly, the authors compute each item's differentially private average rating, with the difference that an arbitrary number of fake ratings (of value G) is systematically. These fake ratings do not significantly distort the average rating of popular items, while hiding real individual ratings for obscure items.

In the second step, the authors compute each user's differentially private average rating by (1) subtracting the relevant average item rating from each user's rating, and (2) adding a number of fake ratings as in the first step. These average user ratings only serve in centering every rating by subtracting them from each rating, then truncating the resulting centered ratings so that they all are contained in a smaller range.

These centered-then-truncated ratings are the final private ratings used when computing the covariance matrix. Moreover, when computing covariance between two items, each rating is weighted by the size of the relevant user's profile. Finally, some random Laplacian noise is added to each covariance value. For an improved recommendation quality, one can remove some of the redundant noise added during the different stages by unifying the noise variances and applying a low rank approximation.

[7] proposes three different ways to make systems based on matrix factorization differentially private. One way consists in preprocessing input ratings similarly as the previous paper. Another way is adding noise during the model generation via stochastic gradient descent. The last way consists in perturbing the output matrices of an alternating least squares algorithm.

The authors evaluate the three ways on MovieLens datasets, using $RMSE$. They find that input preprocessing yields the best recommendation quality. They also find that differentially private memory-based CF is able to cope with higher levels of noise than matrix factorization equivalents for the same recommendation quality. Conversely, for less strict privacy requirements, matrix factorization techniques are more accurate.

The authors of [48] propose D2P, a privacy-preserving protocol for memory-based CF systems ensuring a strong version of differential privacy. This protocol alters user profiles used for recommendation, making them private according to distance-based differential privacy (D2 privacy). The authors extend differential privacy so that a D2-private mechanism prevents anyone who see a recommendation for item i from guessing not only that any user has i in her profile, but also that her profile contains items similar to i within some distance. The D2P protocol consists in replacing some items i from the original profile, with some probability, with either items similar to i within some distance or random items, with some other probability. D2P still has good recommendation performances, and a low

computational overhead. It is applicable to systems regardless of their architecture. However, when the recommendation system is not decentralized, D2P requires users to trust the operator of the system, or of an intermediary server performing D2P, because these operators have access to users' original profile.

In summary, we saw that researchers became aware quite early of potential privacy issues caused by RSs, thus yielding a significant number of works proposing different privacy-preserving mechanisms. The main approaches are encryption, data obfuscation, and differentially-private ones.

Encryption of users' private data does not reduce recommendation quality, but burdens the RS with expensive computations. Data obfuscation consists in modifying users' ratings in a way which preserves most of their utility, while hiding original ratings. However, obfuscation must be applied careful in order to be effective, and it affects recommendation quality. Finally, differential privacy is a generic and well-defined way to evaluate privacy-preserving mechanisms by the correlation of their inputs and outputs, though it is not straightforward to adapt it to contexts where new information is often released, as is the case of recommendation.

Additionally, many of the aforementioned mechanisms are not fit for P2P systems, either because they were designed for centralized settings, or because they are too computationally expensive for individual peers.

Chapter 3

Preserving User Privacy in Decentralized Collaborative-Filtering Systems

In this chapter, we present *Hide & Share* (*H&S*) a novel mechanism for similarity computation which offers a reasonable level of privacy for users' profile. It is specifically designed to be computationally lightweight so that it can be applied to contexts requiring an important number of similarity computations, such as K-Nearest-Neighbors algorithms for fully distributed (a.k.a. peer-to-peer) systems.

We evaluate *H&S* using real data traces. We also demonstrate formally its privacy guarantees by computing an upper bound on the amount of information leaked by *H&S*'s similarity approximation. Our results show that *H&S*'s KNN provides a reasonable trade-off between privacy and utility. *H&S* disturbs similarity values but it does not significantly hamper the quality of the resulting recommendations. Approximate similarity values constitute instead an asset towards privacy preservation as they effectively prevent adversaries from performing profile reconstruction attacks as we show in Section 3.4.

In the remainder of this chapter, we first discuss the motivation behind this work in Section 3.1 before describing our system model in Section 3.2, and detailing our contribution in Section 3.3. Then we evaluate *H&S* experimentally in terms of recommendation quality, and overhead in Section 3.4. We analyze its privacy protection, empirically in Section 3.5, and formally in Section 3.6. Finally, we present our conclusions in Section 3.7.

3.1 Motivation

The observation from Chapter 1 that virtually all recommendation systems deployed nowadays are centralized, tells us that the privacy of users of these systems is under the threat of at least one type of adversary: "Big Brother" ones. Because the operator of a centralized RS is in control of users' data and/or the recommendation algorithms, users have no way to easily check that the operator operates her system as she claim she does, especially regarding potential privacy-preserving mechanism. Moreover, even if the operator is honest and implements strong privacy-preserving

mechanisms, she can be coerced into disabling or weakening them by entities such as intelligence agencies of the country they are incorporated in. Essentially, users of centralized RSs must put blind faith in these systems' operators, or stop to use them altogether when it comes to privacy concerns.

Therefore, one approach to avoid threats of "Big Brother" adversaries is to use decentralized recommendation systems. Deconcentrating data across peers makes it more difficult for content providers to access and possibly reuse personal data for purposes other than recommendation. Collaborative filtering techniques, and particularly memory-based ones lend themselves well to decentralized architectures because they generate recommendations using information which is local to the current user/item. We focus on user-based CF techniques in this thesis as neighbor users used to generate recommendations can straightforwardly be mapped to peers in a decentralized system.

Recent years saw the emergence of peer-to-peer (P2P) recommendation systems based on fully decentralized CF algorithms [16, 6]. Peer-to-peer recommendation systems are particularly scalable, and have therefore been proposed as a way to address not only the privacy issues but also the scalability issues that characterize centralized systems. Distributing computations across peers makes it possible to compute recommendations without requiring huge servers or data centers.

Although "Big Brother" adversaries do not exist in decentralized systems, performing collaborative filtering in a decentralized fashion is not free of privacy threats for users. Indeed, users must cooperate with each other to run the CF algorithm, but trusting all other users by default would be a mistake because anyone can participate in a decentralized system, including ill-intended users. Thus, it is necessary to adapt such algorithms so that they strike a delicate trade-off between cooperating with others, and holding back some information to avoid leaking users' data.

In the type of decentralized user-based CF systems we consider in this chapter, the three main elements requiring specific attention to avoid leakage of a user's privacy are her profile, her neighbors, and the items she forwards. Obviously, a user's profile should be revealed as little as possible, because it captures the user's interests. Neighbors of a user should not be freely available to any user, because when the neighbors of a user are selected based on the similarity of their profiles, knowing which users are similar to a target user enables an adversary to indirectly learn her target's profile via the profiles of the target user's neighbors. Finally, in the considered type of system, peers are responsible for item dissemination, which they usually do by forwarding items to each other, but for efficiency reasons they avoid forwarding all items they receive. A simple and efficient strategy is to only forward items they like. Hence, watching which items a target peer forwards is another indirect way to learn pieces of the target's profile.

So preserving users' privacy implies making changes to the first logical step of memory-based CF (finding the most similar users) to protect profiles and neighbors, as well as to the second logical step (selecting recommendations from candidate items) to handle correctly item forwarding. Privacy-preserving mechanisms addressing issues within each logical step deserve discussion on their own given the importance of both. In this chapter, we focus on privacy aspects of the first logical step, and specifically, our proposed mechanism.

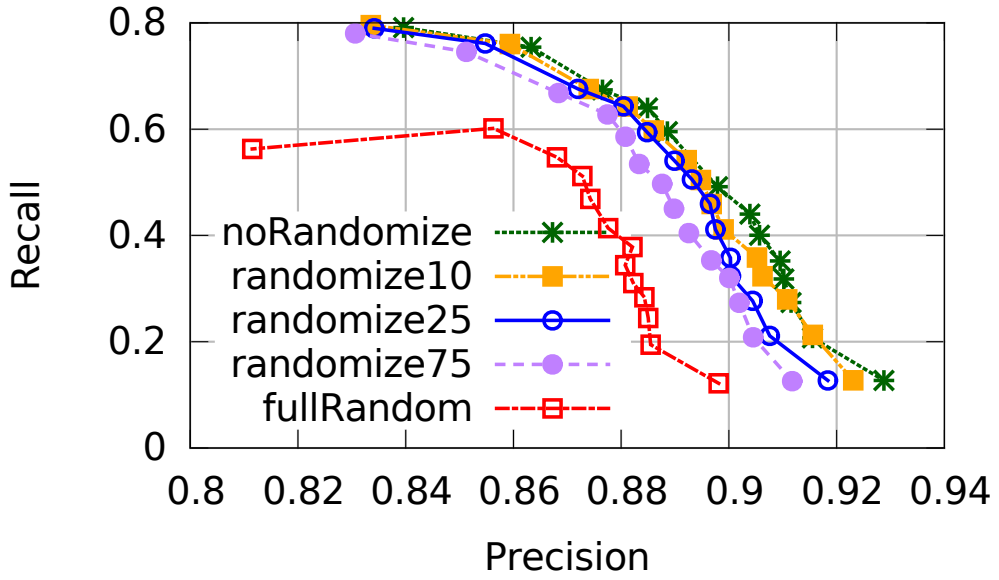

Figure 3.1 – Recommendation quality with different levels of randomization of user profiles. This quality is not significantly hampered by levels of randomization of up to 75%.

The most popular method implementing the first logical step of memory-based CF is K-Nearest-Neighbors (KNN) algorithms, a fundamental tool to mine and explore large amounts of data.

The decentralized KNN algorithms at the basis of most peer-to-peer recommendation systems [16, 6] require peers to exchange their profiles with other peers in order to compute similarity values. In doing so, they do not simply *risk* to share sensitive information; they systematically require users to share personal data with random other users. This makes it very easy for an adversary to learn about the interests of a large number of victims.

To address this challenge, we propose *Hide & Share* (*H&S*), a novel similarity mechanism for P2P KNN computation. *H&S* makes it possible to compute the KNN graph without requiring users to share their profile with anyone else. *H&S* relies on a simple observation: user-centric KNN applications such as recommendation do not require perfect knowledge. To illustrate this fact, Figure 3.1 depicts recommendation quality (quality increases towards the top and the right) with varying level of randomness injected into user profiles. Randomness injection consists in flipping with a 50% chance a number of bits in profiles which depends on the level of randomness. The plot shows that randomness levels of up to 75% do not significantly hamper recommendation quality. Each point within the same line represents the recommendation quality obtained when recommending a different number of items. Data is generated using our simulator with the ML-100k dataset, 10 neighbors per user, and profiles represented by Bloom filter-based compact profiles. We come back to this in Section 3.4.

Based on this observation *H&S* trades-off precision in the computation of similarity for privacy. This allows it to gain significant protection in terms of privacy

with a minimal impact on applications like recommendation. This makes *H&S* a perfect fit for decentralized CF systems.

H&S's key contribution lies in a novel *landmark-based* approximation technique as well as in a fair landmark-generation protocol. The landmarks of our solution allow two users to indirectly measure their similarity by comparing their own profiles with a set of randomly generated profiles (the landmarks). The similarity between a user's profile and a landmark acts as a coordinate in a coordinate system. Users then exchange vectors of coordinates and compute an approximation of their actual similarity. This preserves user privacy as users do not exchange their full profiles and landmark coordinates only reveal a limited amount of information about a user.

3.2 System Model

We present *H&S* in the context of a user-based peer-to-peer (P2P) collaborative-filtering system (CF) relying on gossip overlays. We start by reminding the reader how such a system works, and highlighting the corresponding privacy risks. We then present our adversary model in Section 3.2.3.

3.2.1 Decentralized User-based CF System

We consider a decentralized CF gossip-based system similar to that of [6], previously described in Section 2.1.3.3. Each user controls a single peer which stores her full profile as a list of ratings for the items she has rated. Ratings may consist either of binary values or of discrete values within a range (e.g. 1 to 5). In the following, we consider binary ratings as in most existing decentralized solutions [6, 9, 115, 18, 43, 16, 10].

The system uses *asynchronous rounds* that are executed periodically by each peer. In each round, each peer attempts to select a better set of similar other nodes (its *neighbors*) according to some similarity metric: for example *Cosine* similarity [94]. We recall that *Cosine* similarity considers profiles as high-dimensional vectors in which each unique item is a dimension and values for each dimension correspond to ratings. For more details, see Section 2.1.2.1.

In what follows, we first describe how the neighbors of a peer are identified in this model (*Neighbor identification*), before moving on to the actual mechanism used to recommend new items to users (*Recommendation*).

3.2.1.1 Neighbor Identification

Peers use two gossip protocols to identify their KNN: a random-peer sampling (RPS) and a clustering protocol. The former maintains a continuously changing topology, while the latter converges to the KNN graph, as illustrated in Figure 3.2. Both protocols follow the same high-level behavior. In each protocol, each peer maintains a data structure, called *view*, consisting of a list of references to other peers: the peer's current neighbors in the corresponding protocol. Periodically, a peer p contacts another peer q from this list and sends it a subset of its own view—half of its view in the RPS protocol, and its entire view in the clustering protocol. Upon

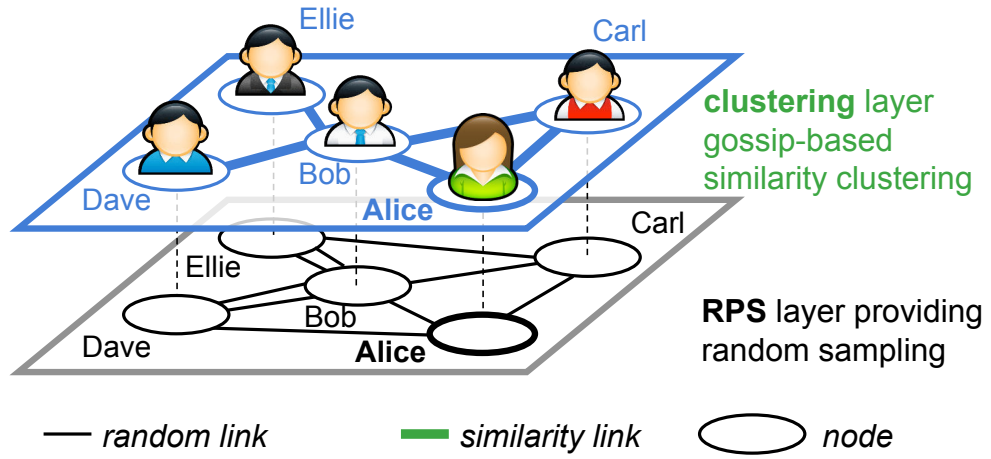

Figure 3.2 – Gossip-based distributed clustering

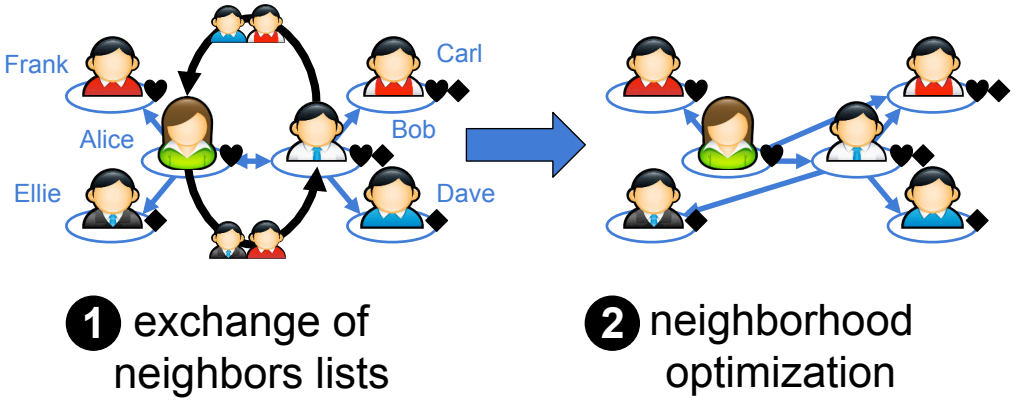

Figure 3.3 – Clustering mechanism for convergence to an optimal neighborhood. In this example, after exchanging their profiles, Alice and Bob modify their neighbors in order to be connected with the users who share the most their interests.

receiving such a subset, q merges the received subset with its own view. In the case of RPS, it keeps e random entries from the union of the two views. In the case of the clustering protocol, it keeps the k entries whose profiles are most similar to its own after combining its own clustering view, its own RPS view and the received clustering view. Then q replies by sending to p a subset of its view before the update, and p updates its view analogously. The clustering protocol provides each peer with a view that converges to its KNN. The RPS provides resilience to churn and partitions and ensure that the process cannot get stuck into a local minimum.

Figure 3.3 exemplifies the operation of the clustering protocol. Alice and Bob are interested in hearts, though Bob prefers diamonds. After exchanging their respective list of neighbors, they keep the users which are closest to their interests. In this example, Alice replaces Ellie with Carl who likes hearts, and Bob replaces Alice with Ellie who likes diamonds. After a few rounds of this protocol, each peer's neighborhood view contains the corresponding KNN.

3.2.1.2 Recommendation

Peers use the KNN identified with the above protocol to recommend items to their users. In typical systems, each peer identifies the items that were found most interesting by its KNN and to which the peer has not yet been exposed. In the case of binary rating, these consist of the items that were *liked* by the largest number of KNN and to which the peer has not been exposed.

3.2.2 Privacy Risks

As suggested, the above protocols require peers to share their profiles with each other in order to identify their KNN. This constitutes a major privacy risk: before convergence, both the RPS and the clustering protocol require peers to communicate with a large number of other peers, even with non similar ones. This means that a malicious non-similar peer can easily copy the profile of a target peer in order to forcibly enter its clustering view. In the rest of this chapter, we remove this privacy threat by introducing *H&S*, a novel similarity mechanism that does not require peers to exchange their profile information.

Thanks to *H&S*, peers can identify their KNN without having to disclose any personal details to other peers. Once they identified their KNN, they do share their profile information with neighbors that are sufficiently stable to compute recommendations as described in Section 3.2.1.2. However, this does not constitute a significant privacy risk because peers identified as KNN already know that they have similar profiles. Learning the details of each other's profiles therefore does not add much to this knowledge. Conversely, a malicious peer that wanted to become a neighbor of a target node would not be able to clone the corresponding profile without being already similar to the target peer.

3.2.3 Adversary Model

In the rest of this chapter, we consider a *curious adversary* model. Our adversary can only take a limited set of active actions to reach her goal, and can otherwise passively gather information. The goal of the adversary is to discover the profile of a chosen user (target) by a profile reconstruction attack, using information obtained during similarity computation. The adversary only controls one peer, *i.e.* we assume there is no collusion between adversaries, and our adversary cannot forge peer identities (no Sybil capacity). She also has no *a priori* knowledge regarding her target's interests. The active actions the adversary can take are: tap unencrypted communications; attempt to bias multi-party computations; compute her similarity with her target as many times as she wants.

3.3 The *Hide & Share* Landmark-based Similarity

We address the privacy issues in decentralized KNN computation by introducing *H&S* (*Hide & Share*), a novel mechanism for similarity computation. *H&S* relies on a simple observation: good recommendations do not require perfect neighborhoods.

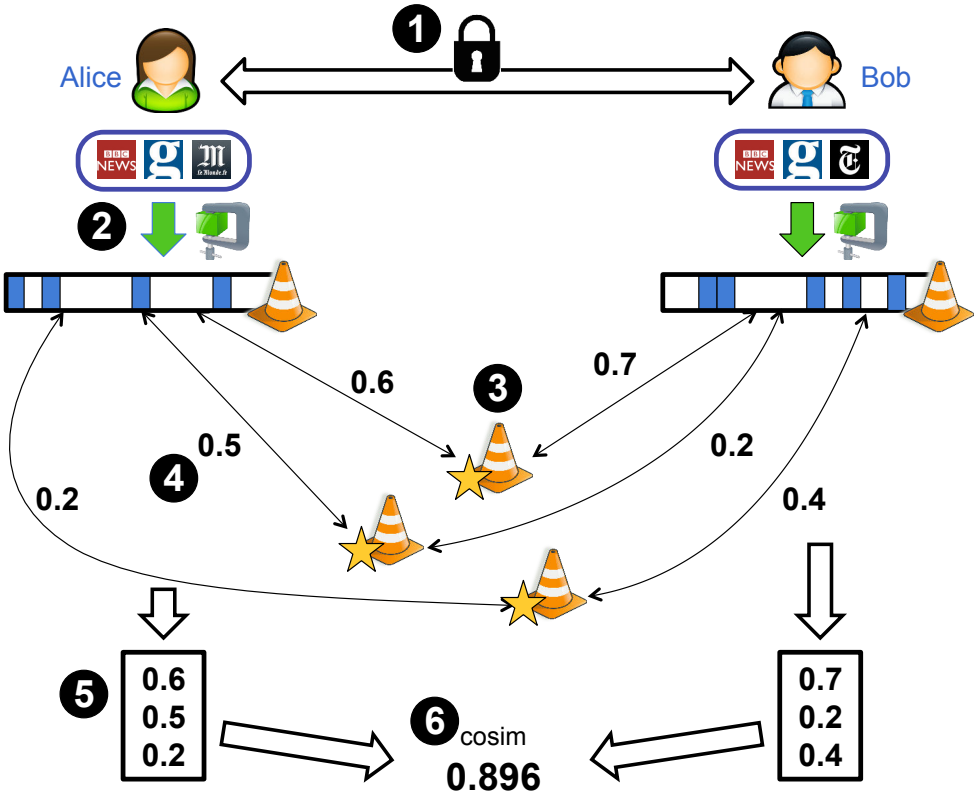

Figure 3.4 – Overview of the *H&S* similarity computation mechanism

H&S therefore relaxes the precision of similarity computation, by exploiting randomly selected intermediate profiles (*landmarks*) with respect to which each peer positions itself. This allows peers to compute similarity scores they can exploit without exchanging cleartext profiles.

H&S landmarks take inspiration from reference points in geo-localization systems. For instance, two-dimensional geographic locations usually refer to *the Equator* and *the Greenwich meridian*: two landmarks that define their latitude and longitude. However, our landmarks also exhibit two important differences with respect to this geographic analogy.

First our landmarks are not fixed and set for the whole system; rather, each pair of peers randomly generates its own set of landmarks. This prevents cross-pair comparisons. Second, we use far fewer landmarks than there are dimensions in our system. This prevents a precise reverse computation of each peer's cleartext coordinates (*i.e.* its profile) from its landmark coordinates. Thanks to these differences, users can safely exchange their landmarks because they do not characterize their interests in any specific topic.

Figure 3.4 presents an overview of the operation of *H&S* by means of an example. Alice and Bob need to compute their similarity with each other. In a traditional system like the one described in Section 3.2, Bob would send his profile to Alice and Alice would send hers to Bob. Each of them would then compute the similarity by applying *Cosine* similarity. With *H&S*, none of this happens. Rather, Alice and Bob follow these 6 steps. (1) They create a secure communication channel. (2) They each

derive a compact version (Bloom filter) of his/her profile. (3) They agree on a set of random landmarks. (4) They each compute the similarity of his/her compact profile with each landmark. (5) They each gather these similarity values in a similarity vector. (6) They exchange each other's similarity vector and compute their final similarity estimate. From a practical perspective, this translates into to two main components: a landmark generation mechanism, and a similarity approximation protocol. In the following we detail each of these two contributions.

3.3.1 Landmark Generation

H&S uses landmarks to estimate the similarity between two peers without requiring them to exchange their profiles with each other. To prevent adversaries from reconstructing profile information from these landmarks, the landmark generation mechanism must satisfy a set of requirements.

i *Computation confidentiality*: Only the two peers participating in the similarity computation may access the data they exchange. This includes landmark and similarity values.

ii *Independence of peer profiles*: Landmarks must be random and independent of the profiles of the peers that generate them.

iii *Fair landmark generation*: The choice of the landmarks must be fair. Neither of the two participating peers may bias the generated landmarks.

iv *Minimal information release*: An attacker should not be able to reconstruct a target profile by combining information from multiple landmark similarities, or by repeatedly computing its *H&S* similarity with the target.

In the following, we present our landmark generation mechanism by focusing on how it addresses each of these requirements. We detail the various steps in lines 1 through 18 of Algorithm 1.

3.3.1.1 Computation Confidentiality

Requirement (i) states that third-party peers should not be able to eavesdrop any communication between peers that are computing their similarity. To achieve this, *H&S* encrypts all the communication between two peers, including that relative to landmark generation.

Specifically, each peer maintains a public/private key pair. Peers exchange their public keys with each other by attaching them to the information transferred through the RPS and clustering protocols, similar to what was is done in [18]. In addition, we assume that peers may verify the authenticity of a public key by means of a certification authority or a web of trust [68, 43].

Peers use their key pairs to establish a secure communication channel whenever they need to evaluate their similarity. To this end, they exploit an authenticated key agreement (AK) protocol [13] as shown in lines 1 and 2. A possible AK protocol consists of an authenticated variation of the elliptic curve Diffie-Hellman key agreement such as the one available in the NaCl cryptographic library [8].

Algorithm 1 *H&S* landmark-based similarity computation protocol between peers $p1$ and $p2$, as executed by $p1$

1: $session\_key \leftarrow AK(key_{p1}, pub\_key_{p2})$
2: $secure\_channel \leftarrow connect(p2, session\_key)$
3: **if** $p2$ is known **then**
4: $s \leftarrow load\_seed(p2)$
5: **if** s is not older than th_L **then**
6: $seed \leftarrow s$
7: **goto** 15
8: **end if**
9: **end if**
10: **for all** i s.t. $0 \leq i < 32$ **do**
11: $r \leftarrow rand\_bit()$
12: $seed[i] \leftarrow coin\_flip(r, secure\_channel)$
13: **end for**
14: $save\_seed(p2, seed, timestamp(now))$
15: $prng \leftarrow init\_prng(seed)$
16: **for all** i s.t. $0 \leq i < L$ **do**
17: $\vec{M}_i \leftarrow generate\_lm(prng)$
18: **end for**
19: **for all** i in $0 \leq i < L$ **do**
20: $\sigma_{p1}[i] \leftarrow cosine(\vec{c}_{p1}, \vec{M}_i)$
21: **end for**
22: $send(\vec{\sigma}_{p1}, secure\_channel)$
23: $\vec{\sigma}_{p2} \leftarrow receive(secure\_channel)$
24: $similarity \leftarrow cosine(\vec{\sigma}_{p1}, \vec{\sigma}_{p2})$
25: **return** $similarity$

3.3.1.2 Independence of Peer Profiles

Requirement (ii) states that landmarks consist of *randomly generated* profiles that are independent of the profiles or of the choices of participating peers. However, as we discussed in Section 3.2, profiles consist of lists of item-score pairs, where the items belong to an unbounded or at least very large universe. This would make it difficult, if not impossible to generate random landmarks. To circumvent this problem, *H&S* replaces traditional profiles with *compact profiles* (step 2 in Figure 3.4).

A *compact profile* consists of a Bloom filter [14] and contains only the items considered as liked by the corresponding peer. A Bloom filter provides a compact representation of a set in the form of an array of n bits. To add an item to the set, the bloom filter applies h hash functions to the item to obtain h bit positions in the array and sets these positions to 1. To query for the presence of an item, the filter uses the same hash functions and checks if all the bits at the h indexes have a value of 1.

Compact profiles carry slightly less information than full profiles. First, Bloom filters can return false positives even though they never return false negatives. Second, compact profiles cannot distinguish between disliked items and items to which

the user has not been exposed. This does not constitute a problem: the like status of items proves sufficient to describe the interests of peers, and the effect of false positives may actually be beneficial in terms of privacy. Compact profiles also reduce *Cosine* similarity to counting the number of common bits between the two bloom filters.

Given a user or peer, $p \in \{1, 2, \ldots, N\}$, we denote her compact profile as $\vec{c}_p \in \mathbb{Z}_2^n$. Lines 10 through 18 of Algorithm 1 show how peers use compact profiles to generate random landmarks. Let L be a system parameter specifying the number of landmarks to generate and let PRNG be a pseudo-random number generator whose code is available to all peers (for example MRG32k3a [69] or Mersenne Twister [75]). Two peers, say $p1$ and $p2$, may generate a set of landmarks by first generating a common random seed (lines 10 to 13 in Algorithm 1). Then, each of them saves this seed (line 14), along with a timestamp, and uses it to initialize the PRNG (line 15). Finally Each of the two peers independently uses the PRNG to generate the L landmarks: $\{M_i\}$ with $i \in \{0, 1, \ldots, L\}$ (lines 16-18). Each generated landmark consists of a vector of bits of the same size as a compact profile, with a few random bits (around 5%) set to 1, while other bits are set to 0. This proportion of set bits mimics that of compact profiles, which are usually sparse.

3.3.1.3 Fair Landmark Generation

Requirement (iii) states that the choice of the landmarks must be fair. To achieve this, peers agree on their common seed using a bit-commitment scheme like Blum's coin-flipping protocol [15]. Blum's protocol operates as follows. Both $p1$ and $p2$ flip a coin. They set the output of the protocol to 1 if they obtain the same result, and to 0 otherwise. To exchange their coin-flip results without cheating, $p1$ and $p2$ employ a bit-commitment scheme. After flipping its coin, $p1$ sends $p2$ a commitment on its result ($f(concatenate(result, nonce))$). Then $p2$ reveals its result to $p1$, and $p1$ reveals its result as well as the nonce it used for the commitment to $p2$. $p2$ cannot cheat because it is the first to send its result. $p1$ cannot cheat because $p2$ can then check its result against the initial commitment.

Blum's protocol does not provide an unbiased coin, which is impossible in the two-party case [27], but a weaker fairness guarantee that suffices for our application. This guarantee holds as long as a malicious party does not abort the protocol before it ends. Since the two peers in our protocol use a secure channel, if $p2$ aborts, $p1$ can deduce that $p2$ is trying to bias the result.

3.3.1.4 Minimal Information Release

Requirement (iv) states that attackers should not be able to reconstruct a target profile by combining information from multiple landmarks or by repeatedly computing their similarity with the target. To satisfy the first part of this requirement, *H&S* similarity uses a small number of landmarks with respect to what would be required to reconstruct the original profile. In Section 3.5, we show that this does not significantly impact the ability to provide good recommendations.

To satisfy the second part of this requirement, *H&S* peers do not generate new landmarks each time they meet. Rather they only do so if their latest common set of

landmarks is older than a threshold, th_L. To achieve this, they verify the timestamp associated with their latest saved common seed. If the timestamp is newer than the threshold, then they reuse the seed, otherwise they generate a new random seed.

3.3.2 Similarity Approximation

We conclude the description of our protocol by presenting how *H&S* approximates the similarity between two peers using its randomly generated landmarks. Let $\{M_1, \ldots, M_L\}$ be a set of common landmarks known to peers $p1$ and $p2$. First, each of the two peers independently computes its similarity with each of these landmarks (step 4 in Figure 3.4 and lines 19-21 in Algorithm 1). This consists in applying *Cosine* similarity to its own profile and each of the landmarks. Both $p1$ and $p2$ then store the results of these computations in a similarity vector (respectively $\vec{\sigma}_{p1}$ and $\vec{\sigma}_{p2}$) as shown in step 5 in Figure 3.4 and on line 20 in Algorithm 1. Second, $p1$ and $p2$ exchange their similarity vectors with each other. This consists of lines 22 and 23 in Algorithm 1. Finally (step 6 and line 24), $p1$ and $p2$ compute their *H&S* similarity by applying *Cosine* to their own similarity vector and to the one they have received (note that $cos(\vec{A}, \vec{B}) = cos(\vec{B}, \vec{A})$).

3.4 Recommendation Evaluation

We evaluate *H&S* by applying it in the context of a gossip-based decentralized recommendation system. Using publicly available traces, we evaluate the quality of its recommendations, and the overhead it implies.

3.4.1 Methodology

3.4.1.1 Simulator

We use our own simulator written in Java. The simulator takes as input a trace from a recommendation system, consisting of user-item matrix of ratings, split into a training set and a test set. The training set (80% of the ratings) allows peer neighborhoods to converge, while the test set (the remaining 20%) provides the ground truth to evaluate the relevance of recommendations. The simulator operates in two steps. First it uses the training set to simulate the convergence of the clustered overlay, then it generates r recommendations for each peer using the converged overlay and compares the results with the ratings in the test set.

3.4.1.2 Datasets

Table 4.1 outlines the characteristics of the three traces we use. ML-100k[1] and ML-1M[1] are traces from the MovieLens [52] online movie-recommendation service. They contain 100,000 and 1,000,000 ratings respectively. Jester-1-1[2] is a trace for the Jester [46] online joke-recommendation service. It is the first third of Jester's

Table 3.1 – Characteristics of the traces in terms of number of users, number of items, number of ratings and rating range.

	# users	# items	# ratings	Rating range
ML-100k	943	1,682	100,000	$[1:5]$ (integers)
ML-1M	6,040	3,900	1,000,000	$[1:5]$ (integers)
Jester-1-1	24,983	100	1,810,455	$[-10:10]$ (continuous)

dataset-1.

3.4.1.3 Evaluation Metrics

We evaluate recommendation quality in terms of *Precision* and *Recall* [94], which we defined in Section 2.1.2.1. We recall them now, in this chapter's context. The former evaluates whether peers like the recommendation they receive. The latter evaluates if recommendation cover all the interests expressed by the ground truth in the test set.

$$precision(user) = \frac{\|recommendedItems \cap likedItems\|}{\|recommendedItems\|}$$

$$recall(user) = \frac{\|recommendedItems \cap likedItems\|}{\|likedItems\|}$$

We consider an item as liked when its rating is greater than or equal to a threshold value which depends on the rating range in use.

Neighborhood quality evaluates how much the neighborhoods provided by *H&S* resemble the optimal neighborhoods, that is those obtained with the standard cosine similarity metric. Specifically, for each user we measure the average of the cosine similarities with all the peers in its *H&S* view, and we normalize it by the average cosine similarity with the peers in the optimal neighborhood obtained using an exhaustive search procedure. Let u be a user with full profile, $profile_u$, and let n_u and N_u be respectively u's *H&S* neighborhood and u's optimal neighborhood. Then we compute u's neighborhood quality as follows.

$$quality(u) = \frac{\frac{1}{k}\sum_{p \in n_u} cos(profile_u, profile_p)}{\frac{1}{k}\sum_{p \in N_u} cos(profile_u, profile_p)}$$

Neighborhood quality provides a first indication of privacy: lower quality implying better privacy.

Finally, we evaluate overhead by comparing the bandwidth consumption and the storage space required by a *H&S*-based recommendation system with those required by a standard implementation like that of the reference model described in Section 3.2.

[1]MovieLens datasets are available at: `http://grouplens.org/datasets/movielens/`

[2]Jester datasets are available at: `http://eigentaste.berkeley.edu/dataset/`

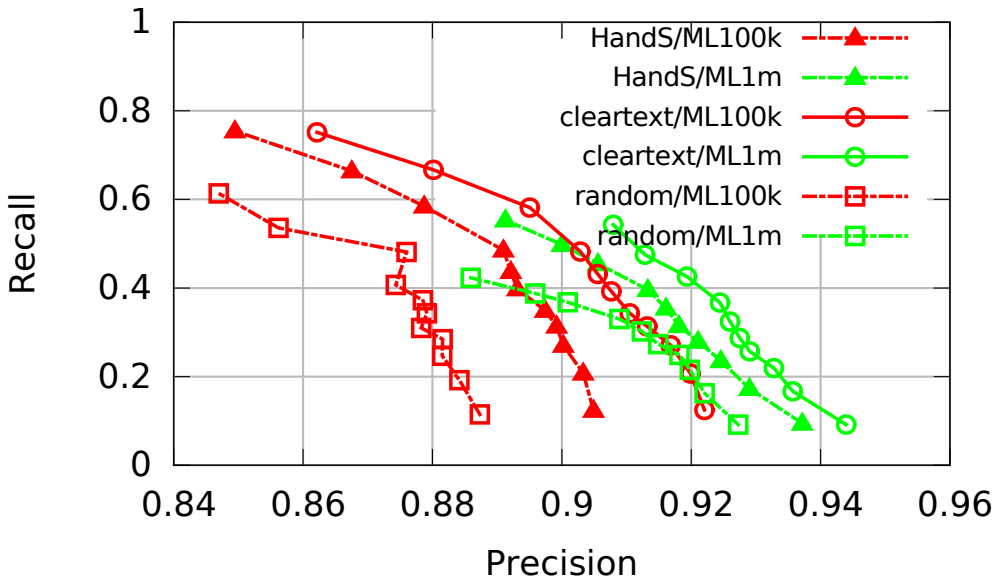

Figure 3.5 – Recommendation quality expressed as precision and recall with a varying number of recommendations r, using the MovieLens datasets.

3.4.1.4 Default Parameters

The subsequent results correspond to simulations using neighborhood and RPS view sizes of 10 peers. Compact profile sizes depend on the dataset used: 660 and 1473 bits for ML-100k and ML-1M respectively (roughly 40% of the number of items), and 99 bits for Jester. When the number of landmarks is not explicitly mentioned, *H&S* uses 50 landmarks. This represents a good trade-off between recommendation quality and privacy. For all the metrics except set score, we plot values averaged over all the peers.

3.4.2 Recommendation Quality

We evaluate the quality of recommendations provided by an *H&S*-based system using *Precision* and *Recall*, as defined beforehand in Section 3.4.1.3. We use dataset-dependent threshold rating values when considering whether an item is liked: $rating \geq 3$ for MovieLens and $rating \geq 0.0$ for Jester. Using user-dependent threshold values such as the average rating, the median rating, or the half of the rating range for each peer results in similar or lower precision/recall values. This suggests that users tend to use the available rating range similarly.

Peers recommend the r most liked item in their neighborhoods, not including those they have already rated. We check whether a recommended item is liked by looking at the rating given to this item by the recipient in the test set.

Figures 3.5 and 3.6 show precision and recall values for several values of r. The former shows the results with the MovieLens datasets, and the latter shows the results with the Jester dataset. For each dataset, we compare the results of the *H&S*-based system (triangle-shaped) with a lower bound (square-shaped) and a cleartext baseline (circle-shaped). The lower bound consists of a CF system that

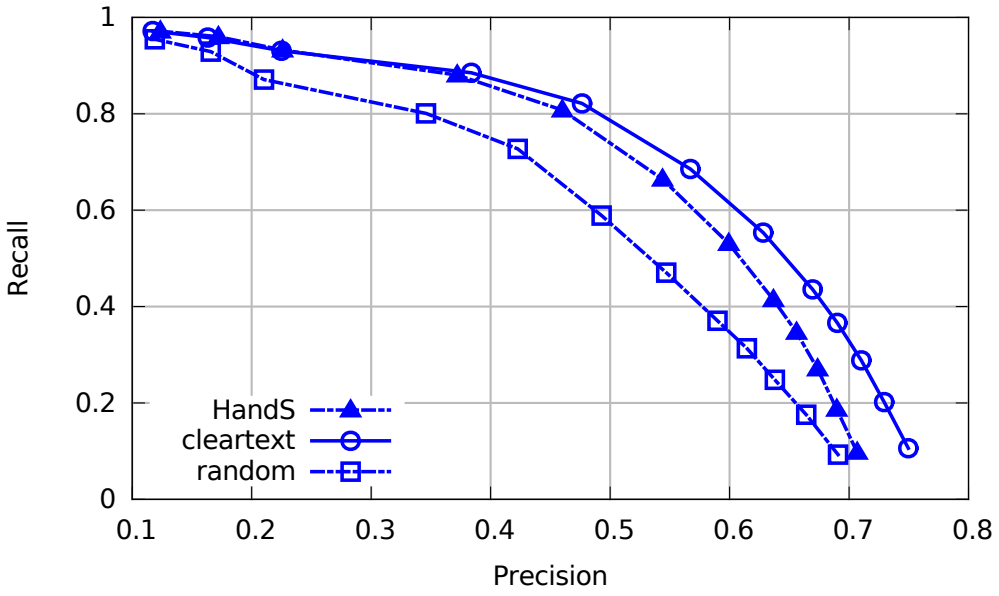

Figure 3.6 – Recommendation quality expressed as precision and recall with a varying number of recommendations r, using the Jester dataset.

uses completely random neighbors. The baseline consists of the reference model with full profiles in cleartext, as described in Section 3.2. The absolute values of recall and precision are quite high even with random neighborhoods because we do not consider items for which a user has no rating in the original dataset as potential recommendations. More generally, absolute values of precision and recall depend on the predictability and regularity of the dataset, and their acceptable levels depend on the application.

Figure 3.5 shows consistent results by the *H&S*-based system across the two MovieLens datasets. *H&S* provides a reasonable quality of recommendations: it never suffers from a degradation of more than 50% with respect to the cleartext baseline. Moreover the higher the value of r, the closer the quality remains to that of the cleartext baseline.

Figure 3.6 shows a similar behavior of the *H&S*-based system with the Jester datastet. Recall reaches almost a value of 1 because the dataset only contains 100 items. This characteristic is also the cause of the maximum precision values being lower than those of the MovieLens datasets. As the test set does not contain many items, we consider that a recommended item without rating in this set is disliked by the recipient, instead of ignoring it as done otherwise. Although this approach is pessimistic, it allows us to make a sufficient number of recommendations.

We showed that *H&S* preserves the quality of recommendation, being only slightly worse than the cleartext baseline. In the following, we show that it achieves this while protecting the privacy of users.

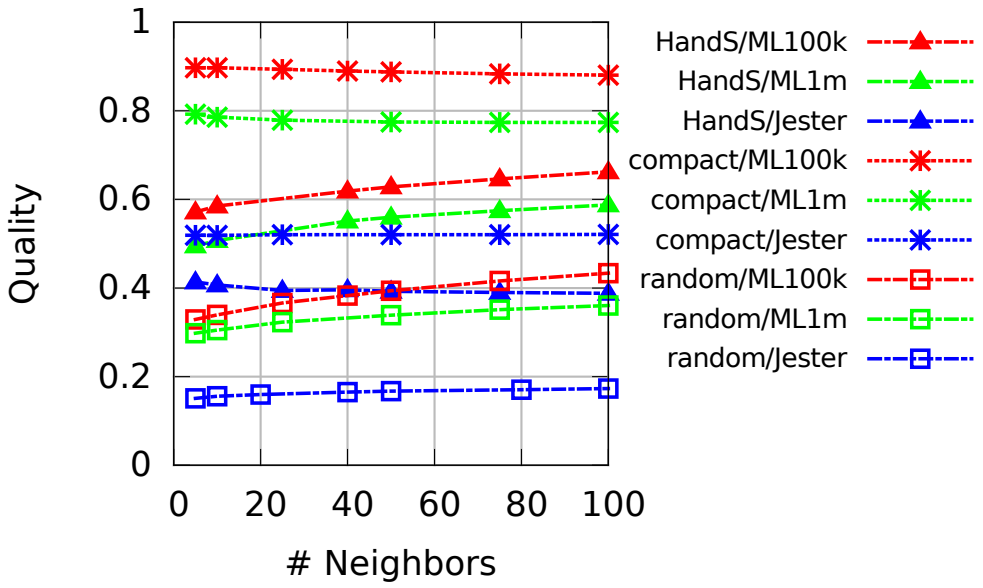

Figure 3.7 – Effect of compact profiles and the *H&S* similarity on neighborhood quality. The *H&S* similarity is the main source of perturbation of neighborhood quality.

3.4.3 Neighborhood Quality

In order to evaluate the extent to which neighborhoods are different from the optimal neighborhoods, we use the neighborhood quality measure as defined in Section 3.4.1.3.

Figure 3.7 shows the evolution of neighborhood quality with the size of neighborhoods. For each dataset, it compares the *H&S*-based system (triangle-shaped) with a CF system using random neighbors as a lower bound (square-shaped) and a variant of our system model using compact profiles (star-shaped). Our reference model from Section 3.2 by definition achieves a neighborhood quality of 1 and compact profiles provide neighborhoods that are almost identical in the ML datasets. In the case of Jester, they lower neighborhood quality by 50% because the Jester dataset contains only a few items. This makes it more sensitive to the collisions in the Bloom filters.

H&S similarity has a more significant impact on neighborhood quality than compact profiles. Yet, *H&S*'s neighborhood still retain their utility in terms of recommendation as we showed in Section 3.4.2. Because landmarks are randomly generated, some of them might be "far" from the two users comparing themselves, thus giving little information about the users' similarity. Moreover, a set of landmarks is not necessarily linearly independent. The lower quality of *H&S*-generated neighborhoods is in fact an asset in terms of privacy. Because of this mix of neighbors with various levels of similarity, the adversary cannot infer her target's interests just by looking at her target's neighbors.

3.4.4 Overhead

We evaluate the overhead caused by *H&S* to peers by comparing its bandwidth consumption and storage requirement with those of the reference model.

Overall, *H&S* incurs most of its overhead when two peers compute their similarity for the first time because they have to generate a seed using the bit commitment scheme and store this seed. So we measure in our simulations the number of similarity computations with new peers. The other parameters influencing *H&S*'s overhead are the sizes of the RPS and neighborhood views, and the number of landmarks.

3.4.4.1 Overhead Computation Methods

Bandwidth consumption of both the *H&S*-based system and the reference model depend on the number of messages exchanged in the random-peer sampling and clustering protocols. Both systems perform the same protocols from a high level point of view, but the size of the protocols' logical messages are different.

In a round of the RPS protocol, a peer p sends half of its RPS view to q, a randomly chosen peer from p's RPS view. Then, q sends half of its view to p. So the bandwidth cost for p and q alike is $e \times peerInfo$, where e is the number of peers in a RPS view, a.k.a. its size. In the *H&S*-based system, $peerInfo$ is the tuple $< peerID, IP, port >$, where $peerID$ is a 64 bits ID, and the peer's IPv4 and port take 48 bits. In the reference model, $peerInfo$ is the same tuple, plus the profile of the exchanged peers, and a timestamp to check the "freshness" of this version of the profile. To simplify calculations, we consider that every profile contains the same number of ratings. We use for this value the average number of ratings per user in each dataset.

In a round of the clustering protocol, a peer p sends its whole neighborhood view to q, the peer from p's neighborhood for which the last contact timestamp is the oldest. Then, q sends its neighborhood view to p. Finally, p (q respectively) computes its similarity with every other peer in its RPS view, in its neighborhood view, and in the view sent by q (p resp.), so as to determine the k most similar peers to be kept in the new neighborhood view of p (q resp.). So the bandwidth cost for p and q alike is $2k \times peerInfo + (2k + e) \times similComp$, where $similComp$ is the bandwidth cost of a similarity computation. For both systems, $peerInfo$ is the same value as in the RPS protocol, plus a similarity value encoded as an IEEE 754 double precision floating point number.

In the *H&S*-based system, the value of $similComp$ is the cost the authenticated key agreement protocol, plus a cost depending on whether the two peers have already generated a common set of landmark. If the two peers have already met previously, they only have to exchange their landmark coordinates vector, so the cost is $2 \times m \times 64$, where m is the number of landmarks used. If the two peers have never met before, there is the additional cost of landmark generation. We consider that the landmarks' seed is a 32 bits integer, so the cost of a landmark generation is $32 \times bitCommit$, where $bitCommit$ is the cost of a bit-commitment scheme. For instance, the cost of Blum's coin-flipping protocol is $256 + 1 + 80$ when using nonces of 80 bits and SHA-256 as the one-way function for committing.

In the reference model, the value of $similComp$ depends on whether the profile of

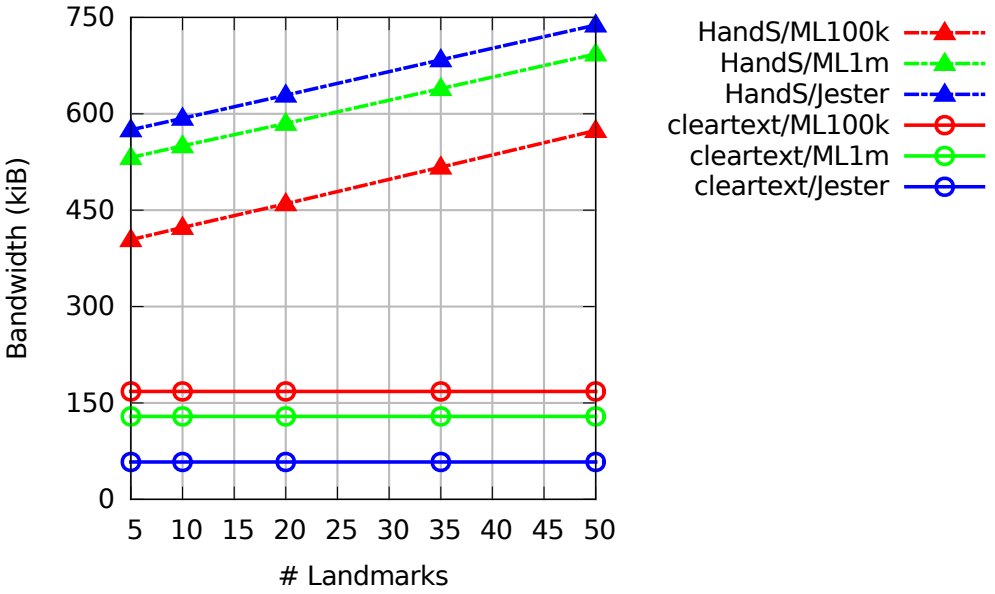

Figure 3.8 – Average bandwidth consumption of a peer per gossip round. The *H&S*-based system consumes roughly twice to seven times more bandwidth than our system model with 5 to 50 landmarks.

q cached by p is up-to-date, and conversely. $similComp$ is at least 128 bits because each peer checks the timestamp of last modification of the other's profile. Then, $similComp$ increases by one or two profile size accordingly.

Storage requirements for the *H&S*-based system and the reference model are different. In the former, a peer p stores one $peerInfo$ per peer in its neighborhood and RPS views, and one 32 bits seed per known peer (even peers which are not currently in p's views) for landmark generation. In the reference model, p also stores one $peerInfo$ per peer in its views, but it includes the peer's timestamped-profile in order to save on bandwidth and computation when a peer's profile does not change between two similarity computations.

3.4.4.2 Bandwidth and Storage Overhead

The main factors impacting *H&S*'s bandwidth consumption are the exchange of landmark coordinates vectors and the bit commitment scheme. The main factor impacting *H&S*'s storage requirement is the need to store seeds.

Figure 3.8 compares the *H&S*-based system (triangle-shaped) and the reference model (circle-shaped) in terms of the average bandwidth consumption of a peer per gossip round. Bandwidth consumption of the *H&S*-based system increases linearly with the number of landmarks used. It consumes roughly twice to seven times more bandwidth than the reference model, but the absolute values remain reasonable (up to 700KiB per round). Moreover, it can probably be improved as a bit commitment protocol with $O(1)$ bits of communication per committed bit exists [80]. It should also be noted that bandwidth consumption values for *H&S* are based on the average number of similarity computations with new peers, made by a peer during

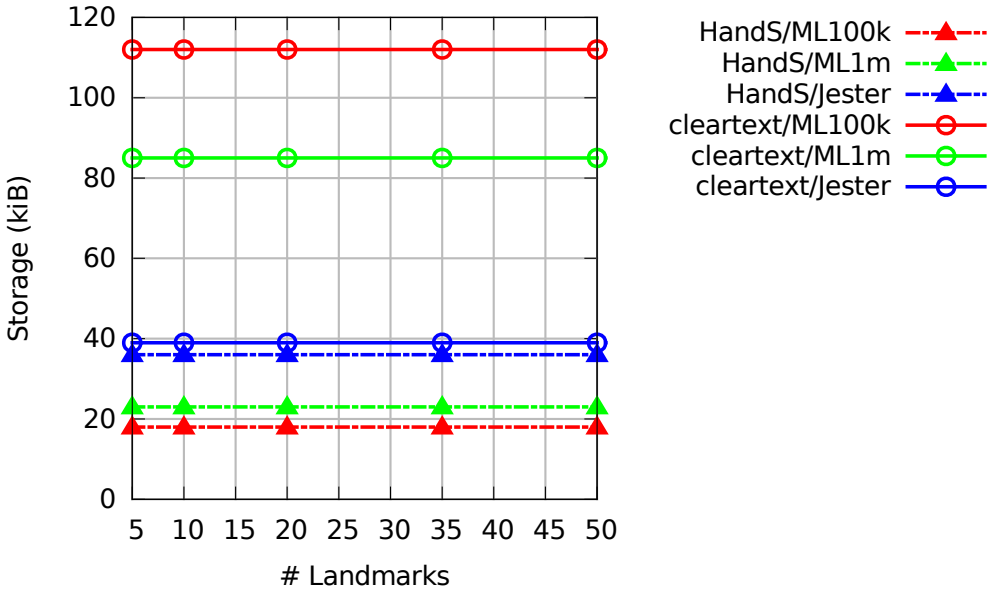

Figure 3.9 – Average storage space needed for a peer. The *H&S*-based system needs less storage space because peers only store the seed used to generate landmarks.

the whole simulation, divided by the number of gossip rounds performed during the simulation. On the one hand, it means actual values are higher when a peer joins the systems, because it does not know any other peer yet, thus incurring more landmark generations. On the other hand, the longer a peer stays in the network, the lower the actual values until the peer's neighborhood view converges to the optimal KNN. Once a peer reaches its converged state, the rate of similarity computations with new peers remains the same on average because new peers only appear in the peer's RPS view.

Figure 3.9 compares the *H&S*-based system and the reference model in terms of the average storage required by a peer. The *H&S*-based system needs less storage space because peers only store the seed used to generate landmarks instead of storing the profiles of peers in their neighborhood and RPS views as done by standard systems. Still, we observe that the required storage space is tiny compared to the storage capacity of modern devices (computers, smartphones, tablets, *etc*).

3.4.4.3 Computational Overhead

We observe that the computational overhead of *H&S* is negligible from the point of view of peers. The most computationally intensive elements of the *H&S* similarity are *(1)* the authenticated key agreement (AK) protocol, *(2)* the generation of random bits (bit commitment scheme and mostly landmark generation), *(3)* the cosine similarity computations with landmarks.

(1) Executing cryptographic primitives incurs negligible cost on modern devices. It is similar to accessing a website with HTTPS: the end user does not perceive a difference between accesses over HTTP and HTTPS. *(2)* Efficient PRNGs such as Mersenne Twister can generate millions of bit/second on modern devices and *H&S*

only needs a few thousands of bits to generate landmarks during a gossip round, which lasts several seconds at least. *(3)* Cosine similarity is cheap to compute on binary vectors such as landmarks and compact profiles. This confirms the applicability of our approach.

3.5 Empirical Privacy

Using the same methodology as defined in Section 3.4.1, we empirically evaluate the privacy offered by *H&S* by running a profile reconstruction attack against it. This attack consists in trying to discover the liked items in a targeted peer's profile using information obtained during similarity computation.

We quantify the resilience of *H&S* to such attacks with the *setScore* defined as follows. This metric measures the success rate of the adversary in the context of a profile reconstruction attack. Let G be the set of items that the adversary guesses as liked by the target, and let P be the set of items actually liked by the target. We then define *setScore* as follows, with $\triangle$ as the symmetric difference of two sets.

$$setScore(G, P) = \frac{|G \triangle P| - |G \cap P|}{|G \cup P|}$$

A *setScore* of 1 (adversary's failure) indicates that all the guessed items are wrong (highest privacy), while a *setScore* of -1 (adversary's success) indicates the adversary guessed exactly the target's liked items (no privacy).

We evaluate *H&S*'s resilience to a basic profile reconstruction attack, and an advanced one using linear programming.

3.5.1 Basic Attack

For this attack, the adversary makes her guess in two steps: *(1)* she tries to infer her target's compact profile, then *(2)* she tries to deduce the items forming this profile. We consider for *(1)* that the adversary uses the closest landmark to her target as her guessed profile. For *(2)*, we consider that the adversary knows all the items in the system, so she includes in her guessed set all the items matching the guessed profile.

We compare our *H&S*-based system with a perturbation-based privacy technique. When using this technique, peers compute their similarity by the usual profile exchange, but they add random noise to their profile to protect their privacy. For the sake of comparison, peers implement this technique by using compact profiles and randomizing a certain percentage of bits in the profile.

Figure 3.10 compares our *H&S*-based system and a recommendation system using randomized compact profiles, in terms of the trade-off between recommendation quality and privacy. We use *setScore* for the latter and $F1$-*measure*, the harmonic mean of precision and recall, for the former (see Section 2.1.2.1 for details regarding $F1$-*measure*). We obtain different values of this trade-off by varying the number of landmarks from 2 to 100 for the *H&S* system, and by varying the number of randomized bits in profiles from 5% to 100% for the perturbation-based system.

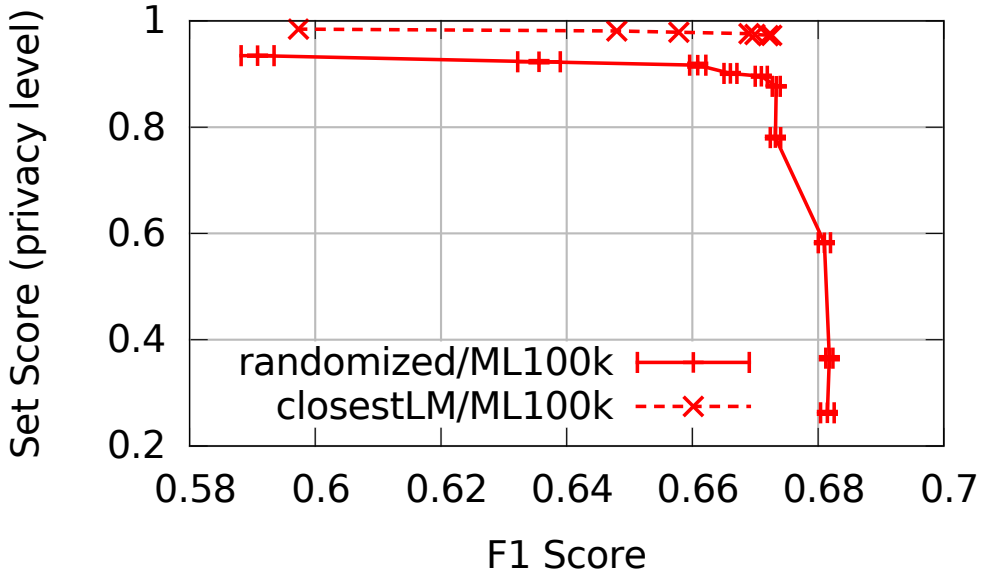

Figure 3.10 – Trade-off between recommendation quality and privacy for the *H&S*-based system and a system with perturbation-based privacy.

setScore values are averages over 100 different adversaries and 200 different targets, i.e. 20,000 different sets of landmarks. F1 score values correspond to $r = 30$ recommendations.

We observe that the *H&S*-based system provides an excellent level of privacy in any case. It also provides a recommendation quality on par with the best values of the other system, starting from 25 landmarks. However, the increase in recommendation quality does not grow as fast as increase in the number of landmarks.

The recommendation system using randomized compact profiles preserves an almost optimal recommendation quality with up to 75% of randomized bits. Although it achieves reasonable privacy ($setScore = 0.8$ approximately) starting from 50% of randomized bits, it never reaches the privacy levels offered by the *H&S*-based system. Even 100% of randomized bits does not yield a *setScore* of 1 because the attacker tries to match all item signatures against the randomized profile. In general, a fully randomized compact profile will contain more bits with value 1 than a landmark. This will cause the attacker to identify more potentially matching items.

With this basic strategy for the profile reconstruction attack, we showed that *H&S* provides improved privacy to users without sacrificing recommendation quality, and without obvious flaws.

3.5.2 Advanced Attack Using Linear Programming

Alternatively to the basic strategy for the adversary that we describe above, we discuss an advanced variant which tries to make a smarter guess of the target's compact profile using linear programming.

For *(1)*, we consider that the adversary uses her knowledge of the landmarks and her target's corresponding similarity values (landmark coordinates) to infer

a compact profiles resembling her target's profile. The adversary formulates her knowledge as a linear programming problem and uses a solver to find such a compact profile. A linear programming (LP) problem consists in maximizing/minimizing a linear function:

$$f(x_1, x_2, \ldots, x_n) = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

subject to linear (in)equality constraints:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq b_2 \end{aligned}$$

where $x_1, x_2, \ldots, x_n$ are non-negative variables.

The adversary's strategy for step *(1)* is a LP minimization problem modelled as follows:

$$min(f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i$$

subject to:

$$\begin{aligned} \textstyle\sum_{j \in lm_1} x_j = d_1 \\ \textstyle\sum_{j \in lm_2} x_j = d_2 \\ \cdots \\ \textstyle\sum_{j \in lm_L} x_j = d_L \end{aligned}$$

where n is the number of bits in compact profiles and landmarks, L is the number of landmarks, $lm_1, \ldots, lm_L$ are the sets of indices of positive bits in each landmark, and $d_1, \ldots, d_L$ are the dot product part of cosine similarity values from landmark coordinates.

The linear function corresponds to finding the sparsest compact profile satisfying the constraints. The constraints require the solution to result in the same landmark coordinates as the target's ones. These constraints use the dot product part (numerator) in the cosine similarity formula as it is easier for writing LP constraints. The adversary can compute the dot products $d_1, \ldots, d_L$ by discovering her target's compact profile norm using an exhaustive search. Landmarks and compact profiles being binary vectors, this makes the exhaustive search manageable because binary vectors simplify the dot product into computing the number of positive bits at the same indices in the two vectors, and the Euclidean norm into the square root of the sum of positive bits.

These simplifications also imply that only landmarks' positive bits provide information about the corresponding bits in profiles. We call *unconstrained bits* the bits in compact profiles for which all landmarks have a value of 0 at the corresponding indices. Because the adversary has no information about unconstrained bits, she cannot guess their value by solving the LP problem. So we consider that she assumes all unconstrained bits to be 0 as it is the most likely value given profiles are generally sparse.

However, this LP-based attack strategy does not work for two reasons. First, the linear programming formalism is not expressive enough to model the problem to be solved as closely as we want. The current linear function try to find the sparsest solution vector satisfying the constraints, while what we want is just one

or several solutions satisfying them. Moreover, we would ideally like to express constraints saying that the combinations of bits constituting a solution should match the combinations produced when representing items in a compact profile (Bloom filter). This indicates that modeling our goal as an optimization problem is not the ideal approach.

Second, even if this strategy can work with small examples, it does not scale to actual parameters. Often, LP solvers, including state-of-the-art commercial ones such as Gurobi, are unable to solve the problem when using compact profiles of hundreds of bits, and tens of landmarks.

3.6 Privacy Guarantee

We saw that even the aforementioned advanced profile reconstruction attack is not able to efficiently exploit the information from landmarks and coordinates vectors. However, this does not mean that a practical advanced attack is unachievable through another approach than linear programming with greater expressiveness, and computational efficiency.

Therefore, we now analyze *H&S* from an information theoretical viewpoint and compute an upper bound on the amount of information leaked during our landmark-based similarity computation. We carry out our analysis from the point of view of an attacking peer, a, that seeks to obtain information about another peer, p.

During the protocol, a, and p share three pieces of information: the common seed they agree upon, the landmarks, $\{M_1, \ldots, M_L\}$, they generate using this seed, and the similarity vectors $\vec{\sigma}$ containing their similarity with respect to $\{M_1, \ldots, M_L\}$. The first two of these items do not depend on the profile of p and thus do not disclose any information. So we concentrate our analysis on the information that $\vec{\sigma}_p$ may leak about the corresponding compact profiles.

3.6.1 Information Leakage Measured as Conditional Entropy

We start our analysis by obtaining a first expression for the amount of information leaked by our landmark-based similarity computation. From the attacker's perspective, we define C as the random variable for p's compact profile, with realization $\vec{c}$. Let $\vec{\sigma}$ be the vector of similarity values between $\vec{c}$ and each of the landmarks in the landmark matrix. According to the definition of cosine similarity, we have $\sigma_i = \cos(\vec{c}, M_i) = \frac{\vec{c}M_i}{||\vec{c}|| \cdot ||M_i||} \forall i \in \{1, ..L\}$.

Let us now define an adjusted similarity vector $\vec{v} = \{v_1, ..., v_n\}$, such that $v_i = \sigma_i \cdot ||M_i||$. Then, $v_i = \frac{\vec{c}}{||\vec{c}||} \cdot M_i$. The goal of an attacker is to guess $\vec{c}$ based on the knowledge of M and $\vec{\sigma}$. But knowledge of M and $\vec{\sigma}$ implies knowledge of $\vec{v}$, while knowledge of $\frac{\vec{c}}{||\vec{c}||}$ implies knowledge of $\vec{c}$ because $\vec{c}$ is a binary vector. We can therefore analyze the case of an attacker that tries to guess $\frac{\vec{c}}{||\vec{c}||}$ based on $\vec{v}$ and M.

To this end, we define W as the random variable for p's normalized compact profile, with realization $\vec{w} = \frac{\vec{c}}{||\vec{c}||}$. We also define V as the random variable for the corresponding adjusted similarity vector with realization $\vec{v}$, and Mt as the random variable for the landmark matrix with realization M.

We can then express the uncertainty about W given V and Mt through the conditional entropy $H(W|V, Mt)$. Such uncertainty corresponds to the amount of information protected from the adversary. According to the definition of conditional entropy, we have:

$$H(W|V, Mt) = \sum_{\vec{w},\vec{v},M} p(\vec{w}, \vec{v}, M) \log \frac{p(\vec{v}, M)}{p(\vec{w}, \vec{v}, M)}. \tag{3.1}$$

We can then express $p(\vec{w}, \vec{v}, M)$ as follows.

$$\begin{aligned} p(\vec{w}, \vec{v}, M) &= p(\vec{v}|\vec{w}, M)p(\vec{w}, M) \\ &= \begin{cases} 1 \cdot p(\vec{w}, M) & \text{if } \vec{v} = \vec{w}M \\ 0 & \text{if } \vec{v} \neq \vec{w}M \end{cases} \end{aligned} \tag{3.2}$$

This allows us to rewrite Equation (3.1).

$$\begin{aligned} H(W|V, Mt) &= \sum_{\vec{w},\vec{v},M, \text{ s.t. } p(\vec{w},\vec{v},M)\neq 0} p(\vec{w})p(M) \log \frac{p(\vec{v}, M)}{p(\vec{w})p(M)} \\ &= \sum_{\vec{w},\vec{v},M, \text{ s.t. } p(\vec{w},\vec{v},M)\neq 0} p(\vec{w})p(M) \log \frac{p(\vec{v}|M)}{p(\vec{w})} \\ &= \sum_{M} p(M) \sum_{\vec{w}} p(\vec{w}) \sum_{\vec{v}=\vec{w}M} \log \frac{p(\vec{v}|M)}{p(\vec{w})}. \end{aligned} \tag{3.3}$$

We can split Equation (3.3) into two parts, using the fact that $log(\frac{a}{b}) = log(a) - log(b)$. Let $H(W|V, Mt) = J + K$, we have

$$J = \sum_{M} p(M) \sum_{\vec{w}} p(\vec{w}) \sum_{\vec{v}=\vec{w}M} \log p(\vec{v}|M) \tag{3.4}$$

$$K = \sum_{M} p(M) \sum_{\vec{w}} p(\vec{w}) \sum_{\vec{v}=\vec{w}M} -\log p(\vec{w}) \tag{3.5}$$

Because there is only one $\vec{v}$ such that $\vec{v} = \vec{w}M$, we can write K as

$$\begin{aligned} K &= \sum_{M} p(M) \cdot \sum_{\vec{w}} p(\vec{w})(-\log p(\vec{w})) \\ &= -\sum_{\vec{w}} p(\vec{w}) \log p(\vec{w}) \\ &= H(W). \end{aligned} \tag{3.6}$$

So we have $H(W|V, Mt) = H(W) + J$. The quantity $\mathcal{L} = -J$ represents the amount of leaked information, that is the amount of information that the adversary, a, can learn about p's compact profile. Equation (3.4) provides a first expression for this amount of information. In the following, we refine this expression and present a way to compute an upper bound for it.

3.6.2 Leaked Information and the Landmark Matrix

We now identify a relationship between the amount of leaked information and the number of non-zero rows in the landmark matrix, M. We start by taking a closer look at the term $p(\vec{v}|M)$ from Equation (3.4). We expand it as follows.

$$\begin{aligned} p(\vec{v}|M) &= \sum_{\vec{w} \text{ s.t. } p(\vec{v},\vec{w}|M)\neq 0} p(\vec{v},\vec{w}|M) \\ &= \sum_{\vec{w} \text{ s.t. } p(\vec{v},\vec{w}|M)\neq 0} p(\vec{v}|\vec{w},M)p(\vec{w}) \\ &= \sum_{\vec{w} \text{ s.t. } p(\vec{v},\vec{w}|M)\neq 0} p(\vec{w}) \\ &= \sum_{\vec{w} \text{ s.t. } \vec{v}=\vec{w}M} p(\vec{w}). \end{aligned} \tag{3.7}$$

The first line follows from the law of total probability while the third and fourth result from the same observations on $p(\vec{v}|\vec{w},M)$ as in Equation (3.2).

To solve the final sum in Equation (3.7), we define $S(\vec{v},M) = \{\vec{c}|\vec{v} = \frac{\vec{c}}{||\vec{c}||}M, \vec{c} \in \mathbb{Z}_2^n\}$ as the set of all compact profiles that have the same adjusted similarity vectors given a set of landmarks. To evaluate the cardinality of $S(\vec{v},M)$, we observe that $\forall \vec{c} \in S(\vec{v},M)$, $||\vec{c}||$ belongs to one of the values $0, \sqrt{1}, \ldots, \sqrt{n}$. The worst case w.r.t. information leakage occurs when all vectors in $S(\vec{v},M)$ have the same norm $\sqrt{wt}$, wt being the hamming weight of one such vector. Obviously, $\vec{v} \times \sqrt{wt}$ produces an integer vector. Moreover, $\vec{v} \times \sqrt{wt}$ must be a sum of some of the non-zero rows of M, or in other words, a linear combination of the non-zero rows of M. Then an even worse case occurs when not only all vectors have one same norm, but also only one such linear combination exists: in this case, $S(\vec{v},M)$ is smallest.

Let $k(\vec{v},M)$ be the number of 1's in the coefficients of such a unique linear combination, and let $j(\vec{v},M) = wt - k(\vec{v},M)$ be the number of remaining 1's, those that correspond to zero rows of M. We can then compute the size of $S(\vec{v},M)$ as $\binom{n-D(M)}{j(\vec{v},M)}$ where $D(M)$ is the number of non-zero rows of M. In the general case, we will therefore have the following lower bound on $|S(\vec{v},M)|$.

$$|S(\vec{v},M)| \geq \binom{n - D(M)}{j(\vec{w},M)} \tag{3.8}$$

where with a slight abuse of notation we write $j(\vec{w},M)$ to mean $j(\vec{w}M,M)$. Then, because we assume that all compact profiles are equally likely ($p(\vec{w}) = \frac{1}{2^n}$), we can simplify Equation (3.7) into Inequality (3.9).

$$p(\vec{v}|M) = p(\vec{w})|S(\vec{v},M)| \geq \frac{1}{2^n}\binom{n - D(M)}{j(\vec{w},M)} \tag{3.9}$$

This allows us to compute an upper bound on the amount of leaked information.

$$\begin{aligned}\mathcal{L} &\leq -\sum_{M} p(M) \sum_{\vec{w}} \frac{1}{2^n} \log \frac{1}{2^n} \binom{n - D(M)}{j(\vec{w}, M)} \\ &\leq \sum_{M} p(M)(n - \frac{1}{2^n} \sum_{\vec{w}} \log \binom{n - D(M)}{j(\vec{w}, M)}) \\ &= n - \frac{1}{2^n} \sum_{M} p(M) \sum_{\vec{w}} \log \binom{n - D(M)}{j(\vec{w}, M)})\end{aligned} \tag{3.10}$$

Let us define $S(D(M)) = \sum_{\vec{w}} \log \binom{n-D(M)}{j(\vec{w},M)})$. $S(D(M))$ sums over all possible $\vec{w}$ and thus depends only on M. Since $0 \leq k(\vec{w}, M) \leq D(M)$, $S(D(M))$ is lower-bounded by $T(D(M))$:

$$\begin{aligned}T(D(M)) = &\sum_{wt=D(M)+1}^{n-1} \binom{n}{wt} \\ &\log(\min(\binom{n - D(M)}{wt - D(M)}, \binom{n - D(M)}{wt}))\end{aligned} \tag{3.11}$$

To further simplify $\mathcal{L}$, let $d \in [0, nm]$ be the number of 1's in the matrix M, and let $N(d, D(M))$ be the number of M matrices with d 1's spread across $D(M)$ non-zero rows. Finally, let $p(M_d)$ be the probability of a matrix with d 1's. Then Inequality (3.12) decomposes the summation in the last line of Inequality (3.10) as follows. The outer sum considers all the matrices with i non-zero rows. The inner sum considers all the matrices with d 1's (at least i and no more than $i\,m$, m being the number of columns).

$$\begin{aligned}\sum_{M} p(M)T(D(M)) &\geq \sum_{i=1}^{n} \sum_{d=i}^{i\,m} N(d, i)p(M_d)T(i) \\ &= \sum_{i=1}^{n} T(i) \sum_{d=i}^{i\,m} N(d, i)p(M_d) \\ &= \sum_{i=1}^{n} T(i)p(D(M) = i) \\ \mathcal{L} &\leq n - \frac{1}{2^n} \sum_{D(M)} p(D(M))T(D(M))\end{aligned} \tag{3.12}$$

The last two lines follow because $\sum_{d=i}^{i\,m} N(d, i)p(M_d) = p(D(M) = i)$ is the probability of having a matrix with i non-zero rows.

3.6.3 Upper Bound on Information Leakage

We conclude our analysis by computing the value of the bound on information leakage in the test configurations of Section 3.4. To this end, let η be the probability

Table 3.2 – F1 score and upper bounds on leaked information in the configurations of Section 3.4.

	n	$\mathcal{L}$	F1 score
ML-100k, $m = 25$	660	660	0.6690
ML-100k, $m = 10$	660	505	0.6602
ML-100k, $m = 7$	660	399	0.6567
ML-100k, $m = 5$	660	338	0.6480
ML-100k, $m = 3$	660	283	0.6360

of an element in the M matrix's being 1, and let $\vec{r}$ be a row vector in matrix M. We can compute the probability of having a non-zero row vector in M as follows.

$$\rho_1 = p(\vec{r} \neq \vec{0}) = 1 - (1 - \eta)^m \tag{3.13}$$

This allows us to rewrite Equation (3.12) to obtain:

$$\mathcal{L} \leq n - \frac{1}{2^n} \sum_{D(M)} \binom{n}{D(M)} (\rho_1)^{D(M)} (1 - \rho_1)^{n - D(M)} T(D(M)). \tag{3.14}$$

For the configuration of Section 3.4, we obtain $\eta = 0.05$. Depending on the dataset and number of landmarks, this leads us to the values of F1 score and leaked-information bound shown in Table 3.2. The results show that 5, and 3 landmarks allow *H&S* to provide good similarity scores while leaking respectively no more than 51% and 43% of the information in the compact profiles. Also, while the value of $\mathcal{L}$ for $m = 25$ may seem bad, it does not mean that 25 landmarks leak all the information in the compact profiles, but only that the bound is not tight.

3.7 Conclusion

We have presented *Hide & Share* (*H&S* for short), a novel peer-to-peer similarity computation mechanism for decentralized KNN computation. We have demonstrated both formally and experimentally that *H&S* protects the privacy of users in the context of a peer-to-peer recommendation system. This protection is provided while preserving the system's effectiveness. *H&S* introduces *landmarks*, random data structures that are shared between peers comparing their profiles. It leverages a combination of cryptography and security mechanisms to ensure that these landmarks cannot be diverted to attack the privacy of users.

We have shown using three real-world datasets that *H&S* maintains a strong level of privacy while providing recommendations close to that of an open system with no particular privacy protection mechanism. We have also shown using preliminary attacks that the *Hide & Share* mechanism performs better than a randomization scheme. Finally we have proposed an upper bound on the amount of information leaked by our scheme.

Although recommendation has been our primary focus in this thesis, the applicability of *H&S* is not limited to recommendation services. In the future, we

would like to investigate how our proposal might be applicable to other services, such as search, news propagation, and decentralized differential privacy. We also plan to improve our upper bound, and further investigate the properties of *H&S* under stronger attack models than those presented here, in particular if we assume that attackers have some (limited) collusion ability, or have some prior knowledge of items and users distribution in the system.

Chapter 4

Sybil Attack on User-based Collaborative Filtering Systems

Recommendation systems collect information about our own preferences, compare them to those of other users, and provide us with suggestions on a variety of topics. But is the information gathered by a recommendation system safe from potential attackers, be them other users, or companies that access the recommendation system? And above all, can service providers protect this information while still providing effective recommendations?

In this chapter, we analyze the effect of Sybil attacks on collaborative-filtering systems, and discuss the impact of different similarity metrics in the trade-off between recommendation quality and privacy. Our results, on a state-of-the-art recommendation framework and on real datasets show that existing similarity metrics exhibit a wide range of behaviors in the presence of Sybil attacks. Yet, they are all subject to the same trade off: Sybil resilience for recommendation quality. We therefore propose and evaluate $2-step$, a novel similarity metric that combines the best of both worlds: a low $RMSE$ score with a prediction accuracy for Sybil users of only a few percent.

The remainder of this chapter is structured as follows. We discuss the motivation of this work in Section 4.1, followed by our system and attack models in Section 4.2, and our experimental protocol in Section 4.3. We then present our first contribution in Section 4.4: a detailed experimental analysis that highlights the impact of similarity metrics on attack resilience. Section 4.5 presents and evaluates our second contribution: the novel $2-step$ similarity metric that combines high recommendation quality with resilience to attacks. Section 4.6 concludes the chapter and presents our future directions.

4.1 Motivation

In Chapter 3, we discussed possible attacks on user privacy which affect CF systems depending on their architecture: attacks by “Big Brother" adversaries for centralized systems, and attacks by “Little Brothers" adversaries for decentralized ones. But there also is a class of attacks on user privacy which is independent of systems architectures. We address in this chapter one type of attack from this class: Sybil attacks

using auxiliary information. We study this type of attack because it is difficult to avoid since it exploits the very principle of CF, and because it was previously not well studied in the literature. Before describing this kind of Sybil attack, we recall two other attacks, which we already described in previous chapters, from the same class. Although they are beyond the scope of this thesis, they are worth considering so that the reader may have a broader view of this class of architecture-independent attacks.

We first recall the passive attack Calandrino *et al.* propose in [21], previously described in Section 2.2. It affects the type of item-based CF systems publicly showing related-items lists, *i.e.* releasing for each item, a list of the most relevant related items. The goal of the adversary is to discover changes in the target user's profile she watches such as the addition of a new item.

The rationale of this attack is that if an item i enters or move upwards in several related-items lists of items known to be liked by the target user, this indicates that the target rated i with high probability. The set of watched related-items lists forms a quasi-identifier of the target user. This attack assumes the adversary knows a subset of the target's profile, which the authors of [21] argue can reasonably be found by auxiliary channels.

This attack is particularly insidious because it does not requires the adversary to directly interact with the recommendation system. Given that the only output of the RS used is related-items lists, this attack is obviously architecture-independent.

We also recall the identification-by-trap-item attack, previously described in Section 1.3. The adversary can be a user or a content-provider, depending on the system's policy regarding who can add new items. Her goal is to find some identifier of users (*e.g.* an IP address) who are interested in some topic (*e.g.* environmental activism).

The adversary can do so by adding to the system an item designed to be relevant to the target interest community, and by bundling with it a trap mechanism leaking identifying information. For instance, it could be a surreptitiously unique URL pointing to an adversary-controlled website within the item's description. Thus, if the target user clicks on this URL, the adversary can find the target's IP address by looking the unique URL up in the website's access log.

Finally, we describe the Sybil attack which we address in this chapter, *i.e.* the same Sybil attack as in Section 2.2. [21] introduced this attack but did not study it, so the first contribution of this chapter consists in evaluating this attack with a variety of CF configurations.

We recall that Sybil attacks consist in a single entity controlling several fake identities, each appearing as a different user to the attacked system. In this particular attack affecting user-based collaborative-filtering systems, the goal of the adversary is to trick the recommendation algorithm into revealing items from the profile of a target user. This attack requires the adversary to have some a priori information about the target user's interests, *i.e.* some items, and their ratings possibly, from the target's profile, found using auxiliary channels. In the context of CF systems, the adversary may use auxiliary channels such as e-commerce reviews, or consumer protection organizations' forums. Consequently, information obtained this way is called auxiliary information.

Once this prerequisite information has been acquired, the adversary creates k Sybil users, giving them the auxiliary information as their profile. Because user-based CF chooses for a user the k most similar other users, each Sybil should have a KNN composed of the $k-1$ other Sybils and the target, with high probability. Indeed, Sybil users have the maximum similarity because they share the same profile, and they should have a quite high similarity with the target, assuming the adversary's auxiliary information is distinctive enough to constitute a quasi-identifier of the target. The composition of Sybils' neighborhoods is the main success criterion for the attack. If this criterion is met, then, because user-based CF takes recommendations from neighbors' profile, any new item recommended to a Sybil must come from the target's profile. Because such an attack exploits the rationale of CF itself, it is difficult to prevent.

In this chapter, we start by studying the aforementioned Sybil attack since the authors of [21] did not evaluate it. Using a state-of-the-art recommendation framework and three real datasets, we show that its performance strongly depends both on the dataset, and thus on the specific user population, and on the similarity metric employed by the system. We also observe that the ability to prevent Sybil nodes from isolating their target (*i.e.* meeting the aforementioned criterion) is key in mitigating the attack.

This observation allows us to propose a new metric that combines the benefits of standard *Cosine* similarity in terms of recommendation quality, with resilience to this Sybil attack. Our new metric, $2-step$, combines a good $RMSE$ score with a very low prediction accuracy for Sybil attackers.

4.2 Problem Statement

We consider a recommendation system using a user-based collaborative filtering technique. As mentioned in Chapter 2, CF algorithms are particularly successful due to their content-agnostic nature. In the user-based variant, a CF system essentially consists of a K-Nearest-Neighbors (KNN) algorithm that identifies for each user, the set of most similar other users according to the opinions they expressed on the items they have been exposed to. In the following, we consider a system in which users rate items with a numerical score (e.g. 1 to 5). For each user, U, the system collects the mapping between items and scores in a user-profile data structure. It then runs a KNN algorithm to identify the users associated with the most similar user profiles. After identifying U's neighbors, the system ranks the items they have rated—for example through a combination of ratings, number of neighbors that rated them, and similarity of U with those neighbors—and recommends to U the top-ranking ones to which she has not yet been exposed.

We consider an adversary that has the capability to (i) observe part of the ratings expressed by the target—we refer to these ratings as auxiliary information—and (ii) create a number of fake identities (Sybils) with the objective to extract information from the recommendation system [21]. We also assume that the adversary knows the value of k, the size of neighborhoods in the KNN algorithm, which enables her to create the right number of Sybils.

Adversaries can obtain auxiliary information in several ways depending on the characteristics of the target recommendation system. In a system like Amazon.com, the adversary can use reviews posted by the target as evidence that she bought this or that item, or posts on social networks such as "I just bought item X". In systems like Last.fm, the adversary may consult publicly available listening histories. In decentralized systems like [16], she may simply exploit the profile exchanges at the basis of the recommendation algorithm. Finally, targeted advertisement boxes (Google-ad like boxes on web pages), may enable malicious websites to track users to identify the advertisements they click, thereby gathering auxiliary information.

How to create fake identities also depends on the system being attacked. On some websites, it is enough to create a number of accounts and perform some actions such as listening to music tracks, or adding reviews. On others, it may be more complex as creating a profile might involve purchasing items or other costly actions. For the purpose of this chapter, we assume that the adversary has the means to give each of its fake identities a profile consisting of the set of auxiliary items and associated ratings. In case creating a profile involves costly actions, the adversary may simply shrink the set of auxiliary items to a manageable number. If the set of ratings copied by the Sybils is large enough, the KNN algorithm should give each Sybil a neighborhood consisting of the target user and the remaining Sybils. The adversary can then monitor all the recommendations received by each of the Sybils and assume that all the recommended items come from the profile of the target.

Although [21] introduced this attack, it did not evaluate its effectiveness. Its authors only suggest that attacks based on auxiliary information should be effective as long as the Sybils have access to $O(log(N))$ items, N being the number of users in the system. In the following, we show that the effectiveness of the attacks closely depends on the considered similarity metric used in the collaborative-filtering system, as well as on the sparsity of the dataset, which in turn depends on the characteristics of the recommendation system's population.

4.3 Experimental Protocol

We implemented the Sybil attack described above on top of Mahout [1], a popular machine-learning framework developed by the Apache Foundation. Using an existing framework allows us to concentrate on the implementation of the attacks, while using a well-tested implementation of state-of-the-art user-based collaborative filtering. In order to effectively implement the attacks, we implemented slight modifications in Mahout that make it possible to model the behavior of Sybil users. Our modified Mahout implementation, as well as the code, the scripts, and the datasets for running our experiments are available at `https://gforge.inria.fr/projects/recopriv`.

In all our experiments, we consider a user-based collaborative-filtering system running with several different similarity metrics. We ran experiments on real datasets from MovieLens, Jester, and MovieTweetings, and we evaluated the efficiency of the attack depending on the considered similarity metrics as well as the trade-off between privacy and quality of recommendation.

4.3.1 Similarity Metrics

We implemented the Sybil attack described above on user-based collaborative filtering using a KNN algorithm with different similarity metrics. To compute this KNN graph, we consider seven metrics: the well-known *Cosine* similarity in three variants, two variants of the metric from [16], the *Jaccard* index, and the *Pearson* correlation coefficient [110]. The classic metrics have already been presented in Section 2.1.2.1. However, we quickly recall how they operate before detailing the other metrics. *Cosine* similarity reflects the similarity between two profiles by measuring the cosine of the angle between them. The *Jaccard* index, in turn, is defined as the size of the intersection divided by the size of the union of two profiles, regardless the score associated to items. Finally, the *Pearson* correlation consists of the covariance of the two profiles divided by the product of their standard deviations.

Several variants of the cosine similarity have been proposed. In our evaluation, we also considered two slightly modified versions of the cosine similarity, called *Cos-overlap* and *CosineAvg* in the following. The former computes the norms of the two profiles by counting only the items that are common to both of them while the latter uses the average rating of user in the scalar product when an item is not rated by both users.

$$\textit{Cos-overlap}(u,n) = \frac{u \cdot n}{\sqrt{\sum_{i \in I_{un}} (u_i)^2} \times \sqrt{\sum_{i \in I_{un}} (n_i)^2}}$$

$$\textit{CosineAvg}(u,n) = \frac{\sum_{i \in I_u \cup I_n} u_i \times n_i}{\|u\|\|n\|}$$

where u_i (and n_i) is equal to $\bar{u}$ (respectively $\bar{n}$) if $i \nexists \in I_u$ (respectively $i \nexists \in I_n$).

The similarity metrics presented above are symmetric, meaning that the similarity of u with n is the same as the similarity of n with u. We therefore complement them by also considering the asymmetric metric proposed in [16] which favors neighbors that have more restrictive tastes by considering only items that are rated by both users, for the scalar product and for the norm of one of the two profiles depending on the direction of the metric. We consider two variants of this metric. Given a user u and a potential neighbor n, the first variant, *WUP-u*, considers only the items in u that also appear in n, and all the items in n. The second variant, *WUP-n*, considers all the items in u and only the items in n that also appear in u.

$$\textit{WUP-u}(u,n) = \frac{\sum_{i \in I_{un}} u_i \times n_i}{\sqrt{\sum_{i \in I_{un}} (u_i)^2} \times \sqrt{\sum_{i \in I_n} (n_i)^2}}$$

$$\textit{WUP-n}(u,n) = \frac{\sum_{i \in I_{un}} u_i \times n_i}{\sqrt{\sum_{i \in I_u} (u_i)^2} \times \sqrt{\sum_{i \in I_{un}} (n_i)^2}}$$

Table 4.1 – Characteristics of the datasets in terms of number of users, number of items, number of ratings and rating type.

	# users	# items	# ratings	rating type
ML-1	943	1,682	100,000	$[1:5]$
Jester-1-1	24,983	100	1,810,455	$[-10.0:10.0]$
MovieTweetings	24,921	15,142	212,835	$[0:10]$

4.3.2 Dataset

We ran our experiments on three datasets, of which two are traces from real recommendation systems while the third one was used for the RecSys challenge 2014.

Table 4.1 outlines the characteristics of the three datasets we use. ML-100k[1] (called ML-1 in the following) is a trace from the MovieLens [52] online movie-recommendation service. Jester-1-1[2] is a trace from the Jester [46] online joke-recommendation service. It is the first third of Jester's dataset-1. MovieTweetings[3] is a collection of movie ratings from IMDb users expressed as well-structured tweets. This dataset was used for the RecSys challenge in 2014. We use the three parts of the dataset as one, and make our own splits where needed. We remove 23 erroneous ratings as they do not respect the expected 10-star rating format. Consequently, this also removes 3 users having only such erroneous ratings.

ML-1 and Jester each contain information on subsets of users who rated at least 20 and 36 items respectively, which makes them less sparse than most of real-world datasets. To investigate the impact of sparsity, we derive a number of variants from ML-1 with different levels of sparsity. Starting from ML-1, we progressively remove randomly chosen ratings and obtain four additional datasets (numbered ML-2 to ML-5). Table 4.2 details the number of ratings and the sparsity, expressed as the percentage of missing ratings in the user-item matrix, for each variant. Note that because of the definition of sparsity, a value of 94% already represents a pretty dense matrix, but values above 99% correspond to a sparse one.

Dataset	Ratings	Sparsity
ML-1	100,000	93.69%
ML-2	50,351	96.82%
ML-3	25,180	98.41%
ML-4	13,698	99.14%
ML-5	7,621	99.52%
Jester-1-1	1,810,455	27.53%
MovieTweetings	212,835	99.94%

Table 4.2 – Sparsity of the 3 original datasets and of the ML variants.

We assume that the auxiliary information available to the attacker consists not only of a list of items, but also of the associated ratings as expressed by the target.

[1]MovieLens dataset available at: `http://grouplens.org/datasets/movielens/`

[2]Jester dataset available at: `http://eigentaste.berkeley.edu/dataset/`

[3]MovieTweetings dataset available at: `http://2014.recsyschallenge.com/dataset/`

We consider several percentages of auxiliary items available to the attacker. As discussed in Section 4.2, these items correspond to the information generally available in decentralized recommendation systems [16], but may also be obtained in a centralized setting if the associated website publishes user ratings. We observe that this is a fairly strong assumption on the attacker as in many cases, available information is much less precise. For example, social network plugins may publish updates such as "Tom just saw Highlander", but without specifying a rating.

Even though the auxiliary knowledge comprises both items and ratings, we define the attack's goal as the need to determine whether the target profile contains a particular item, with a high-enough rating (≥ 3).

4.3.3 Metrics

We evaluate the attack according to three metrics. The first one measures the ability of Sybils to construct the ideal neighborhood to carry out the attack. As discussed, this consists of the target user plus $k-1$ Sybils. Specifically, we measure the fraction of Sybils that obtain such an ideal neighborhood.

The other two metrics measures instead the success of the attack itself and consists of *yield* and *accuracy*. Yield simply measures the number of guesses that the attacker can make thanks to the action of the Sybils. Accuracy measures instead the fraction of correct guesses.

Finally, we measure the quality of recommendations in terms of the Root Mean Square Error ($RMSE$). To this end, we split the dataset into a training and a testing set: the former containing 90% of the ratings, and the latter containing the remaining 10%. We build the KNN of each user using the training set and we issue recommendations to try to predict the ratings of in the testing set. The top-N list computed and recommended to users contains only items that are in the testing set [34]. $RMSE$ measures how close recommendations are to the actual ratings of users in the testing set. It is defined in Chapter 2.1.2.1: We use Mahout's default rating predictor function, which is similar to the basic predictor described in Section 2.1.2.4.

4.4 Results

User-based collaborative filtering relies on a KNN process to identify, for each user, the set of most similar other users in term of interests. The Sybil attack described in this chapter relies on the adversary's ability to control users that use partial information about the target to approach it within the KNN graph structure. More precisely, the attack aims at populating the neighborhoods of each of the Sybil identities with the target user and exactly $k-1$ other Sybils. Although this may seem straightforward, the ability to do so strongly depends on the similarity metric and the distance of the target with other real users.

For instance, a measure which drastically segregates users even with few changes in their profiles leaves enough room around the target for the adversary to place its Sybil users. In contrast, a similarity measure that does not differentiate among a large set of users regardless of few changes in their profiles will tend to provide

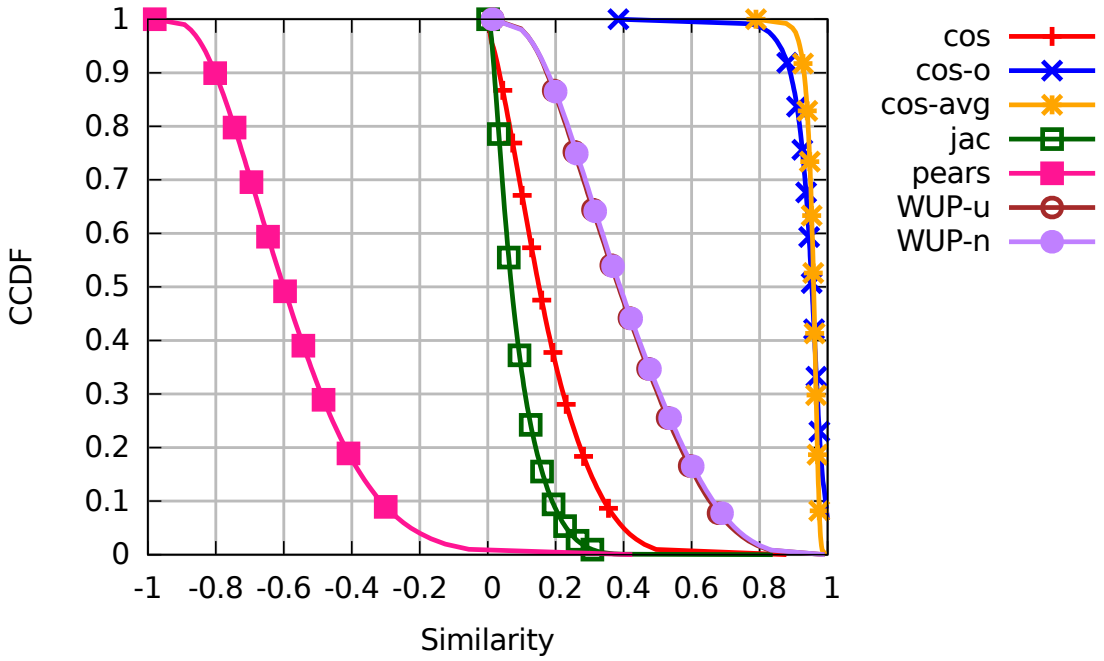

Figure 4.1 – Distribution of similarity between couples of users with various similarity metrics on ML-1.

similar scores to a number of potential neighbors in the KNN structure. This will make it more difficult for the adversary to create Sybil users that have only the target and other Sybils as potential neighbors.

To illustrate these differences among metrics, Figures 4.1,4.2, and 4.3 depicts the Complementary Cumulative Distribution Function (CCDF) of the similarity between all pairs of users for different similarity metrics in each of the three considered datasets. Results show that the behaviors of the similarity measures can vary considerably depending on the dataset. ML-1 and MovieTweetings exhibit significant differences between similarities, while in Jester all similarities except *Jaccard* exhibit similar behaviors. The information in Table4.2, suggests a reason for this different behavior across datasets: Jester is a particularly dense dataset with a sparsity value of only 27.53% while both ML-1 and MovieTweetings are particularly sparse with sparsity values above 90%.

4.4.1 Similarity Metrics vs Sybil Attacks

Given the differences among similarities, we start our analysis by examining the effectiveness of the Sybil attack on a recommendation system based on each of the seven similarity metrics described in Section 4.3.3. Our results, depicted in Figures 4.4 through 4.12, highlight the high variability in the effectiveness both with respect to the different metrics and with respect to the datasets. In all this experiments, we select randomly from the target user's profile the items given as auxiliary information to the Sybil users.

Figures 4.4, 4.5, and 4.6 show the percentage of Sybils that obtain their desired

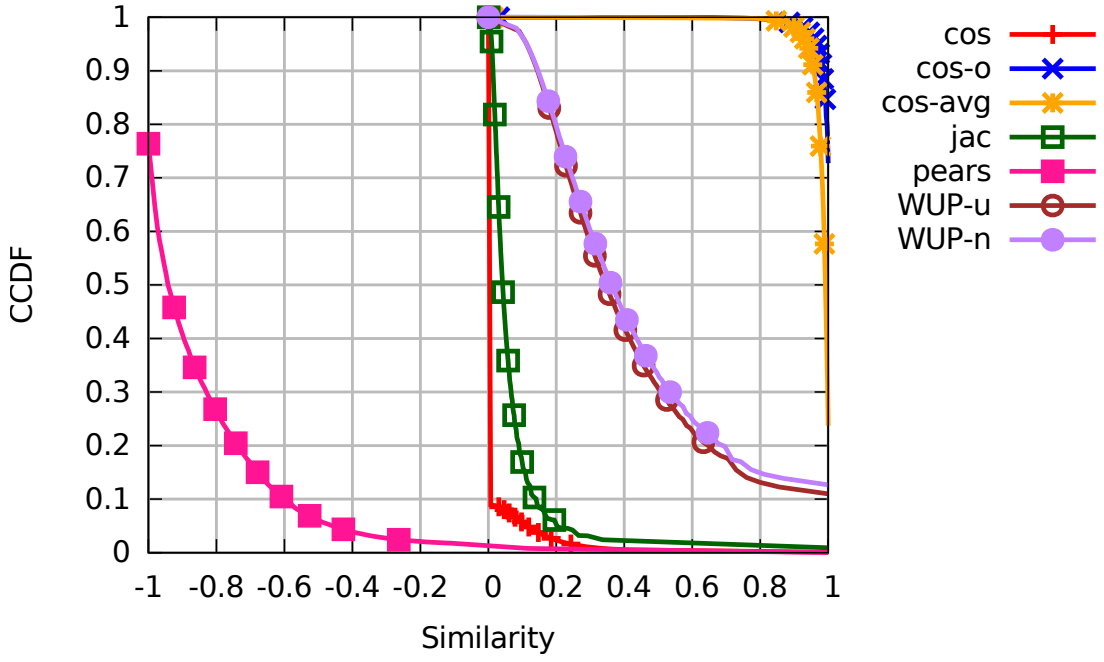

Figure 4.2 – Distribution of similarity between couples of users with various similarity metrics on MovieTweetings.

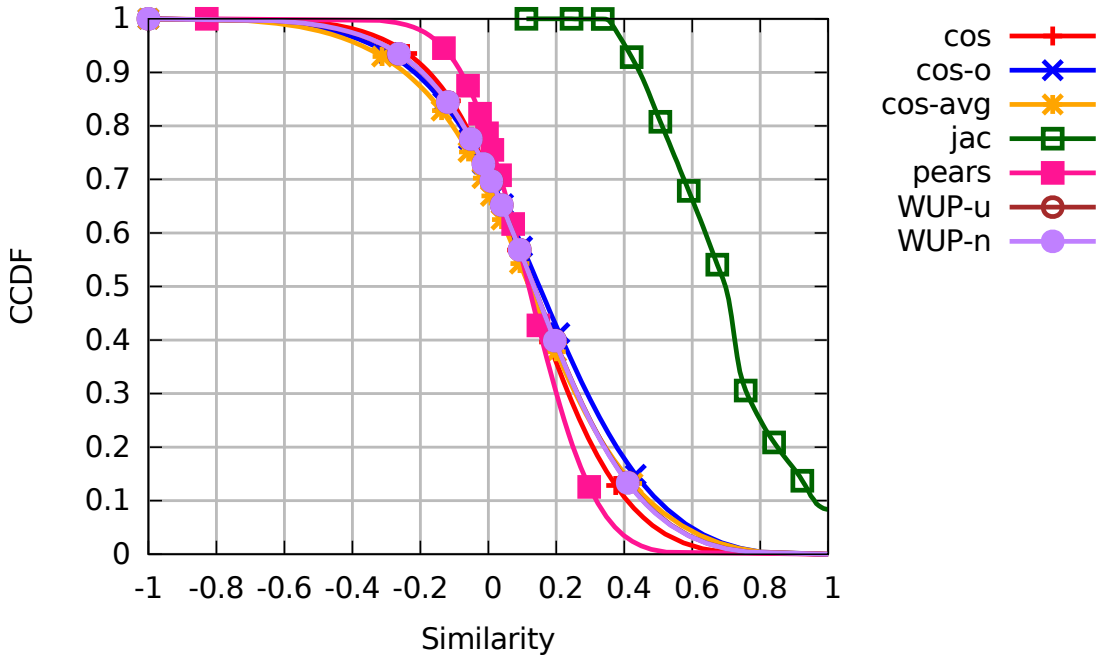

Figure 4.3 – Distribution of similarity between couples of users with various similarity metrics on Jester.

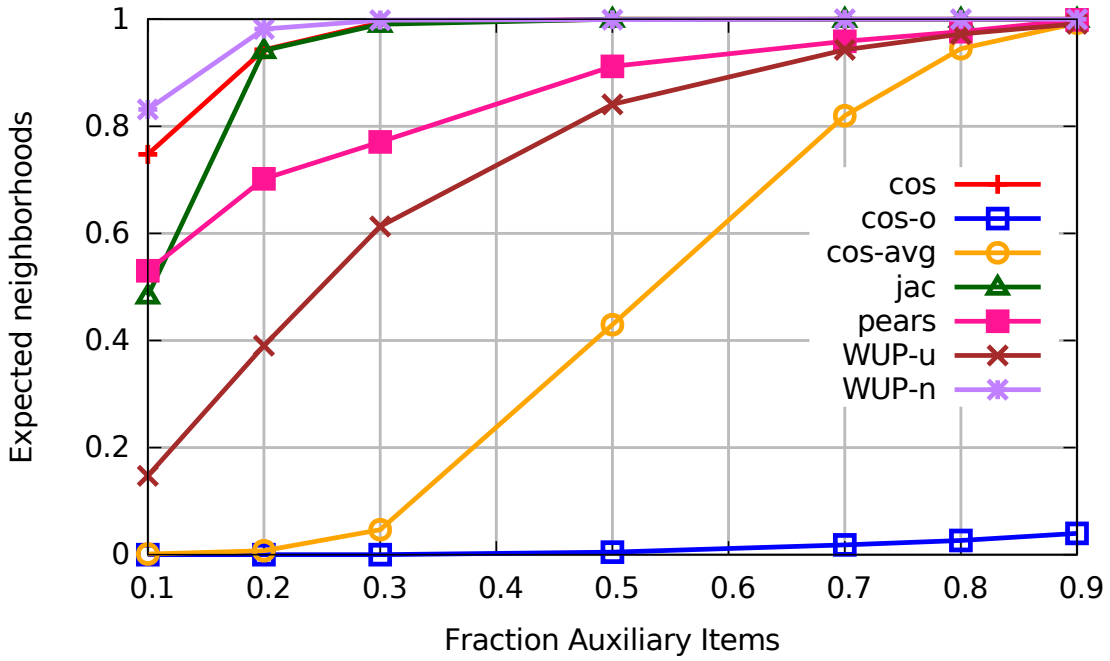

Figure 4.4 – Fraction of Sybils with ideal neighborhoods for an isolated attack on ML-1.

neighborhoods as a function of the fraction of auxiliary items in their possession. Like for Figures 4.1 to 4.3, we observe different behaviors depending on the dataset. In Figure 4.4, we can distinguish three groups of metrics. For the first group, consisting of *Cosine*, *Jaccard*, and *WUP-n*, knowing 20% of the items in the target profile allows each Sybil to obtain a neighborhood that consists exactly of the target and $k-1$ other Sybils. The metrics in the second group, namely *Pearson*, *WUP-u*, and *CosineAvg*, require instead as many as 80% or 90% of the target's items in order to achieve the same result. When knowing 30% of the items, *Pearson* and *WUP-u* only allow respectively 80% and 60% of the Sybils to build their target neighborhoods, while *CosineAvg* only provides the desired neighborhood to about 5% of the Sybils. Finally, the third group consists only of *Cos-overlap*: with this metric, only a few Sybils manage to obtain an ideal neighborhood even when they have as much as 90% of the target items in their profiles.

Figure 4.5 shows similar relative success-rates, even though with lower absolute values. The first group, *Cosine*, *Jaccard*, and *WUP-n* allows Sybils to obtain ideal neighborhood in the highest number of cases. The second group is slightly more resilient to the attack, while, *Cos-overlap* makes it very hard for Sybils to obtain their ideal neighborhoods.

Figure 4.6 offers instead a very different results for the Jester dataset. In this case, the Sybils obtain the best neighborhoods with *WUP-n* and *Cos-overlap*, while *Cosine*, *CosineAvg*, *Pearson*, and *WUP-u* yield lower quality but still good neighborhoods. Finally, *Jaccard* surprisingly yields the worst neighborhoods. Again, the reasons for the surprising behavior of the metrics in the Jester dataset lies in the high density of this dataset.

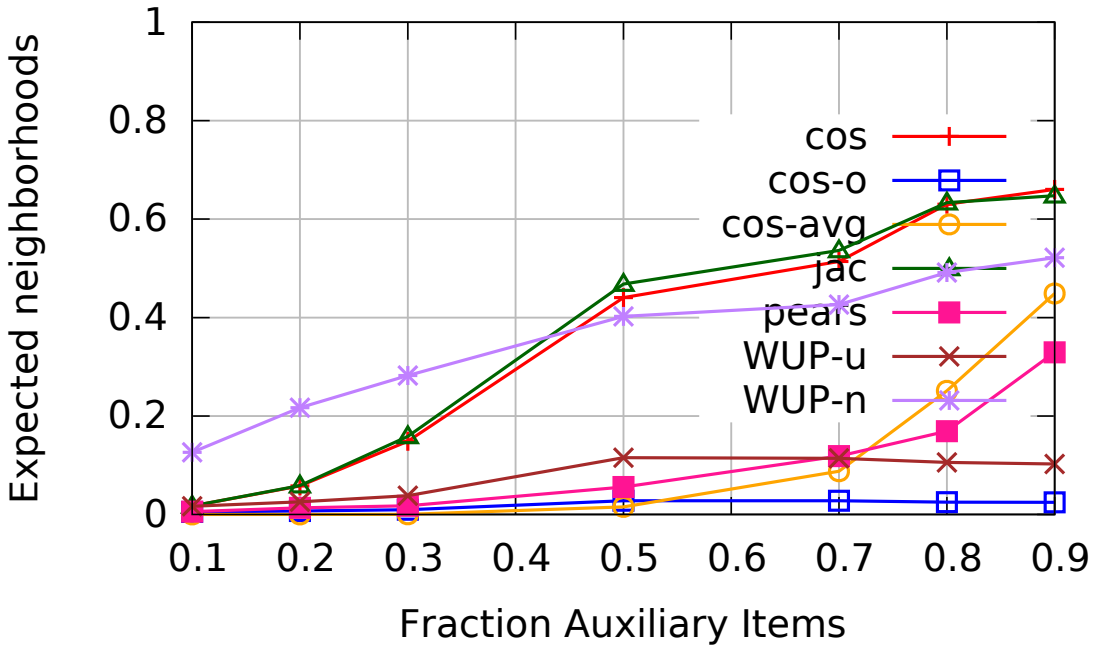

Figure 4.5 – Fraction of Sybils with ideal neighborhoods for an isolated attack on MovieTweetings.

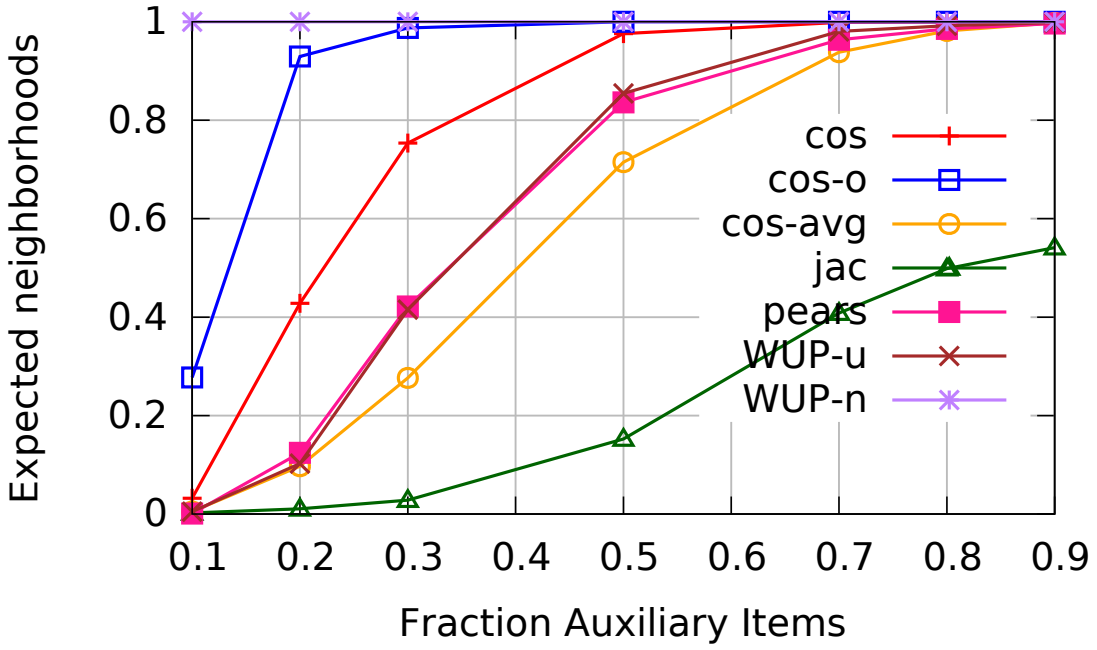

Figure 4.6 – Fraction of Sybils with ideal neighborhoods for an isolated attack on Jester.

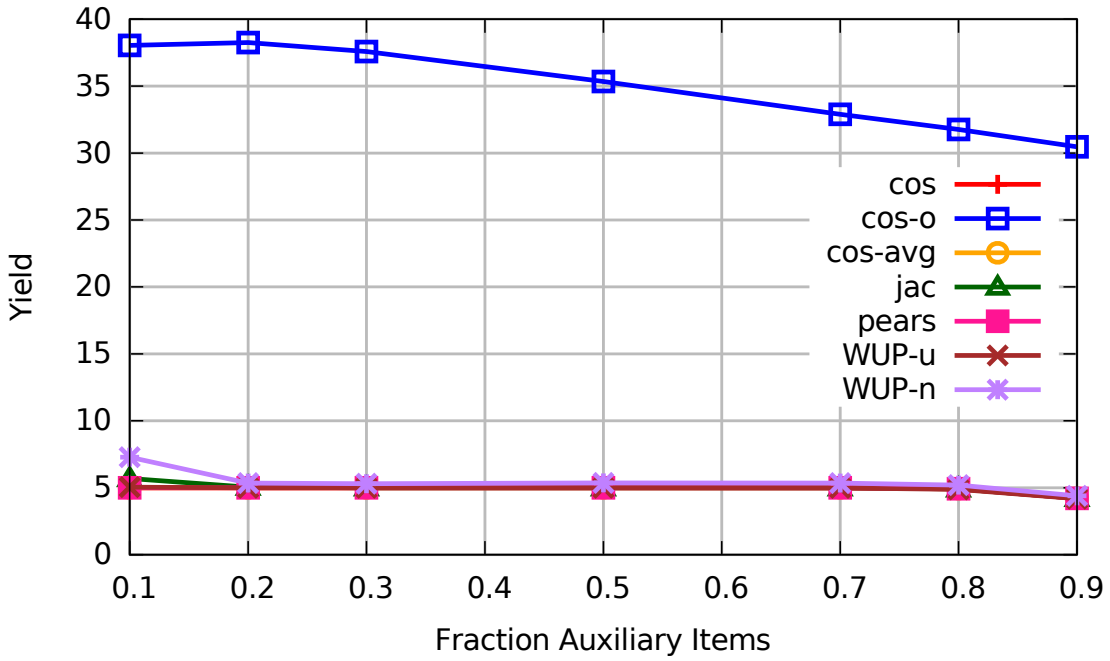

Figure 4.7 – Yield for an isolated attack on ML-1.

Figures 4.7, 4.8, and 4.9 provide a different perspective on these results by depicting the yield obtained by the attack as a function of the fraction of the target's profile available as auxiliary items. Figures 4.10, 4.11, and 4.12 are the equivalent for the attack's accuracy. To measure these, we have each Sybil request 5 recommendations from its neighborhood. This gives a maximum possible yield of 50—5 recommendations for each of the 10 Sybils.

Again, the behaviors vary depending on the datasets. For ML-1 and MovieTweetings (Figures 4.7 and 4.8), only *Cos-overlap* obtains a high recommendation yield. In Figure 4.7 all the remaining metrics achieve a yield of around 5%, while in Figure 4.8 yield values range from 0 to 25%.

A comparison with Figures 4.7 to 4.9 shows that yield negatively correlates with accuracy. With reference to Figure 4.7, consider two Sybils that have reached their ideal neighborhoods. If both ask for 5 recommendations, they will both receive the same 5 items because their only neighbor that can provide recommendations consists of the target user. The high yield of *Cos-overlap* in ML-1 therefore result from the Sybils' inability to obtain a high-quality neighborhood with this metric.

Figures 4.10, 4.11, and 4.12 confirm this reasoning by showing the fraction of correct Sybil guesses. In Figures 4.10 and 4.11, *Cos-overlap* allows Sybil users to obtain good predictions for a very small number of items (only 10% in ML-1 and close to 0 in MovieTweetings). The remaining lines in each plot follow the trend of Figures 4.4 to 4.6 pretty closely. In all three plots, Sybils obtain higher accuracy as the fraction of auxiliary items increase. With very high percentages of auxiliary items, accuracy decreases only because there profiles contain fewer and fewer items to guess.

Finally, we observe that in Figure 4.12, Sybils obtain very high accuracy for

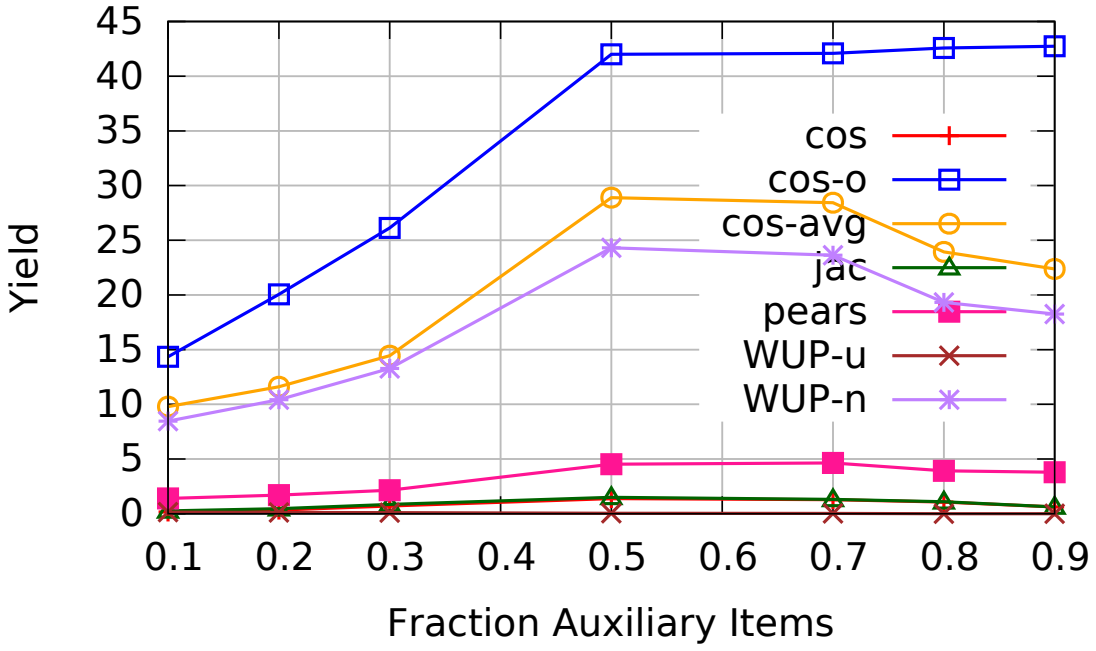

Figure 4.8 – Yield for an isolated attack on MovieTweetings.

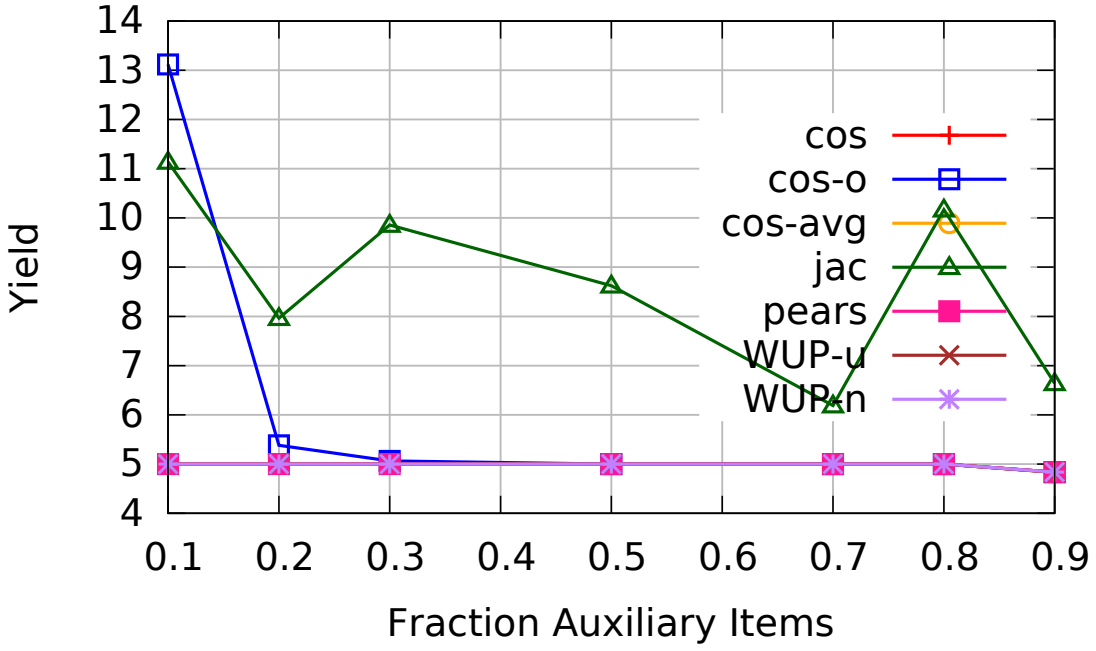

Figure 4.9 – Yield for an isolated attack on Jester.

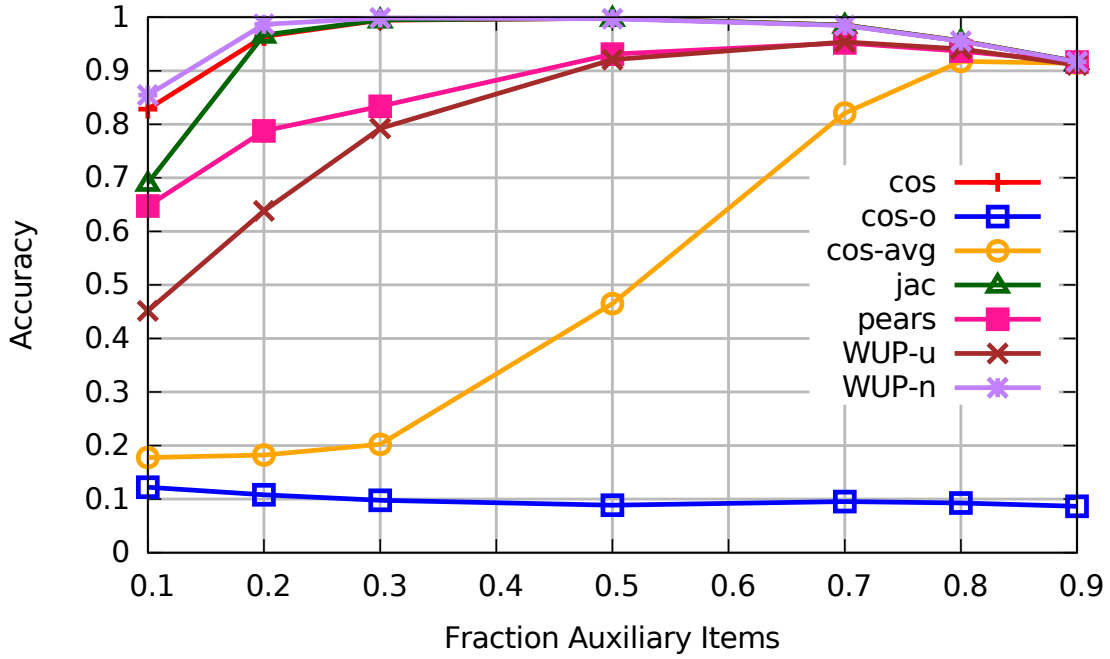

Figure 4.10 – Accuracy for an isolated attack on ML-1.

pretty much all metrics. We suspected this behavior to be due to the density of this dataset. To confirm this hypothesis, we also ran an experiment in which Sybils select random neighbors to perform their predictions. This gave us a baseline accuracy on the Jester dataset of about 44%, a very high value resulting from the high density of this dataset.

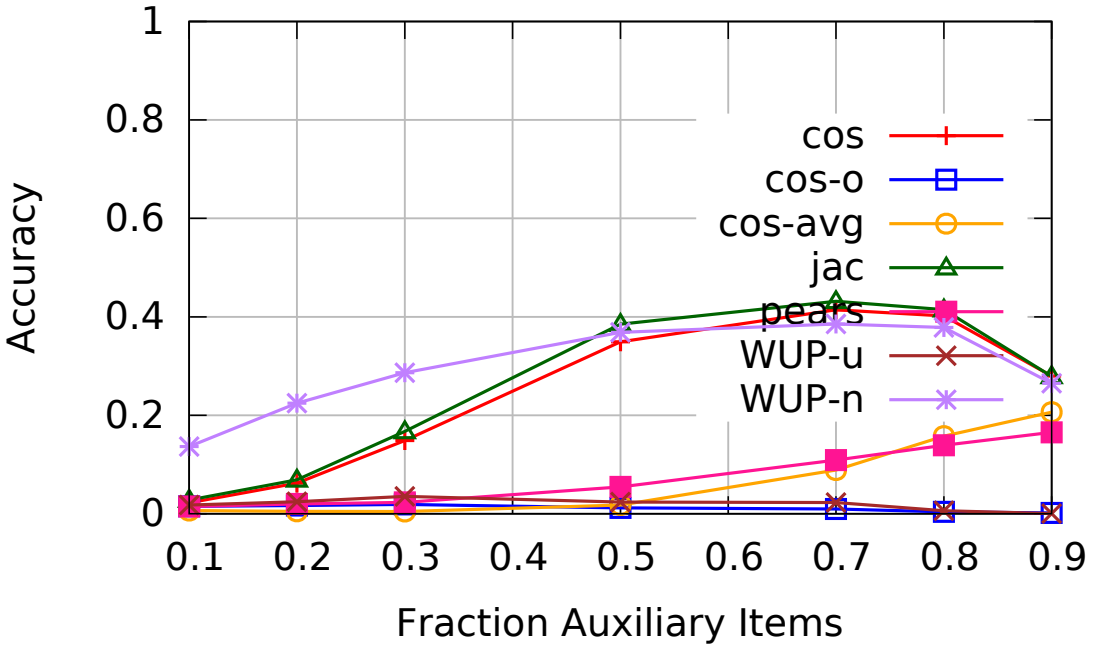

Figure 4.11 – Accuracy for an isolated attack on MovieTweetings.

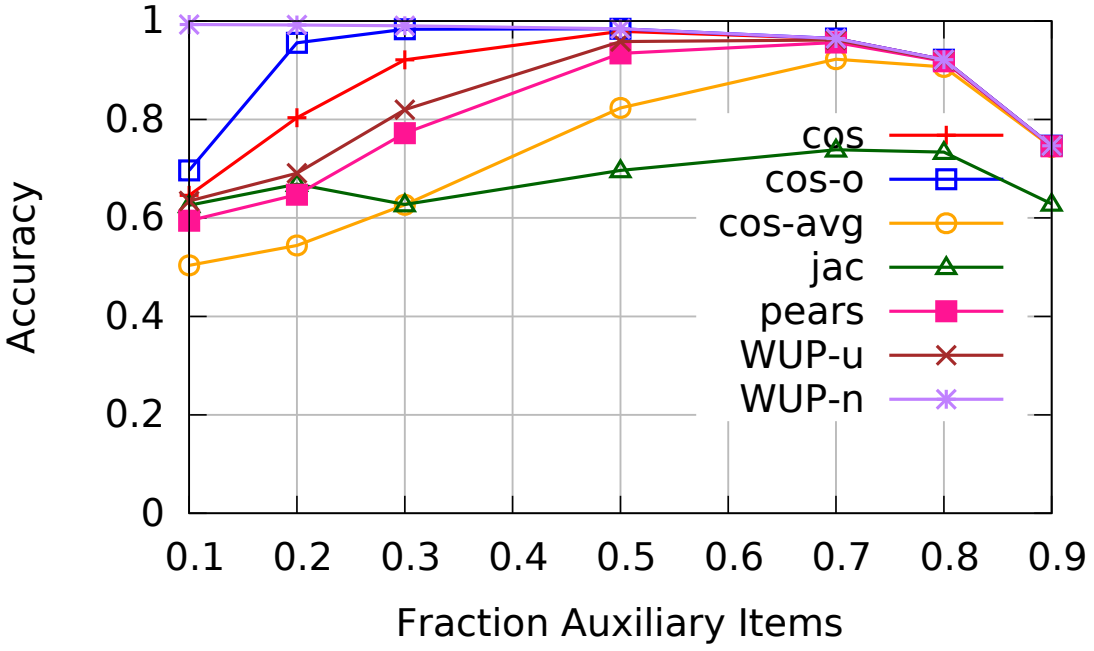

Figure 4.12 – Accuracy for an isolated attack on Jester.

4.4.2 Recommendation Quality

If different metrics yield varying responses to Sybil attacks, the primary purpose of a similarity metric consists in providing accurate recommendations. To get a clearer picture of the trade-off between recommendation quality and resilience to attacks, we now analyze the impact of each of the considered metrics on recommendation performance. Figures 4.13, 4.14, and 4.15 depict the root mean squared error ($RMSE$) obtained with each metric for varying neighborhood sizes in each of the datasets. Lower values mean better recommendations.

The plots also show a significant variability in performance depending on the metric and on the dataset. For ML-1, as in earlier plots, we can identify three groups of similarity metrics characterized by increasing levels of recommendation performance. The first group consists only of *Cos-overlap*. The good resilience to censorship exhibited by this metric in Section 4.4.1 comes at the cost of much poorer recommendation quality. By considering only the items that belong to both profiles being considered, *Cos-overlap* completely ignores the important distinction between users with specific—and thus useful in terms of recommendation—interests, and *hubs* with very unspecific behaviors.

The second group of metrics, consisting of *Pearson*, *WUP-u*, and *CosineAvg*, exhibits significantly better performance with an MAE of about 0.85 and an $RMSE$ of 1.1 with $k = 50$ neighbors. But the best results remain those of the last group of metrics: *Cosine*, *Jaccard*, and *WUP-n*. In this respect, it is interesting to observe the difference between the two metrics inspired by [16]: *WUP-u*, and *WUP-n*, which reflects their asymmetric nature.

In MovieTweetings and Jester, the relative differences between the metrics vary. *Pearson* exhibits particularly poor performance, *Jaccard* performs worse in Jester than in other datasets, while other metrics tend to reflect the relative differences highlighted in Figures 4.10 to 4.12. Indeed, apart from any metric-specific remark, the main information we can extract from Figures 4.13 to 4.15 consist of its correlation with the data in Figures 4.4 through 4.12. For all metrics, high resistance to Sybil attacks results in poorer recommendation quality. This raises the research question of how to design a metric that can combine both benefits. We provide a preliminary answer to this question in Section 4.5.

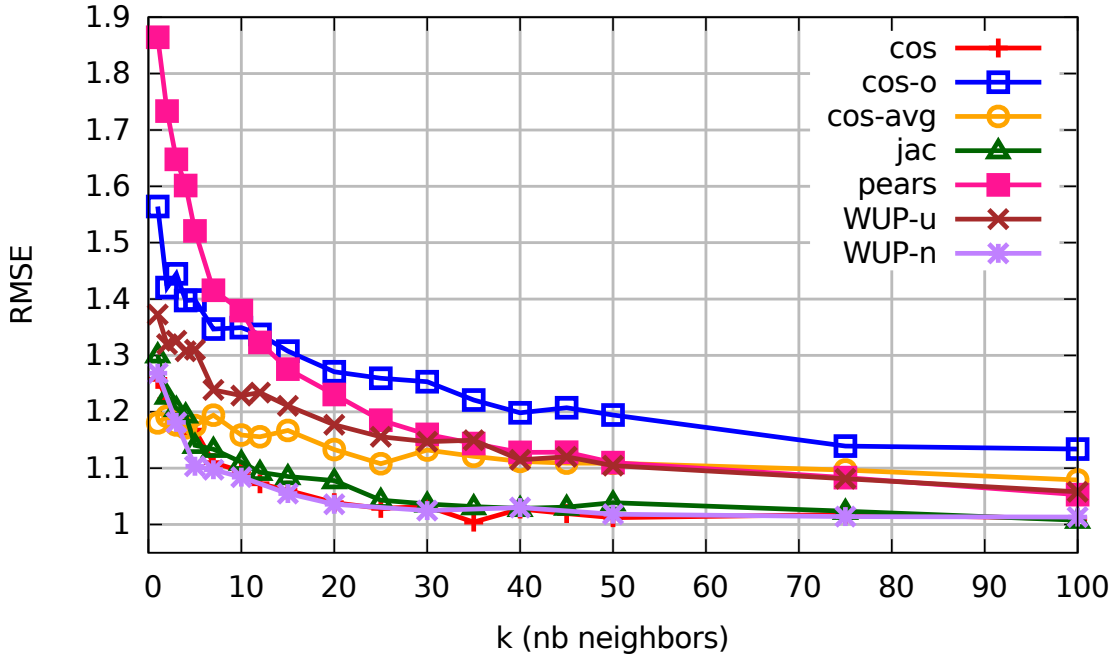

Figure 4.13 – Root Mean Square Error using 90% of the dataset for training on ML-1.

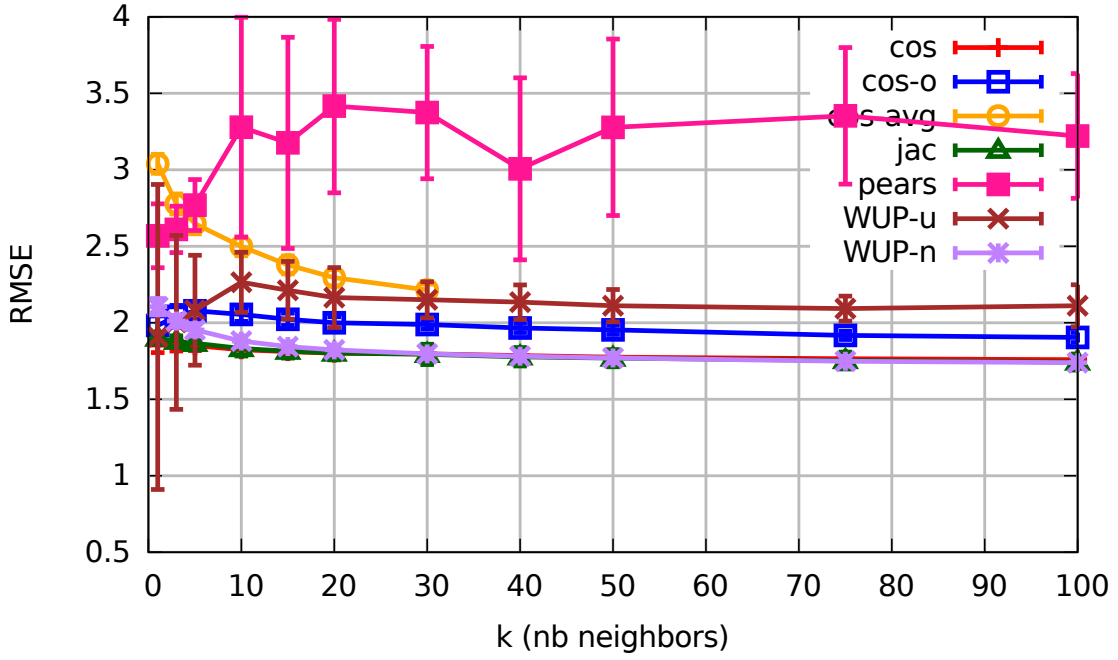

Figure 4.14 – Root Mean Square Error using 90% of the dataset for training on MovieTweetings.

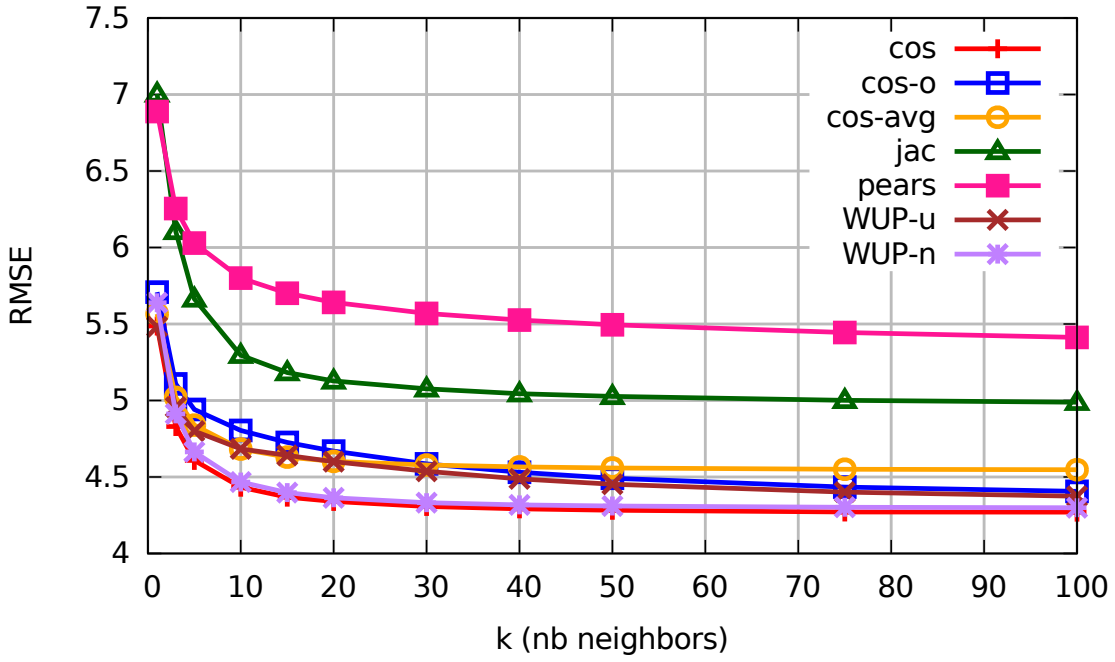

Figure 4.15 – Root Mean Square Error using 90% of the dataset for training on Jester.

4.4.3 Impact of Sparsity

An important parameter in a recommendation system consists of the sparsity of the user-item matrix. As defined in Section 4.3, sparsity measures the percentage of empty "cells" in the matrix and we have already observed how the different results in our different datasets reflect their different sparsity levels. In this section, we confirm this hypothesis, by considering five variants of the ML-1 dataset with increasing levels of sparsity.

Figure 4.16 depicts the impact of sparsity on the distribution of the similarity between users using the cosine similarity metric. The curves show that the similarity between users drastically decreases according to the sparsity. As a result, sparsity impacts not only the ability to make relevant recommendations, but also the effectiveness of the attacks we discuss in this chapter. As a side note, Figure 4.16 also explains the peculiar behavior of the *Cosine* metric in Figure 4.2. The sparsity value of MovieTweetings is in fact approximately the same as that of ML-5.

Figures 4.17, 4.18, and 4.19 depict, respectively, the proportion of expected Sybil neighborhoods, the attack's yield, and the attack's accuracy with varying sparsity levels and with 20% of auxiliary items. The figures show that the attack becomes less and less effective as the sparsity of the user-item matrix increases. The fraction of expected neighborhoods drops drastically even for the metrics that tend to facilitate the attack in the base dataset (ML-1). Accuracy (Figure 4.19) exhibits a similar behavior, confirming its strong dependence on the fraction of expected neighborhoods. Yield follows instead a more interesting pattern. While some metrics see the yield increase with sparsity, which is coherent with decreasing accuracy, others such as *Cos-overlap*, and *WUP-n* exhibit a positive correlation between yield and sparsity.

Figures 4.20, 4.21, and 4.22 present the same analysis but with Sybils equipped with 80% of the target's profile. In this case, sparsity has a weaker impact on the attack, but accuracy still decreases with all of the metrics. Finally, Figure 4.23 presents the results for recommendation quality in terms of $RMSE$. As expected the values of $RMSE$ increase with sparsity, a sign of poorer recommendation quality. This confirms the correlation between recommendation quality and accuracy highlighted in Section 4.4.2.

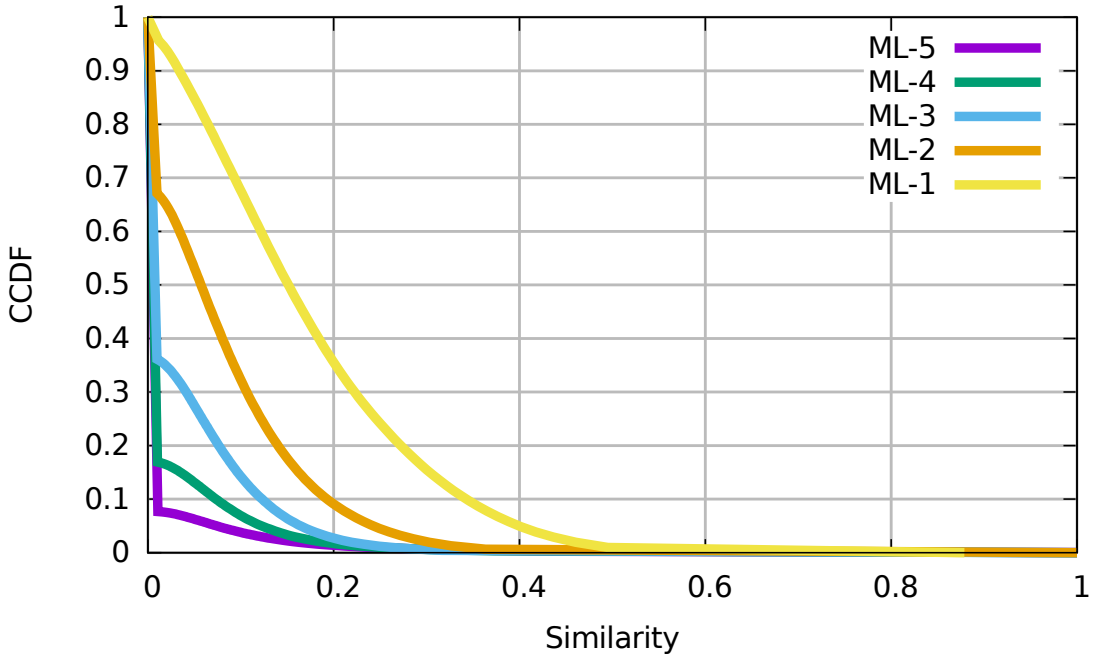

Figure 4.16 – Distribution of similarity between couple of users using cosine similarity metric for different datasets with varying sparsity.

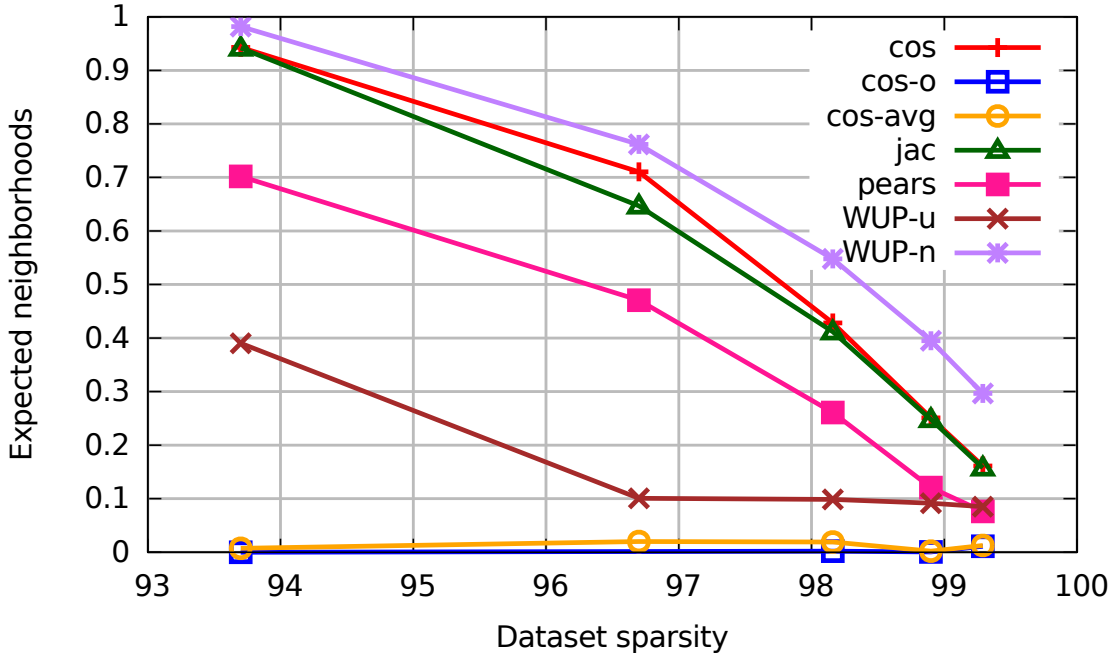

Figure 4.17 – Fraction of Sybils that obtain their expected neighborhood with 20% of auxiliary items in ML-1.

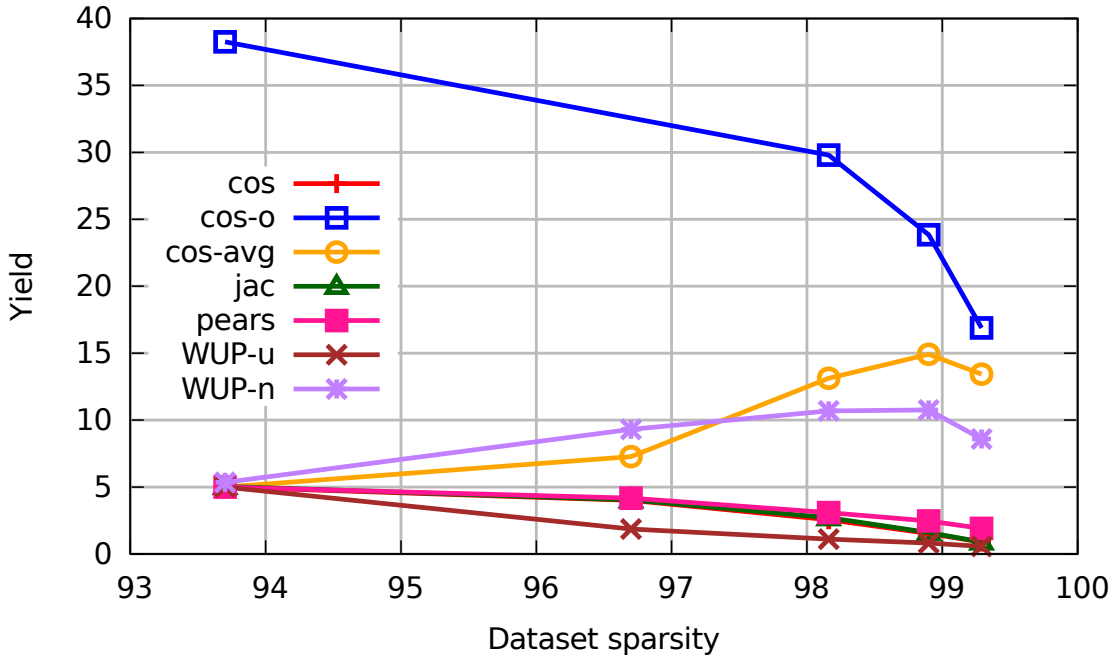

Figure 4.18 – Yield with 20% of auxiliary items in ML-1.

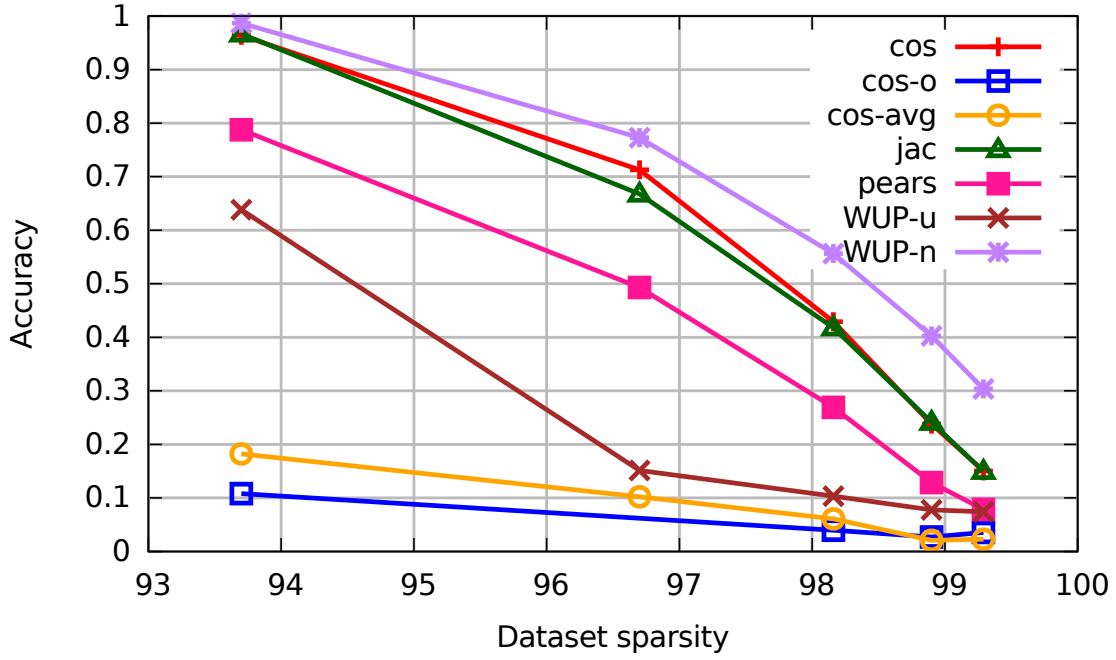

Figure 4.19 – Accuracy with 20% of auxiliary items in ML-1.

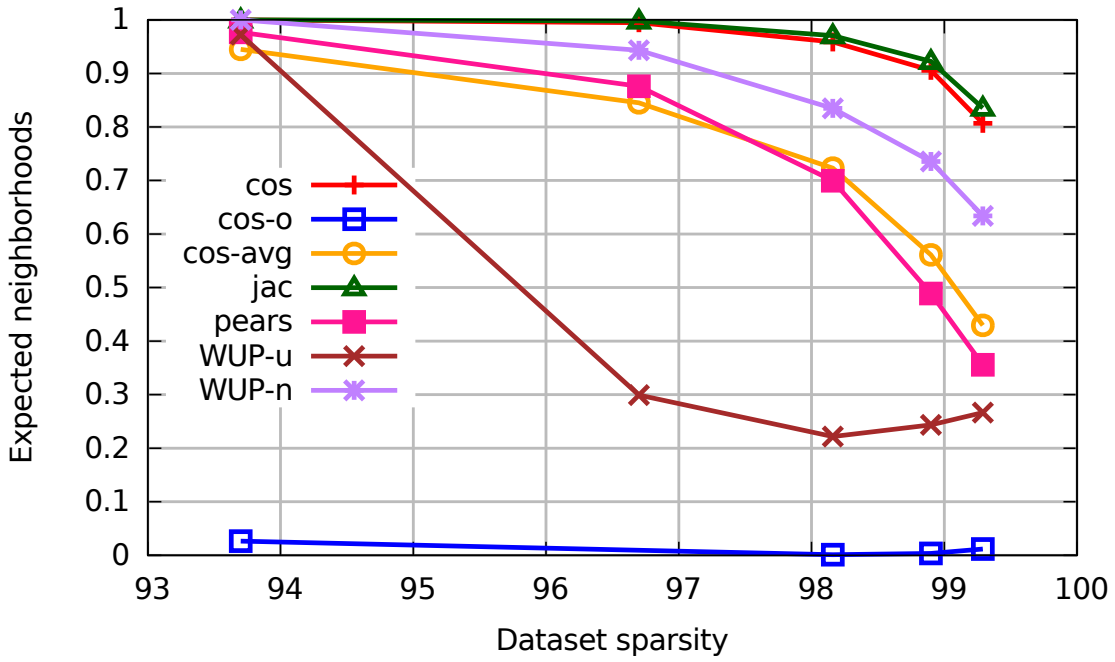

Figure 4.20 – Fraction of Sybils that obtain their expected neighborhood with 80% of auxiliary items in ML-1.

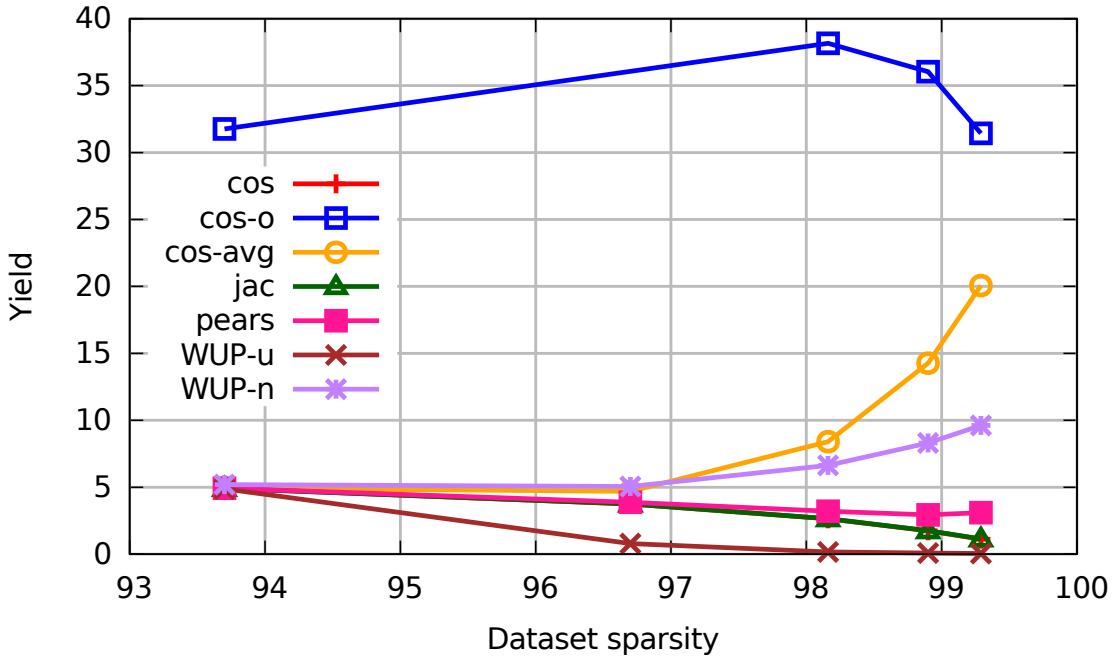

Figure 4.21 – Yield with 80% of auxiliary items in ML-1.

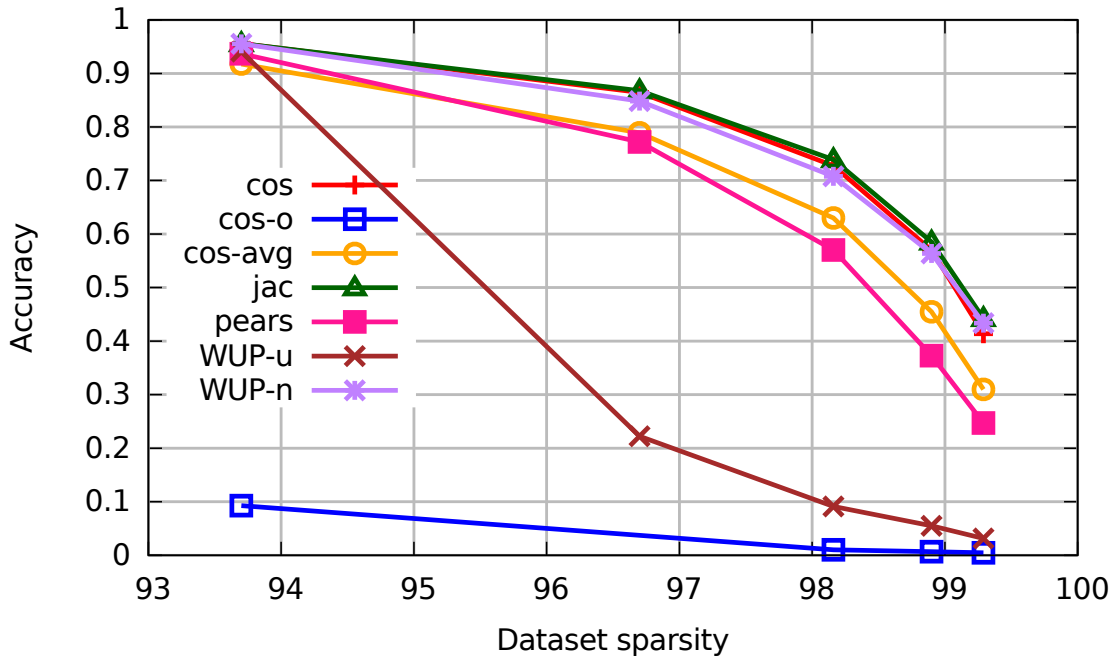

Figure 4.22 – Accuracy with 80% of auxiliary items in ML-1.

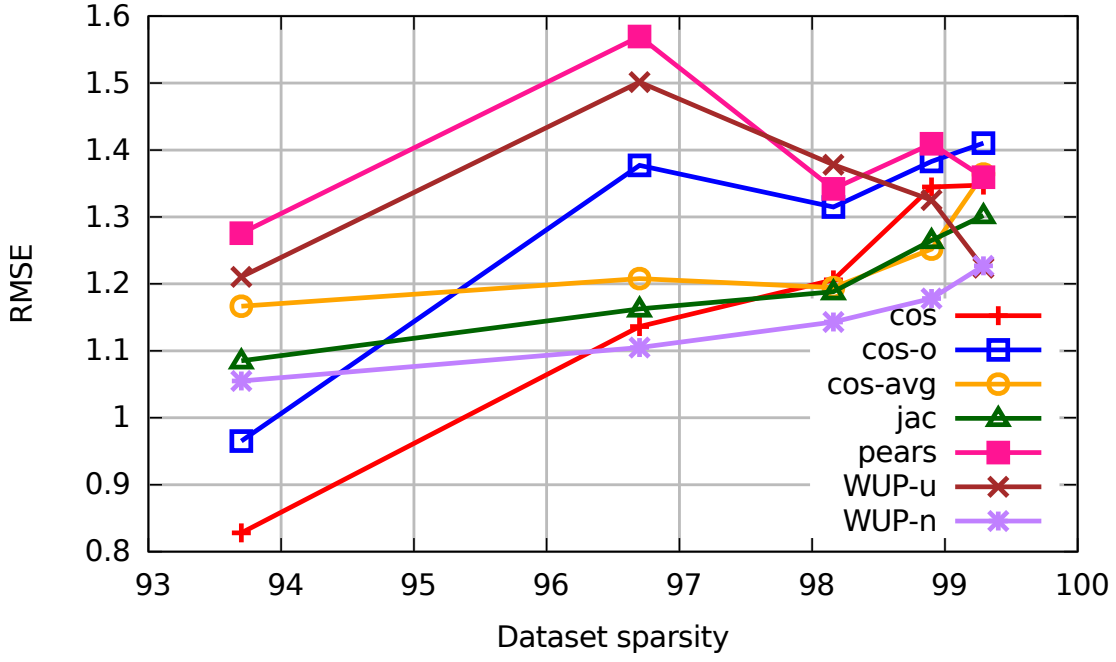

Figure 4.23 – Root Mean Square Error versus sparsity with neighborhoods of size 15.

4.4.4 Adaptive Sybils

We have so far concentrated on the effectiveness of a single round of attack. The Sybils request recommendations and attempt to guess information about the target's profile based on what they receive. In a real setting, however, an attacker may iterate the attack while incorporating the new information he/she learned about the target. In Figures 4.24, 4.25, and 4.26, we evaluate exactly this scenario on the ML1 dataset.

We configure each Sybil user so as to request one recommendation at a time. Once the Sybil receives the recommendation, we add the recommended item to the Sybil's profile with the score it has in the profile of the target, or in that of the other user that provided the recommendation if the item is not in the target profile. We then take this new item into account to compute a new neighborhood and obtain a new round of recommendations. We then repeat the process for 10 consecutive rounds.

Results shows that, with most of the metrics, the performance of the attack remains constant throughout the recommendation rounds. However, *Cos-overlap* exhibits a dramatic increase in both the fraction of expected neighborhoods and accuracy, with a corresponding decrease in yield. The reason lies in the way *Cos-overlap* treats the items that appear in the Sybil's profile but not in that of the target, and on the assumptions we made on the attacker. When a Sybil receives a recommendation, this may come either from the target's profile, or from some other user's profile. In both cases, the Sybil includes the item in its profile with the rating it has in the profile of the user that provided the recommendation. However, if the item comes from both profiles, we only consider the rating in the target user's profile. With *Cos-overlap*, a user can be in a Sybil's neighborhood only if she has a similarity of 1 with the Sybil. As a result, when a Sybil receives a recommendation that is not in the target profile, this does not penalize its similarity with the target. However, when it receives a recommendation for an item that is both in the profile of the target and in that of another user, our assumption that the Sybil can guess the profile of the target penalizes its similarity with the other user. This explains the increase in accuracy over successive rounds in the case of *Cos-overlap*.

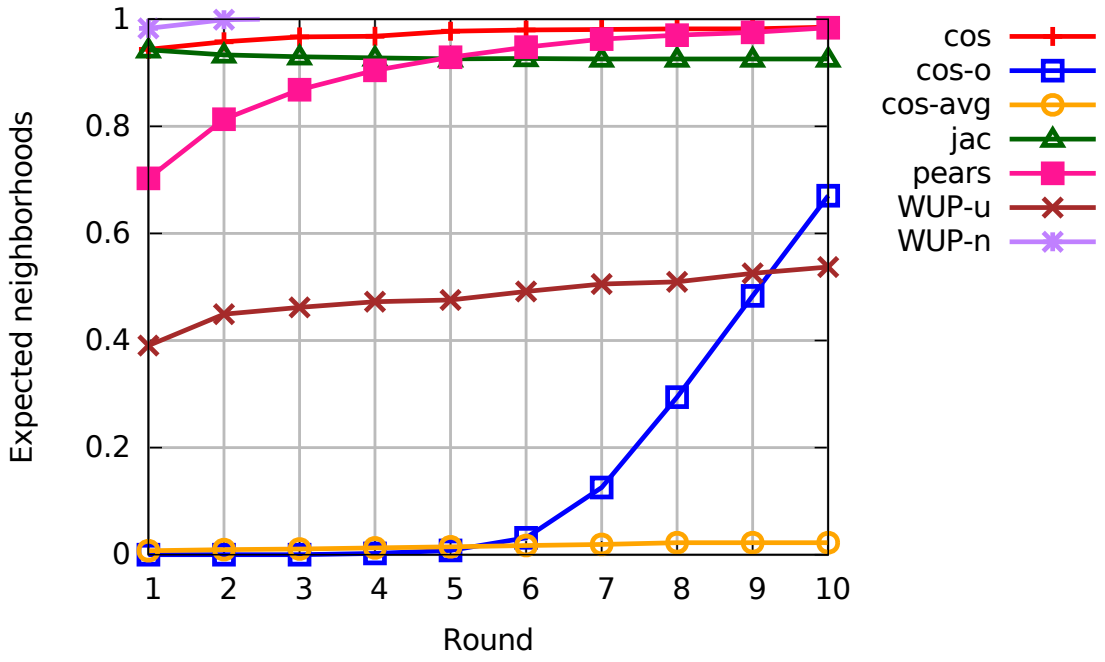

Figure 4.24 – Expected neighborhoods of adaptive Sybils on the ML-1 dataset. Sybils have 20% of auxiliary items.

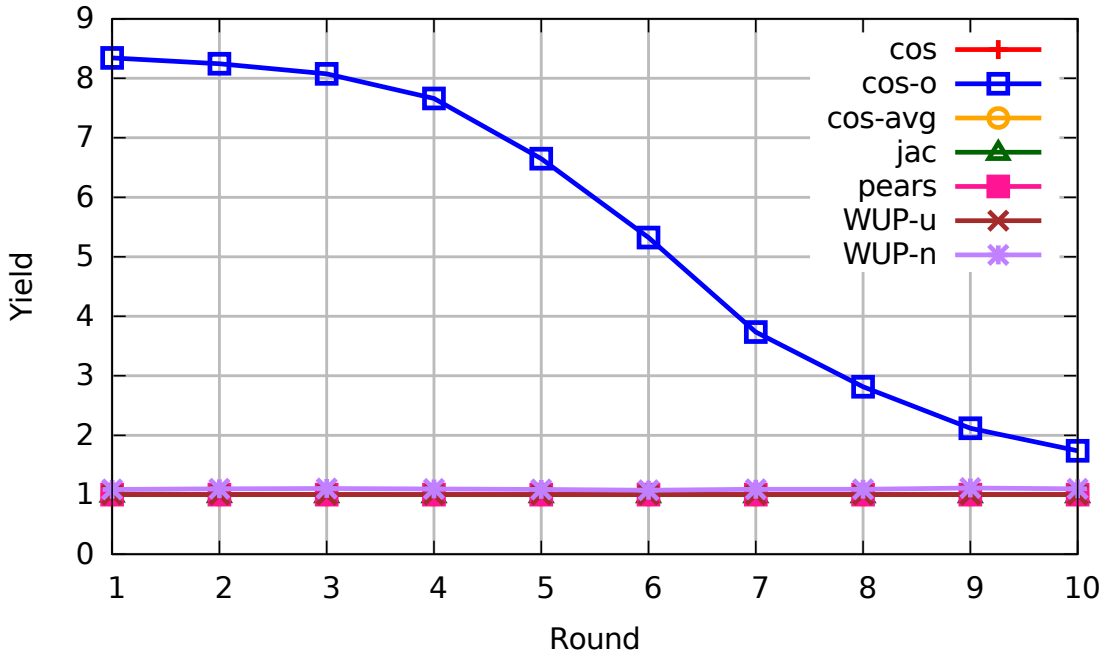

Figure 4.25 – Yield of adaptive Sybils on the ML-1 dataset. Sybils have 20% of auxiliary items.

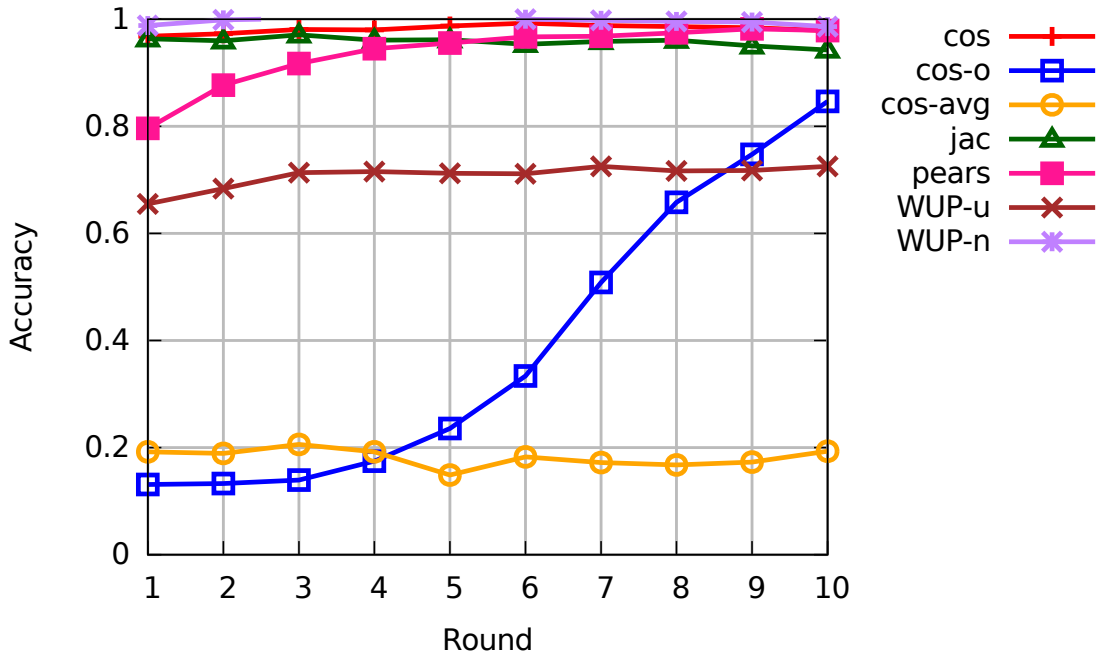

Figure 4.26 – Accuracy of adaptive Sybils on the ML-1 dataset. Sybils have 20% of auxiliary items.

4.4.5 Understanding the Sybil Resiliency of *Cos-overlap*

We observed in Section 4.4.2 that good recommendation quality correlates with a poor resistance to the attack and we expressed the need to design a metric that can improve the tradeoff between these two aspects. To address this goal, we consider the peculiar performance of the *Cos-overlap* metric on ML-1 and try to extract some guidelines for the design of a new metric.

We start by comparing the definition of *Cos-overlap* with that of the standard *Cosine* similarity. The denominator in the standard *Cosine* similarity discounts the scores of users with very large profiles thereby benefiting those that have more specific interests, but the *Cos-overlap* variant entirely removes this behavior. As described in Section 4.3, *Cos-overlap* considers only the ratings of items that appear in both user profiles and completely ignores those that appear in only one of them. This means that two users may have a similarity value of 1 even if their item sets are not exactly identical. It is enough for them to have expressed the same ratings on their common items.

Figure 4.27 visualizes this observation by plotting the distribution of the number of users that have perfectly similar counterparts in the datasets (i.e. other users with whom they have a similarity of 1). Results highlight that a very large proportion of users have perfect or almost perfect homologous users. For example, the point at $(0, 45)$ means that only 45 out of 943 users in the dataset are not perfectly similar to any other user. This contrasts sharply with the behavior of the other metrics for which no user has any perfectly similar counterpart in the ML dataset. In the following, we show that similar counterparts effectively protects users from the action of Sybils, albeit at the cost of poorer recommendation quality.

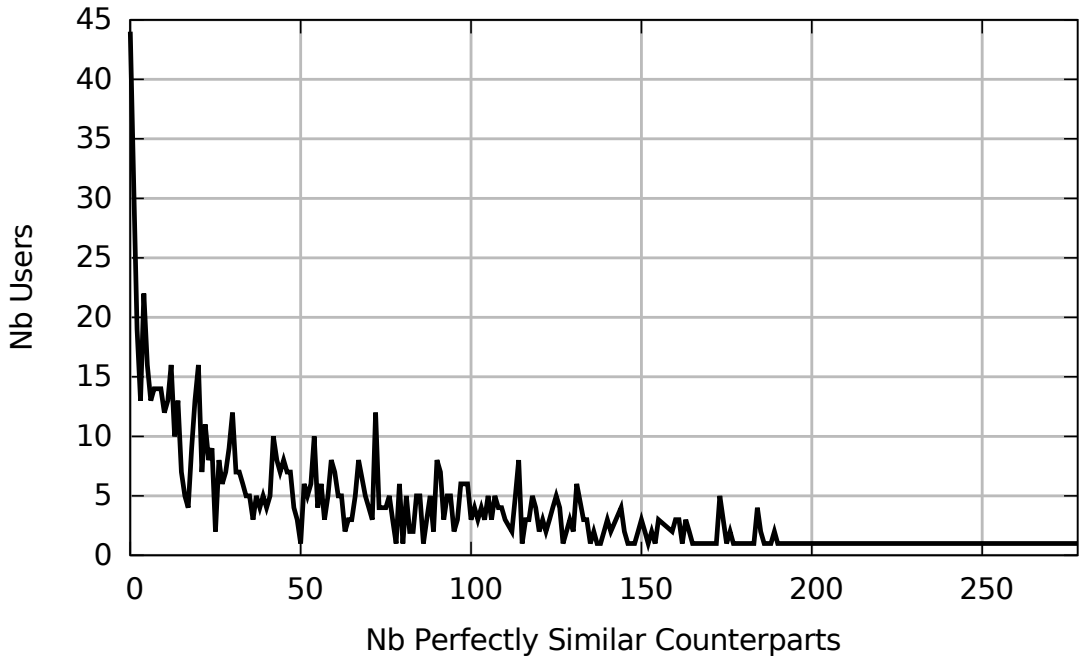

Figure 4.27 – Distribution of perfectly similar counterparts for the users in ML-1.

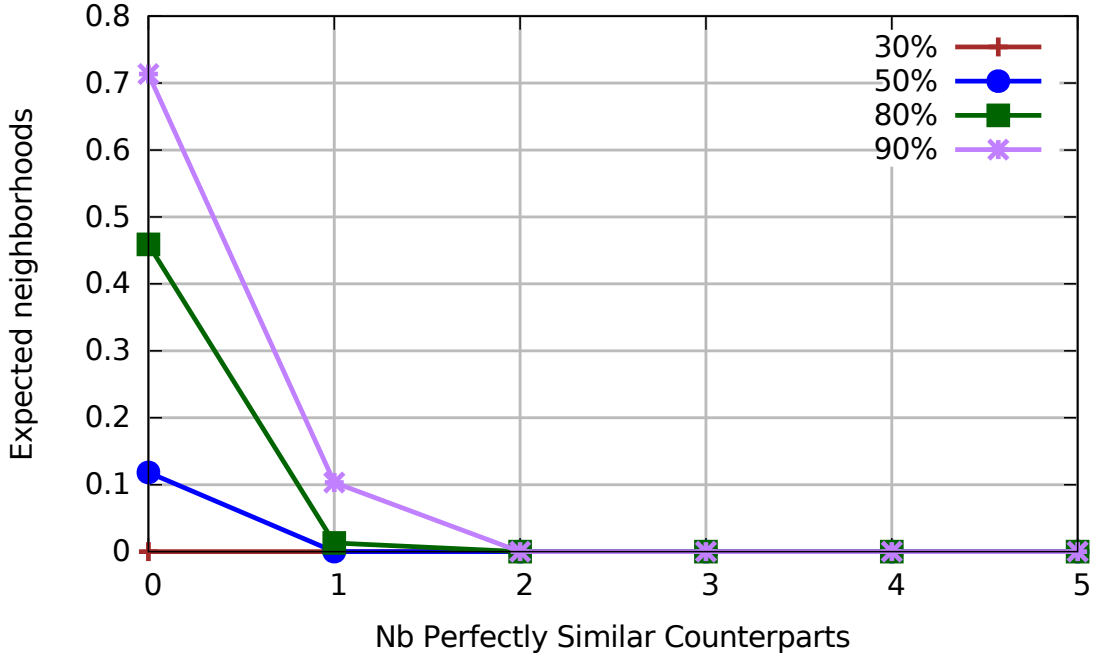

Figure 4.28 – Expected neighborhoods depending on the number of perfectly similar counterparts for the users in ML-1.

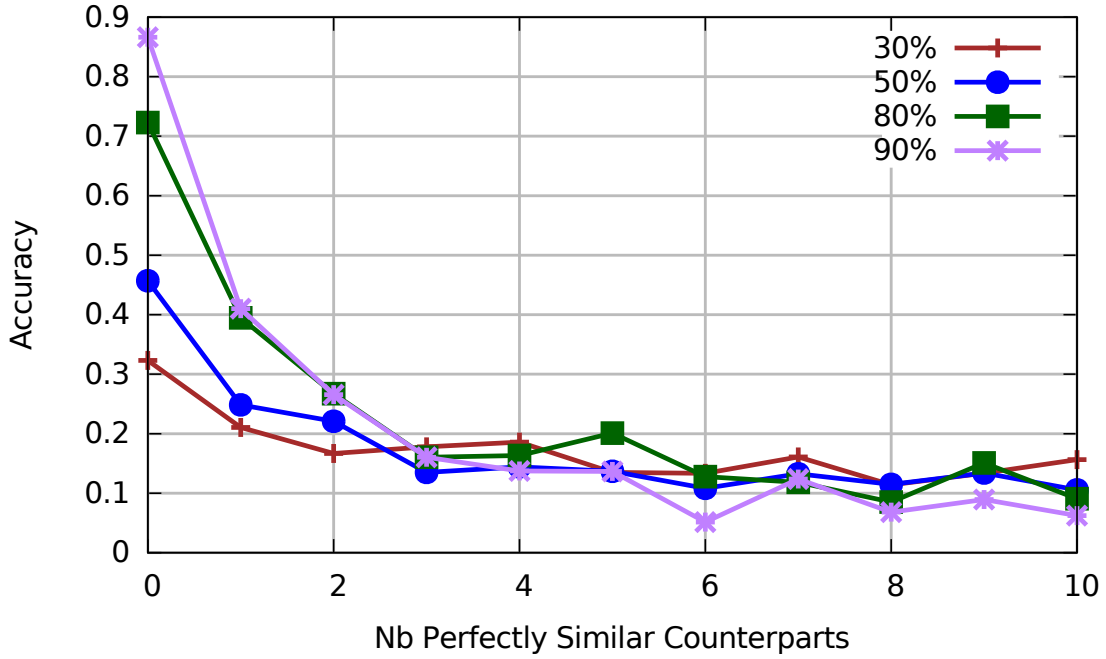

Figure 4.29 – Accuracy depending on the number of perfectly similar counterparts for the users in ML-1.

To visualize the impact of this similarity distribution on the attack, Figure 4.28 breaks down the data from Figure 4.4 and shows the fraction of perfect neighborhoods as a function of the number perfectly similar counterparts of the target. Each line corresponds to a different amount of auxiliary knowledge made available to the attacker. The data for users that have no perfectly similar counterparts pretty much follows the behaviors of the other similarity metrics in Figure 4.4. The percentage of Sybils that get an ideal neighborhood strongly depends on the amount of available auxiliary knowledge. However, as soon as the target has at least 2 perfectly similar counterparts, none of the Sybils manages to obtain an ideal neighborhood, regardless of the amount of auxiliary knowledge they have. The results for prediction accuracy in Figure 4.29 follow a similar pattern. Sybils can guess the profiles of users that are sufficiently unique, but can do little for users that have good alter-egos.

4.5 Towards a Sybil-Resistant Cosine Similarity

Our analysis of the results in Sections 4.4.1 and 4.4.2 reveals a very clear trade-off between the quality of recommendation and Sybil resistance. In this section, we propose a new metric that minimizes the impact of this trade-off. To this end, we already observed in Section 4.4.5, that the Sybil resistance exhibited by the *Cos-overlap* metric in ML-1 results from its low discriminatory power.

Specifically, *Cos-overlap* considers two users to be perfectly similar as soon as they have given the same ratings to the common items in their profiles. Thus, two users whose profiles share only one item may appear perfectly similar simply because they have given the same rating to this common item. This poor discriminatory

power makes it hard for Sybils to distinguish the target and the other Sybils from the target's perfectly similar alter-egos.

While the above property constitutes an asset for Sybil resistance, it clearly hampers the system's ability to provide good recommendations. Consider a user, A, with two perfectly similar alter egos, B, and C. A and B share a single common rating on a single common item. A and C, on the other hand, share common ratings on a significant portion of their two profiles. Clearly, C will be a better candidate than B to provide recommendations to A. But *Cos-overlap* will consider B and C as equally good.

4.5.1 $2-step$ Similarity Metric

The above observations suggest that we should try to preserve the ability to discriminate good from bad profiles for recommendations, while preventing Sybils from (i) identifying the target, and (ii) identifying other Sybils.

We satisfy these requirements by proposing a novel similarity metric: $2-step$. It is a composite metric. Given a user u, and a potential neighbor n, the first step employs one of the three top-performing metrics from Section 4.4 (for example *Cosine*), and post-processes its values so that all the potential neighbors that score beyond a certain threshold appear to be the same to u. Unlike focusing on a small subset of items as in the case of *Cos-overlap*, using a threshold makes it possible to coalesce users that are likely to provide similar results in terms of recommendation. This makes it difficult for a Sybil user to obtain a neighborhood that contains the desired target user.

The second step of the metric goes beyond the threshold and attempts to distinguish which of the top-scoring potential neighbors (those with a *Cosine* above the threshold) may be useful to compute recommendations for user u. The recommendation process consists in finding potentially interesting items in the profiles of u's neighbors. This implies that a neighbor that has no items that do not appear in u's profile brings nothing to the recommendation system and should therefore be discarded. Rather, a good neighbor should have at least some items that do not appear in u's profile. Yet, it should not have too many: users with too large profiles tend to provide less accurate suggestions.

The second step therefore differentiates the potential neighbors that score above the threshold by taking into account the number of items in their profiles that do not appear in the profile of u. Because the recommendation system computes the neighborhoods for all users, including the Sybils, this heuristic has the beneficial effect of discouraging the presence of other Sybils in the neighborhood of a Sybil user, thereby making the attack more difficult.

To summarize, the threshold makes it hard for a Sybil to differentiate the target, or another Sybil from other very similar nodes. The second step complements this feature by *preferring* legitimate users to Sybils. In the following, we describe the details of our $2-step$ metric.

$2-step$ details Let u be a user for which we have to evaluate the goodness of a candidate neighbor, w. Both, u and w may be either legitimate users or Sybils.

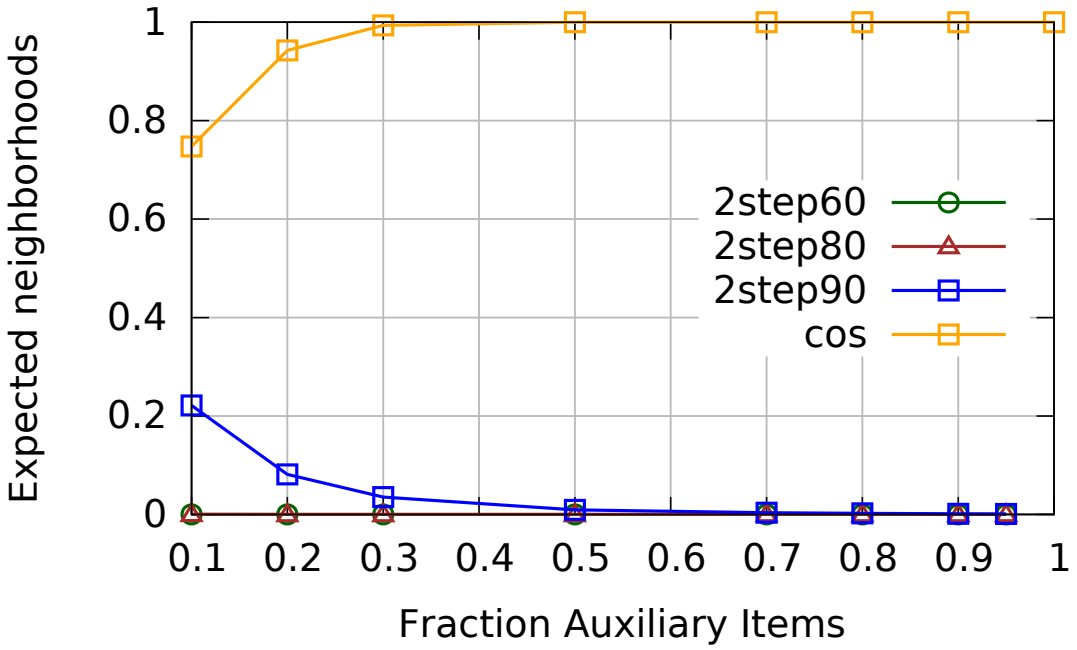

Figure 4.30 – Fraction of Sybils that obtain their expected neighborhood for the $2-step$ metric on ML-1.

Also, with some abuse of notation, let $w - u$ denote the set of items that appear in w's profile but not in u's.

In the first of the two steps, we compute the *Cosine* similarity between u and w. If $cos(u, w)$ is less than a threshold th then we use $cos(u, w)$ as the final similarity value. Otherwise we compute $th + f_i(|w - u|)$, where (i) $|w - u|$ is the number of items that appear in the profile of w but not in that of u, (ii) i is a parameter representing an ideal value for $|w - u|$, and (iii), $f_i : \mathbb{N} \to [0, 1 - th]$ is a function defined as follows.

$$f_i(x) = \begin{cases} (1-th)\frac{x}{i} & \text{if } x < i \\ (1-th)\frac{2i-x}{i} & \text{if } i \leq x < 2i \\ 0 & \text{if } x \geq 2i \end{cases}$$

Parameter i attempts to ensure that the neighbor's profile contains at least some items that are not in u's profile, but not too many. Its choice should therefore be based on the distribution of profile sizes. By combining the two above steps, we obtain the following definition for our $2-step$ metric.

$$2-step(u, v) = \begin{cases} cos(u, v) & \text{if } cos(u, v) < th \\ th + f_i(|v - u|) & \text{if } cos(u, v) \geq th \end{cases}$$

4.5.2 Evaluating $2-step$

We evaluate $2-step$ using the same metrics as in Section 4.4. To set the metric's threshold, we make a pass on the entire user-item matrix before computing the nearest-neighbor graph. For each user, we compute the distribution of similarities

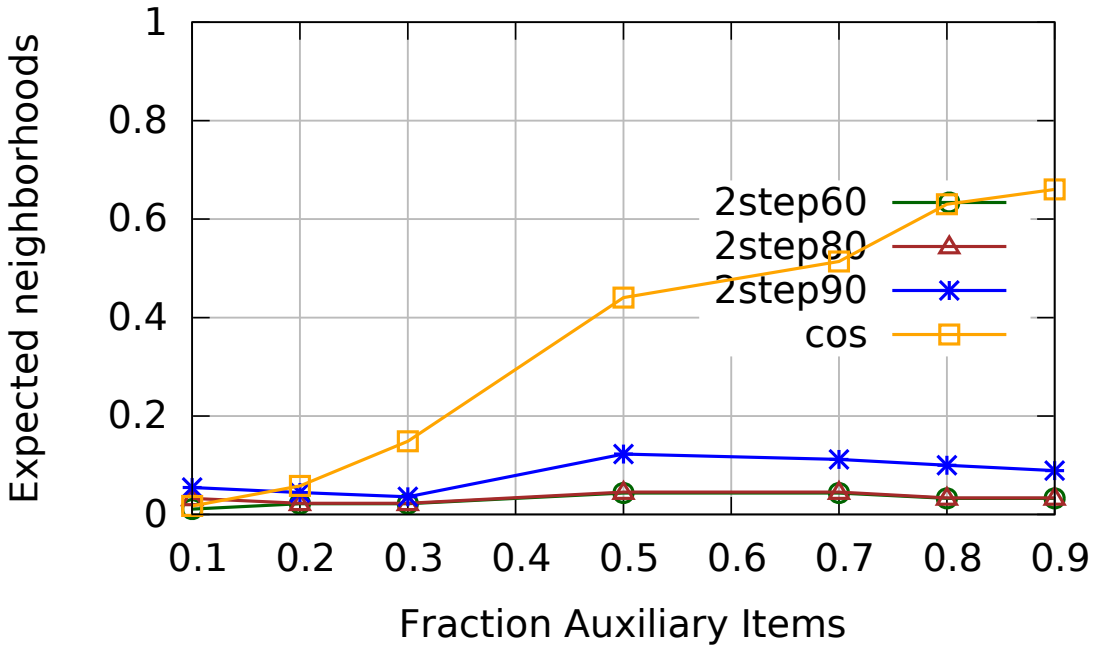

Figure 4.31 – Fraction of Sybils that obtain their expected neighborhood for the 2−*step* metric on MovieTweetings.

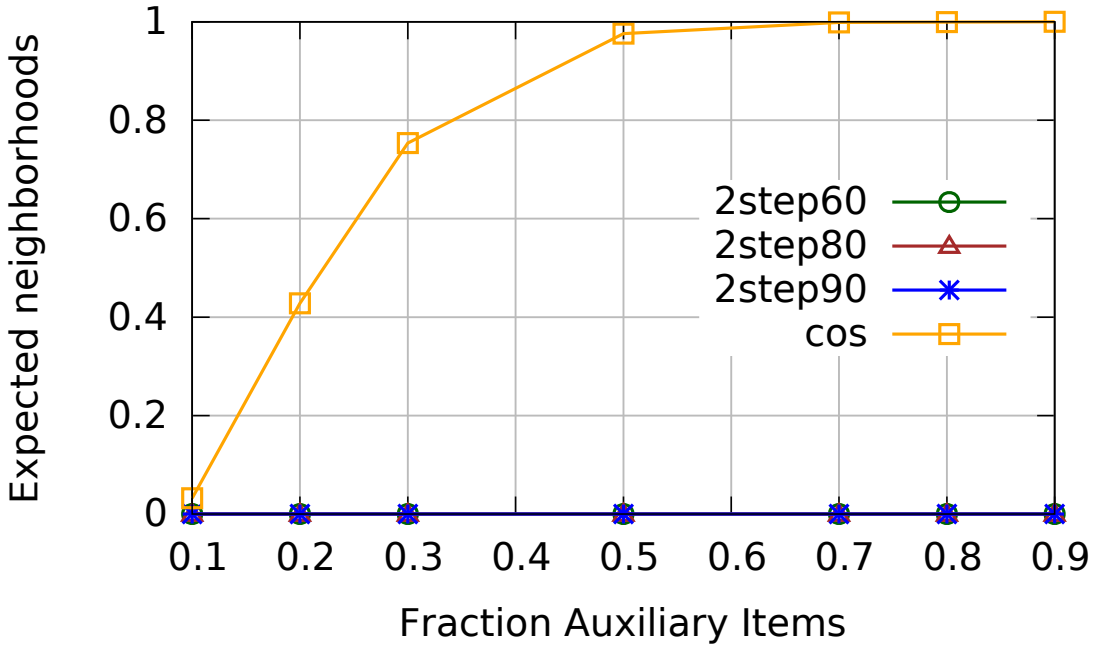

Figure 4.32 – Fraction of Sybils that obtain their expected neighborhood for the 2−*step* metric on Jester.

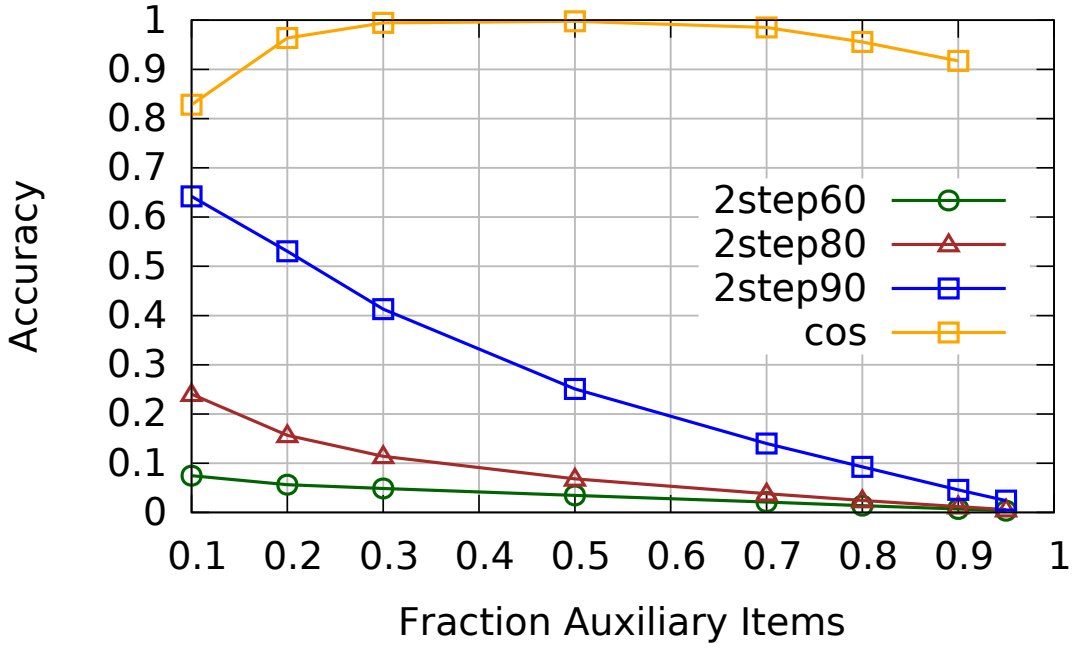

Figure 4.33 – Accuracy for the 2−*step* metric on ML-1.

with all the other users. We then round similarity values to the nearest hundredth, sort them, remove all duplicate values, and set the threshold as the t^{th} percentile of the resulting sorted sequence: we considered values of $t \in \{60, 80, 90\}$. For the *ideal number of excess items*, i, we tested values between 50 and 500: results showing limited variability, we settled on a value of 400.

These values provide a very good compromise between attack resilience and recommendation quality as we show in Figures 4.30 to 4.38. Figures 4.30, 4.31, and 4.32 compare the fraction of expected Sybil neighborhoods in the case of 2−*step* with that obtained in the case of *Cosine*. The difference is dramatic. With 2−*step*, very few Sybils manage to obtain their desired neighborhood—none with a threshold of up to 80% in ML-1 and Jester—while most of them succeed in the case of *Cosine*. Figures 4.33, 4.34, and 4.35 complement this data by showing the accuracy values of the Sybils' predictions with the two metrics. In ML-1, the high accuracy of predictions made with *Cosine* sharply contrast with the very low accuracy with the 2−*step* metric. In MovieTweetings, the accuracy of *Cosine* is already low, but 2−*step* manages to decrease it even further. Finally, in Jester, 2−*step* manages to bring the accuracy of *Cosine* closer to that of random-neighbor predictions.

While these results appear promising, the clear advantage of 2−*step* comes from its ability to combine Sybil resilience with good recommendations. Figures 4.36, 4.37, and 4.38 investigate this aspect and plots the *RMSE* values obtained by both metrics with increasing neighborhood sizes. Our novel metric closely follows the behavior of *Cosine* in both ML-1 and Jester, while it is only slightly worse in MovieTweetings. Albeit slightly higher than *Cosine*, the *RMSE* of 2−*step* remains within the range of the top-performing group in Figures 4.13 to 4.15. This allows us to state that 2−*step* provides an important improvement in the management of

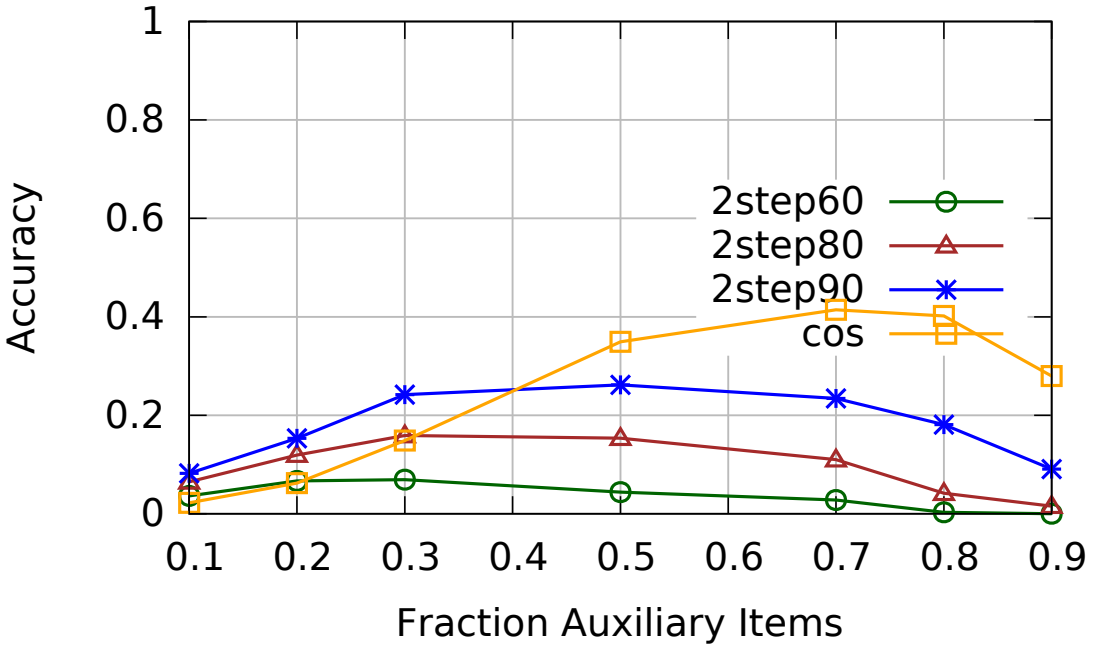

Figure 4.34 – Accuracy for the 2−*step* metric on MovieTweetings.

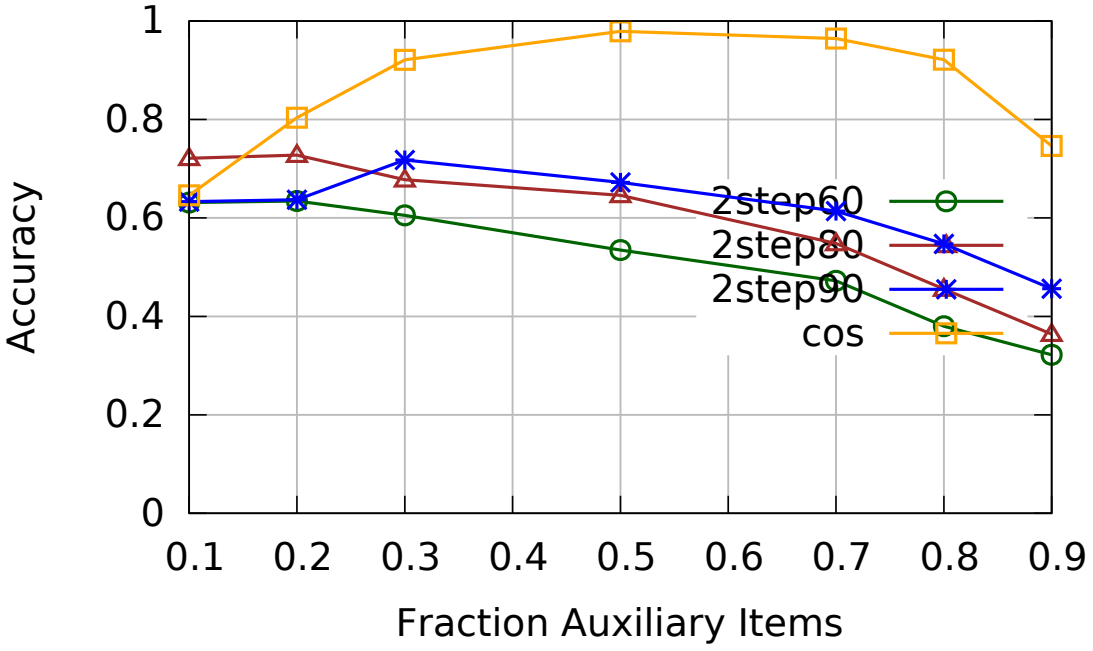

Figure 4.35 – Accuracy for the 2−*step* metric on Jester.

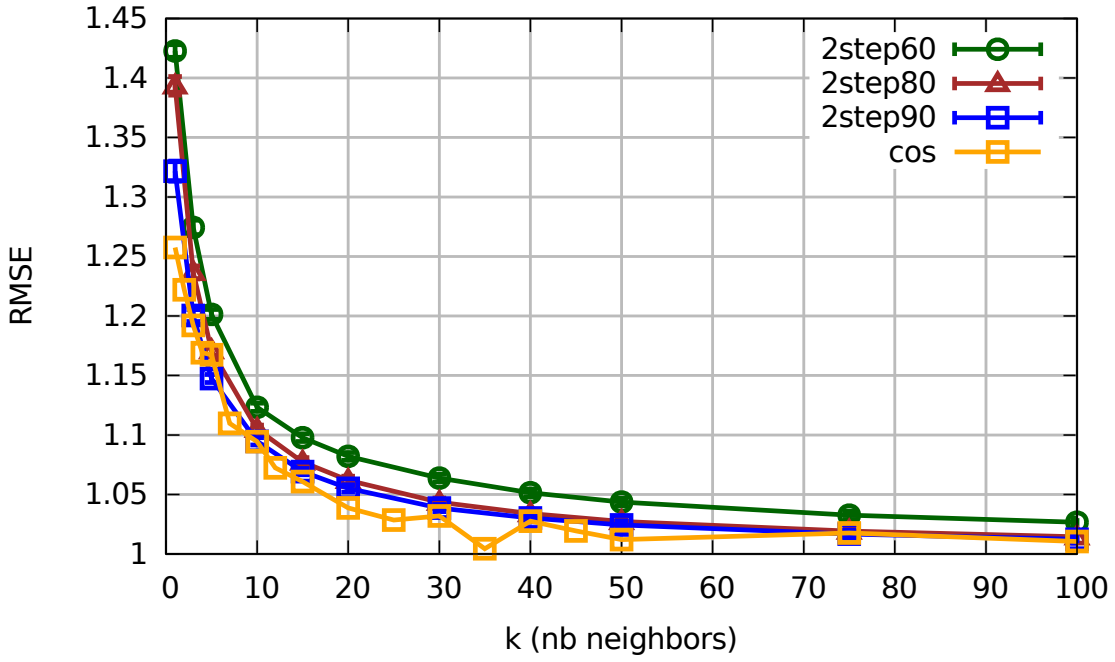

Figure 4.36 – Root Mean Square Error for the 2−*step* metric on ML-1.

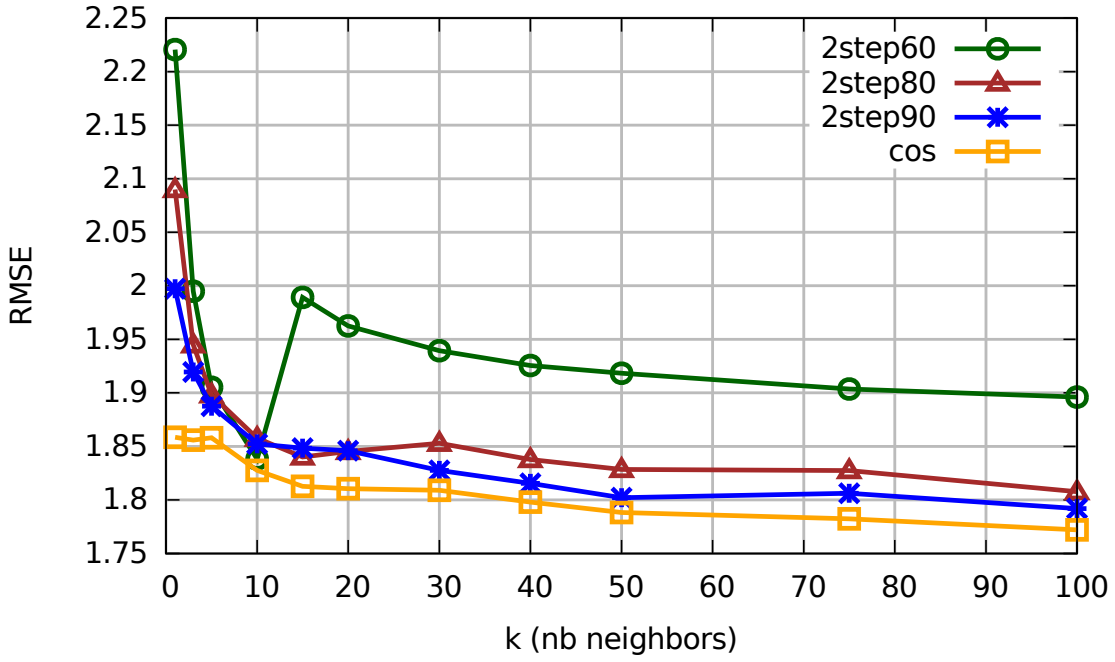

Figure 4.37 – Root Mean Square Error for the 2−*step* metric on MovieTweetings.

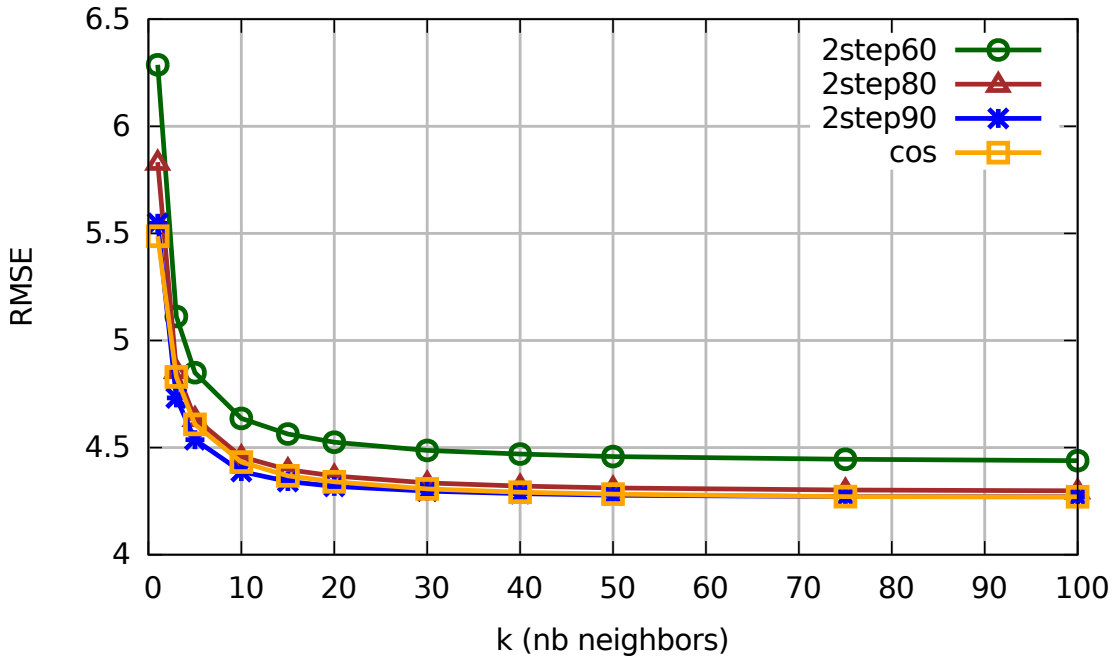

Figure 4.38 – Root Mean Square Error for the 2−*step* metric on Jester.

the trade-off between Sybil resistance and recommendation quality.

To conclude this section, we acknowledge that our evaluation of 2−*step* is still preliminary. A more thorough evaluation as well as guidelines on how to tune its parameters are planned as future work.

4.6 Concluding Remarks

We presented a comprehensive experimental analysis of the impact of similarity metrics on the Sybil resilience of existing user-based collaborative-filtering systems. Our results show that while attacks are generally effective, some metrics offer significant protection by trading off some of their recommendation accuracy. Our results, obtained on a state-of-the-art recommendation framework highlight the requirements for a Sybil-resistant collaborative-filtering system, and allow us to introduce a novel similarity metric that combines the benefits of high recommendation quality, and resilience to attacks.

Our results open new avenues for interesting research, though they are beyond the scope of this thesis. In the future, we plan to carry out a detailed experimental analysis of our 2−*step* metric, in order to highlight its requirements, and its best configuration according to the characteristics of the recommendation system. We also plan to explore variants of the attack, for example by considering what the attacker can guess in the absence of auxiliary information, or in the presence of imprecise information.

Chapter 5

Conclusion

5.1 Summary of Contributions

In conclusion, we presented two contributions in this thesis. Each contribution aims at thwarting a different type of attack against the privacy of users of recommendation systems (RSs) based on collaborative filtering (CF).

First, in Chapter 3, we consider user privacy in the context of decentralized collaborative-filtering systems for two reasons. We choose decentralized architectures because centralized ones inherently suffer from the threat of "Big Brother" adversaries. We choose CF techniques for their ability to provide good recommendations, and their fitness to decentralization.

The most immediate threat to user privacy in our context of decentralized CF comes from the need for peers to communicate with many other peers to make up for the lack of a central coordinator. Therefore, it is easy for ill-intended users to participate in the system in order to learn information about many peers. We name this type of adversary "Little Brothers".

We propose *Hide & Share* (*H&S*) as a computationally lightweight solution against threats by "Little Brothers" in peer-to-peer systems. *H&S* allows two users to compute an approximation of their similarity without requiring them to reveal their profile to each other. *H&S* makes it possible by indirectly computing two users' similarity, *i.e.* by comparing each profile with a set of randomly generated profiles which we call landmarks. Thus, similarity values with landmarks act as a user's coordinates in a coordinates system. Meanwhile, these similarity values do not reveal much information about a user's profile because it is difficult to reverse them and reconstruct that user's profile.

We show through simulations on three datasets of traces from real RSs that the quality of recommendations made by a *H&S*-based system is reasonable compared to a similar system without privacy-preserving mechanism. Moreover, the bandwidth overhead of *H&S* is reasonable too by today's high capacity networks, while its requires less storage than a similar system without privacy-preserving mechanism. The recommendation quality traded by *H&S* for privacy enables it to outperform a mechanism based on profile obfuscation by random perturbation. Finally, we also demonstrate formally *H&S*'s privacy guarantee by giving an information-theoretical upper bound on the amount of information it reveals.

Second, in Chapter 4, we consider user privacy in the context of an adversary trying to trick both centralized and decentralized user-based collaborative-filtering systems into revealing private user data. The rationale of this kind of attack is to exploit the very idea at the core of memory-based CF, since CF draw recommended items from the profiles of a user's neighbors, these attacks try to create the conditions for the system to recommend items from the target user's profile to an adversary-controlled user. Specifically, we study the version of this attack from [21] which uses Sybil users, and some auxiliary information about the target user's profile. In this attack, the adversary creates k fake users (a.k.a. Sybils) who all share the same profile made of a subset of the target user's profile (a.k.a. auxiliary information), gleaned by the adversary using auxiliary channels. By doing so, she tries to create the conditions in which the CF system select for each Sybil a KNN consisting in the $k-1$ other Sybils, and the target user. If these conditions are met, any item recommended to a Sybil user comes from the target user's profile with high probability.

We thoroughly study the behavior of this attack via simulations using a state-of-the-art recommendation framework and three datasets of traces from real recommendation systems. We observe that the effectiveness of the attack in creating the right conditions depends more on the similarity measure used by the system than on the amount of auxiliary information that the adversary has about the target user. Indeed, when a similarity measure considers many users to be very similar to the target user, the probability that the right conditions will be met are low. Consequently, we observe a strong correlation between the characteristics of the system's user population, and the success rate of the attack. We also observe that, among the different similarity measures we evaluated, those which yield the best recommendation quality are also those which are the less resilient to the Sybil attack.

We propose $2-step$ as a solution against this Sybil attack of CF systems, regardless of their architecture. $2-step$ is a composite similarity measure which not only significantly reduces the success rate of such attacks, but it also enables the system to provide recommendations of good quality. $2-step$ works in two stages.

This measure starts by modifying the similarity of all users scoring above a given similarity threshold so that they become appear as perfectly similar. This stage prevents the right conditions for Sybils from happening by making all users who are similar enough to the current user look perfectly similar. Note that this does not significantly hamper recommendation quality because recommendation does not require perfect knowledge. Rather, it only requires users who are approximately as similar as those found in the optimal KNN.

Then, $2-step$ applies a second modification to the similarity of some of these users in order to put forth the most likely users to enhance recommendation quality. This is done by slightly increasing the similarity value of users whose profile contains a few new items for the current user. Conversely, it slightly decreases the similarity of users with large profiles as they tend to provide less precise recommendations. The benefits of this stage are twofold. It not only makes the system favor potential neighbors bringing new items, but it also make Sybils less likely to be selected as neighbors as their profiles are identical.

5.2 Limitations

Like any contribution, ours are not without limitations. As a consequence, we discuss the limitations of *H&S* and 2−*step*.

We first discuss some limitations regarding *Hide & Share*, specifically in terms of adversary model, of privacy protection for the whole recommendation process, and of compact profiles configuration.

In our privacy evaluation of *H&S*, we consider a model of adversary which is a slightly more powerful one than the classic honest-but-curious (a.k.a. semi-honest) adversary. It means that in the honest-but-curious model, the adversary performs the *H&S* protocol normally (honest aspect) but she tries to infer private user data with any information part of the protocol's output (curious aspect). Our adversary model, which we call the curious adversary, is similar to the honest-but-curious one, except that the adversary has three additional active attack capabilities: she can tap unencrypted communications; she can attempt to bias multi-party computations; she can compute her similarity with her target as many times as she wants. However, our curious adversary cannot collude or create Sybil users. This is a limitation of our evaluation because collusions and Sybil attacks are capabilities often deemed realistic for real life adversaries in peer-to-peer systems.

Moreover, a stronger dishonest adversary could conceivably lie about its landmark coordinates during the *H&S* protocol. During her first similarity computation with her target, she could give any coordinates and receive her target's coordinates in exchange. She could then give her target's previous coordinates on subsequent computations in order to appear similar. This would make the adversary likely to become a neighbor of her target, in turn increasing her probability to discover her target's other neighbors, and perform more similarity computations.

Another limitation of *H&S* is that it only provides privacy-preservation at the first logical step of user-based CF. From a high level point of view, this type of collaborative filtering works in two logical steps: finding user A's K-Nearest-Neighbors (first step), and selecting the items to recommend to A among the candidate items from the profiles of A's neighbors (second step). *H&S* preserves A's privacy at the first step because it not only protects A's profile during similarity computations, but it also mitigates privacy leaks via A's neighbors. Indeed, *H&S*'s approximation of users similarity induces that A's neighbors are not A's optimal KNN, so even if the adversary knows who are her target's neighbors, she cannot infer her target's interests from this knowledge. However, one also needs privacy-preservation at the second step for the whole recommendation process to be privacy-preserving.

H&S implements compact profiles using Bloom filters so that user profiles and landmarks can be mapped within the same finite space, and for the fact that Bloom filters may return false positive, which is beneficial for profile privacy. The false positive rate of a Bloom filter depends on three parameters: the filter's size (number of bits), the number of hash functions it uses, and the expected number of elements stored. A limitation of our evaluation is that we chose empirically the values for these parameters. One could find analytically these values based on a desired false positive rate and the average number of items in a user's profile.

We now discuss some limitations regarding 2−*step* and the Sybil attack, specif-

ically in terms of knowledge about the RS granted to the adversary, of the attack dependency on the RS's similarity tie-break strategy, of Sybil attack detection, and of parameter values for $2-step$.

A first limitation of our study of the Sybil attack on CF systems is that we consider that the adversary knows the value of k, the size of user neighborhoods. However, it is not trivial for the adversary to discover this value as it is an internal parameters of the recommendation system which the operator does not typically make public. This implies that the results we show correspond to the most favorable real life situations for the adversary. A possible heuristic to discover k is for the adversary to performs a series of Sybil attacks, with several values of k selected by an exponential back off strategy, on a user she controls. The adversary first creates a non-Sybil fake user U whose profile she fills with random items. Then, she creates k Sybil users with profiles containing a relevant percentage of U's profile, depending on the amount of auxiliary information the adversary expects to glean in a real attack. Subsequently, she performs one Sybil attack after the other, starting with a small k, and increasing it exponentially until she reaches a satisfactory rate of successful guesses. She repeats this process starting with a new k value which is half of the last value yielding the expected success rate, until the adversary finds the smallest k which meets the same criterion.

A limitation of the Sybil attack itself is that its success strongly depends on the collaborative-filtering system's strategy to break similarity ties between potential neighbors. In the case where the KNN algorithm can choose from several potential neighbors who are all very similar to the current user (*e.g.* similarity of 1), it must make an arbitrary choice. In our evaluation, we considered a system breaking ties by selecting equivalent neighbors randomly but other strategies exist, such as breaking ties based on user IDs. On the one hand, our choice seems to be the fairest strategy, and it is less favorable to the adversary than an ID-based strategy. On the other hand, it makes the composition of neighborhoods less stable, which is a property that some RS operators might want.

Another limitation of the Sybil attack that we touch on is its high detectability in its current form, in a centralized system. Indeed, because all Sybils have exactly the same profile, it is easy for the system's operator to spot them by watching users who registered at similar times, and whose profile becomes identical after a short period. However, this can be circumvented by varying the Sybil profiles, *e.g.* the adversary can partition its auxiliary information between the different Sybils.

Finally, a limitation of $2-step$ is how to choose the value of its parameters, namely the similarity threshold th, and the ideal number of excess items in a potential neighbor's profile i. In our evaluation of $2-step$, we determined the best values of th and i empirically. It should be possible to devise analytical heuristics for them.

5.3 Future Work and Opening Questions

We now discuss future works to improve our contributions, as well as some opening questions which go beyond the particular scope of this thesis.

First of all, we already introduced possible future works in the previous section

when illustrating some limitations of our approaches. For instance, such future works regarding *H&S* include developing a systematic method to determine the best parameters for Bloom filters as used by *H&S*, or studying the evolution of *H&S*'s empirical and theoretical privacy protection when we consider an adversary able to collude with other users. Similarly for $2-step$, example future works stemming from the aforementioned limitations include tweaking the Sybil attack so that it is less detectable, or studying how the adversary can discover the value of k in practice.

We also envision future works which extend our contributions instead of just removing some of their limitations.

A research direction is to evaluate if *H&S* has intrinsic properties regarding resilience to the type of Sybil attacks that we studied in Chapter 4. As we mentioned in Section 3.4.4, it is worth studying if other bit-commitment schemes which are more bandwidth-efficient than the basic protocol of Blum can be used in *H&S*'s P2P context. Another interesting extension of *H&S* is to include some sort of proof-of-work [57], similarly to Bitcoin, in order to further deter adversaries from intentionally computing their similarity too often with their target. A proof-of-work is a piece of data which is costly to produce, in terms of computation or time, but easy to verify. Furthermore, it would be relevant to evaluate the privacy-preserving aspect of Bloom filters on their own. If compact profiles can be configured so as to provide a decent privacy protection, this would help in fine-tuning *H&S* to yield the best trade-off between recommendation quality and privacy.

Regarding $2-step$, it would be interesting to study whether this measure can be applied to model-based CF techniques, and if so, what adaptations this would require. Alternatively, it should be possible to strengthen $2-step$ by adding a small variance to its threshold similarity th. This way, even an adversary who discovered the value of th, would have troubles keeping the similarity of its Sybil users below the threshold. Another research direction is to explore ways to obtain auxiliary information in a more systematic fashion. Otherwise, it is worth studying whether a similar kind of Sybil attack can yield something valuable when the adversary does not have any auxiliary information. Perhaps one can reliably learn private data by using a random profile for the Sybils.

Finally, we end this manuscript by discussing opening questions related to the works we presented.

In this thesis, we argue that recommendation systems, and especially collaborative filtering ones, should integrate privacy-preserving mechanisms otherwise they will suffer from attacks and privacy leaks. However, as we hinted in Section 2.2, there is no publicly reported real large-scale privacy attack against a RS, to the best of our knowledge. Therefore, it is worth asking ourselves: are such privacy attacks real or only theoretical threats? The only way to answer this question with certainty is to ask large RS operators or run such a system ourselves. In either case, it is unlikely that we will ever get a definitive answer.

Nevertheless, we explore hypothetical situations and conclude that the best course of action is to integrate privacy-preserving mechanisms.

Let us suppose large-scale privacy attacks against RSs are a reality. Then, there is a least two explanations to why no one hears about them. First explanation: victim operators may prefer to keep quiet about it to preserve their image and/or their

business, as well as to surreptitiously track the attackers while they are unaware that they have been discovered. Second explanation: no one expects such attacks thus no one watch for them, and they go completely unnoticed. This is plausible because some attacks such as the passive one from [21] are difficult to notice. Regardless of the reason for keeping reports private, these attacks are bound to eventually attract public attention if they are indeed real because either a victim operator will be forced to admit the breach, or someone will track down the source of some private data sold on a black market.

Alternatively, let us suppose that large-scale privacy attacks against RSs are only theoretical. It could be that no one is interested in attacking RSs to gain private data because it is not lucrative enough, or there is easier targets. Another possible explanation is that operators of commercial RSs patched their systems against the attacks studied in the literature. Whatever the explanation, it is unreasonable to dismiss the need for privacy-preserving mechanisms based on the fact that such attacks never happened before. Moreover, the increasing pervasiveness of recommendation systems makes them more likely to be the target of a large-scale attack in the future. We can draw a parallel here between online advertising networks (a.k.a. ad networks), intermediaries between advertisers and websites seeking to sell ad space, and RSs. Online ad networks became ubiquitous on websites since the mid-1990s [20], and they are now so successful that in 2011 their revenue exceeded that their cable television counterparts in the USA [89]. Despite their economic significance nowadays, typical online advertising networks failed to address their security and privacy issues, even in the presence of solutions in the literature [40]. This led to a quick growth of malvertising, *i.e.* diverting ad networks to silently serve malwares to wide audiences, in the 2010s [30].

So if it is only logical that recommendation systems implement privacy-preserving mechanisms, how come commercial systems did not do it already? The most probable cause lies in economics, that is companies lack economic incentives to implement privacy-preserving mechanisms in their RS. The first deterring factor is that any privacy-aware RS implies some loss of recommendation precision. In a context where companies use RSs to increase their revenue, a loss of precision is not acceptable as it probably means a loss in revenue. This factor is even more relevant to companies which are publicly traded or funded by venture capitalists because investors often value more short-term profit. This priority for investors also explains why the prospect of a privacy attack on the company's RS is not a big enough incentive despite the harm it would do to the company's image if it were revealed.

In the case of companies where the core business is providing recommendations for others, another factor preventing the advent of commercial privacy-preserving recommendation systems is the high level of competitiveness of that market segment. Indeed, implementing privacy-preserving mechanisms requires R&D, and running them in production increase costs, either through the additional computational power required or the reduced rate of recommendations per unit of time. However, being respectful of users' privacy is not yet an important enough competitive advantage to compensate for its costs. So it is unnecessary and even dangerous for a company to spend costs on something which will not give it an edge over competitors.

Moreover, the dominant business model for online-based companies revolves around making use of the value of personal data since several years. This creates an inertia which makes it harder for companies departing from this business model to thrive. From the point of view of investors, it takes more convincing to elicit their funding. From the point of view of users, they have become used to using online services without paying a monetary fee. That being said, this situation is starting to change and a market for privacy-aware online services seems to start forming, as attested by the success of companies such as DuckDuckGo [1] or ProtonMail [2], especially since the Snowden revelations.

We believe that the solution for companies, including those running a RS, to embrace a respectful stance towards user privacy lies in a mix of technology, economics, and user awareness.

Finally, we touch on the subject users of recommendation systems. Nowadays, the most common setup of a RS is as a companion to a website. In this context, an individual visiting such a website does not explicitly choose to use the RS. The individual may not even become aware that, by viewing some webpages she is interacting with the RS, until she actively looks for more content on the same website only to discover that her interests have been inferred. Moreover, most such websites do not offer the option to opt-out of the recommendation system. Use of the RS is imposed on individuals who want to access the website's content. Thus, can RS users really be considered as users? Naming them users, as is the case in the literature, implies that they use the system knowingly or that it is the result of a conscious choice, yet they really are visitors in the aforementioned context. Besides this inconsistency of terminology, the real issue is that it may be fair to trade some private information in exchange for the utility of recommendations, but it not acceptable when this compromise is not the result of an informed choice. Therefore, it is morally dubious to assume that all visitors accept by default to participate in RSs. This position is even less honestly sustainable when considering systems without privacy-preserving mechanisms. In conclusion, we consider that participation in recommendation systems should be opt-in by default, or at the very least that their presence should be notified to visitors as is already the case for cookies for instance.

[1] `https://duckduckgo.com`
[2] `https://protonmail.com`

Annexe A

Résumé en français

A.1 Motivation

Les systèmes de recommandation sont apparus dans les années 1990 afin de résoudre la surcharge d'information. En effet, la démocratisation des accès personnels à Internet et les débuts du commerce en ligne durant cette période ont provoqué une importante augmentation de la quantité de contenus ou d'informations disponibles. Cela a rendu n'importe quelle personne incapable de traiter toutes les informations aussi vite qu'elles sont produites, suscitant ainsi la création de systèmes capables de filtrer automatiquement les informations tels que les systèmes de recommandation. De nos jours, les systèmes de recommandation ne sont pas seulement omniprésents dans les systèmes d'information, et plus particulièrement sur le web, mais ils sont aussi au cœur de la manière dont beaucoup d'entreprises mènent leurs affaires [44]. Par exemple, le système de recommandation de Netflix influence ce que la plupart des utilisateurs regardent depuis au moins 2013 [112]. Les ventes d'Amazon ont augmenté de 29% après qu'ils aient intégré un système de recommandation [61]. Facebook utilise plusieurs systèmes de recommandation, leur « News Feed » en est un et leur « App Center » a été conçu autour d'un autre, avec la personnalisation des contenus en tête [2].

Il y deux raisons principales pour lesquelles beaucoup de sites web comportent un système de recommandation. Premièrement, un tel système opère du filtrage d'informations, permettant ainsi aux visiteurs du site qu'il accompagne de trouver des contenus pertinents plus rapidement. C'est la partie attrayante pour les utilisateurs du système de recommandation. Deuxièmement, un tel système augmente l'implication des utilisateurs à travers une expérience utilisateur personnalisées du site compagnon. C'est la partie attrayante pour les éditeurs de site web.

D'une manière générale, on définit la tâche d'un système de recommandation ainsi : le système doit sélectionner automatiquement une poignée d'items parmi le très grand nombre d'items disponibles. Cette sélection, destinée à un utilisateur en particulier, est faite d'après la pertinence prédite des items sélectionnés aux yeux de cet utilisateur, en se fondant sur ses préférences. La prédiction du niveau de pertinence des items dépend de la manière d'inférer les préférences des utilisateurs ainsi que de la technique de recommandation utilisée.

La catégorisation acceptée des techniques de recommandation [93] identifie six

types : le filtrage collaboratif, les techniques fondées sur le contenu, les techniques démographiques, les techniques fondées sur la connaissance, les techniques sociales et les techniques hybrides. De nos jours, la majorité des systèmes utilisent soit le filtrage collaboratif, soit des techniques fondées sur le contenu, soit des techniques hybrides. Puisque cette thèse concerne les systèmes de filtrage collaboratif (FC), nous donnons plus de détails sur ce type de techniques. Le FC déduit les préférences des utilisateurs de leur historique d'interactions avec le système, puis il fonde ses recommandations sur les préférences d'utilisateurs aux intérêts similaires.

D'une part, il est évident que les systèmes de recommandation sont utiles pour les utilisateurs comme pour les fournisseurs de contenus. Mais de l'autre, ces systèmes représentent potentiellement une menace pour la vie privée de leurs utilisateurs. Nous allons voir que cette menace est inhérente aux systèmes de recommandation, quelle que soit leur échelle, et qu'elle est exacerbée par l'omniprésence de tels systèmes.

Tout système de recommandation représente une menace potentielle pour la vie privée de ses utilisateurs car il agrège des données sur les préférences de chaque utilisateur. Ces données sont personnelles et devrait rester confidentielles car elles capturent les centres d'intérêts de l'utilisateur correspondant. Il est évident que les préférences en termes de religion, de sexualité ou de santé physique sont des données personnelles et sensibles en elles-mêmes. De manière moins évidente, les préférences concernant n'importe quel sujet sont aussi des données personnelles. Même les préférences apparemment les plus inoffensives telles que pour les blagues ou les jeux sont des données personnelles parce qu'elles servent de quasi-identifiant. Un quasi-identifiant [113] est une caractéristique d'un individu qui, prise indépendamment, ne l'identifie pas de manière unique, mais qui peut constituer un identifiant unique quand elle est combinée à d'autres quasi-identifiants ou à des informations publiques. Ainsi, la probabilité que l'ensemble des préférences d'un utilisateur, son profil, constitue un identifiant unique augmente d'autant plus qu'il y ajoute des préférences supplémentaires. De plus, il est irréalisable en général de garantir la non-unicité de n'importe quel sous-ensemble d'un profil s'il venait à être révélé car un profil typique contient un trop grand nombre de préférences. Par exemple, même l'anonymisation par la généralisation, tel que la conversion de notes sur 5 étoiles en notes binaires (j'aime ou je n'aime pas) n'est pas suffisant pour rendre tout profil non-unique. Un exemple d'attaque utilisant des quasi-identifiants est la célèbre réidentification d'utilisateurs de Netflix, censés avoir été rendus anonymes, par Narayanan et Shmatikov [81]. Ces derniers utilisent un jeu de données contenant des notes de films par des utilisateurs de Netflix anonymisés ainsi que des informations auxiliaires provenant de l'Internet Movie Database, afin de réidentifier certains utilisateurs tout en découvrant leurs préférences politiques.

De plus, l'omniprésence des systèmes de recommandation exacerbe cette menace inhérente pour la vie privée qu'ils entraînent. Cela veut dire que les individus laissent quelques informations personnelles sur chaque site web ou service en ligne disposant un système de recommandation, qu'ils utilisent. Que les services en ligne utilisent leur propre système ou un système tiers, cela résulte en des risques accrus pour la vie privée des individus.

Un système de recommandation est une menace pour la vie privée de ses utilisateurs quelle que soit son échelle. Quand la majorité des services en ligne mettent

en œuvre leur propre système de recommandation, cela crée une multitude de systèmes à petite échelle, dont certains peuvent être mal sécurisés ou contenir des bugs entrainant des risques d'atteinte à la confidentialité. Quand la majorité des services en lignes utilisent un petit nombre de fournisseurs tiers de système de recommandation, cela crée quelques systèmes à large échelle qui sont probablement correctement sécurisés mais plus les données personnelles sont concentrées chez un nombre réduit de fournisseurs, plus l'attractivité de chacun augmente aux yeux des adversaires.

Maintenant que le lecteur est convaincu que les systèmes de recommandation représentent une menace pour la vie privée de leurs utilisateurs, nous allons voir que la menace peut prendre de multiples formes, dont certaines sont convenues alors que d'autres sont plus insidieuses. Nous présentons d'abord les menaces (ou attaques) issues des adversaires de type « Big Brother » dans les systèmes centralisés, puis de ceux que nous appelons « Little Brothers » dans les systèmes décentralisés. Ensuite, nous présentons certaines autres attaques concernant les systèmes de recommandation, indépendamment de leur architecture.

Premièrement, nous observons que virtuellement tout système de recommandation déployé de nos jours est centralisé. Cela nous indique que la vie privée des utilisateurs de ces systèmes est menacée par au moins les adversaires de type « Big Brother ». Parce que l'opérateur d'un système de recommandation centralisé contrôle les données des utilisateurs et/ou les algorithmes de recommandation, les utilisateurs n'ont aucun moyen de vérifier facilement que l'opérateur opère son système comme il le prétend, notamment concernant de potentiels mécanismes de protection de la vie privée. De plus, même si l'opérateur est honnête et met en œuvre de tels mécanismes, il peut être contraint de les désactiver ou de les affaiblir par des entités telles que les services de renseignement de son pays d'appartenance. En résumé, les utilisateurs de systèmes de recommandation centralisés doivent faire aveuglément confiance aux opérateurs ou arrêter complètement de les utiliser s'ils tiennent à leur vie privée.

En conséquence, une approche possible pour éviter les menaces de type « Big Brother » est d'utiliser des systèmes décentralisés. La déconcentration des données parmi les pairs rend la réutilisation des données personnelles pour des fins autres que la recommandation plus dure pour les fournisseurs de contenus. Les techniques de filtrage collaboratif (FC), et particulièrement celles à représentation en mémoire, se prêtent bien aux architectures décentralisées car elles génèrent des recommandations en utilisant les informations locales à l'utilisateur courant.

Les systèmes de FC pair-à-pair passent particulièrement bien à l'échelle et ont donc été proposé comme une manière de résoudre les problèmes de vie privée et de passage à l'échelle des systèmes centralisés. La distribution des calculs entre les pairs permet de générer des recommandations sans nécessiter d'énormes serveurs.

Bien que les adversaires de type « Big Brother » n'existent pas dans les systèmes décentralisés, faire du filtrage collaboratif de telle manière ne résout pas tous les problèmes de vie privée pour les utilisateurs. En effet, ces derniers doivent coopérer entre eux pour exécuter l'algorithme de FC, mais faire confiance par défaut à tous les autres utilisateurs serait une erreur car n'importe qui peut participer à un système décentralisé, y compris des utilisateurs mal intentionnés, que nous appellerons des « Little Brothers ». Il est donc nécessaire d'adapter ces algorithmes pour atteindre un

compromis délicat entre coopération avec autrui, et freiner le partage d'informations pour ne pas divulguer ses données personnelles.

Les principaux éléments du filtrage collaboratif décentralisé à représentation en mémoire qui requièrent une attention spécifique pour préserver la vie privée d'un utilisateur sont son profil et ses voisins, ainsi que, dans certains types de systèmes, les items qu'il transmet. Naturellement, il faut révéler le moins possible du profil d'un utilisateur puisqu'il capture ses intérêts. Ensuite, les voisins d'un utilisateur ne devraient pas être librement disponibles à tout autre utilisateur car, lorsque les voisins d'un utilisateur sont sélectionnés selon la similarité de leurs profils, savoir qui est similaire à l'utilisateur cible permet à l'adversaire d'apprendre indirectement le profil de la cible. Enfin, dans certains systèmes pair-à-pair, la diffusion des items incombe aux pairs. Ils se transmettent alors les uns les autres les items qu'ils aiment, ce qui ne peut être fait sans mécanisme de protection sinon un adversaire surveillant quels items sont transmis par un pair pourrait apprendre des morceaux du profil de la cible.

Hide & Share, notre première contribution, protège la vie privée des utilisateurs de systèmes de filtrage collaboratif pair-à-pair en évitant la divulgation, directe ou indirecte via les voisins, de leur profil.

Cependant, les systèmes de recommandation, y compris ceux utilisant le filtrage collaboratif, sont sujets à encore d'autres menaces potentielles pour la vie privée de leurs utilisateurs.

Afin de souligner la diversité des types d'attaques restants qui ciblent les données personnelles des utilisateurs, quelle que soit l'architecture du système de recommandation, nous présentons les types d'attaques suivants : les exploitations de failles de sécurité, les attaques passives utilisant les listes d'items liés et les attaques « Sybil » avec informations auxiliaires.

Premièrement, quel que soit le but d'un système, il peut être victime de failles de sécurité causées par des facteurs humains, des exploitations de bug, des configurations incorrectes, *etc.* Tout adversaire, qu'il soit opérateur, utilisateur, ou un tiers sans rôle dans le système de recommandation, peut illégalement obtenir des données personnelles telles que les profils d'utilisateurs en exploitant un bug dans un composant de base du système comme une bibliothèque TLS. Ce type d'attaque affecte les systèmes centralisés comme décentralisés, bien qu'il soit plus efficace dans les premiers car l'adversaire peut accéder à plus de données en une seule occurrence de l'attaque.

Deuxièmement, il est un type d'attaque passive particulièrement insidieux qui exploite des données publiques de certains systèmes de filtrage collaboratif (FC), qu'ils soient centralisés ou décentralisés. Calandrino *et al.* introduisent dans [21] ce type d'attaque exploitant pour chaque item, la liste publique d'items liés à l'item courant. L'attaque tire profit de ces listes d'items liés et d'informations auxiliaires sur l'utilisateur cible afin de deviner si des changements au cours du temps dans certaines listes d'items liés signifient que la cible a ajouté ou retiré des items dans son profil. Cette attaque est particulièrement insidieuse car l'adversaire peut l'effectuer sans interagir directement avec le système de FC. De plus, les seuls produits du système que cette attaque utilise sont les listes d'items liées de certains items. Or ces listes semblent à première vue inoffensives du point de vue de la vie privée des utilisateurs

puisqu'elles indiquent seulement que certains items sont corrélés.

Enfin, les auteurs de [21] introduisent également une attaque créant un nombre de fausses identités (les utilisateurs « Sybil ») afin d'extraire des informations d'un système de filtrage collaboratif, quelle que soit son architecture. Pour ce faire, l'adversaire créé k utilisateurs « Sybil » partageant le même profil qui doit être un sous-ensemble de celui de l'utilisateur cible. L'adversaire obtient une partie du profil de la cible par des canaux auxiliaires tels que des avis de produits sur Amazon. Puis, si les « Sybils » sont suffisamment similaires à la cible, le système choisira pour chaque « Sybil » une liste de k plus proches voisins ou KNN (de l'anglais K-Nearest-Neighbors), composée des $k-1$ autres « Sybils », puisqu'ils ont tous un profil identique, et de la cible. Ainsi, quand le système génère une recommandation pour un « Sybil », l'item recommandé ne peut que venir du profil de la cible car, premièrement, le type de système considéré ne recommande pas un item déjà noté et, deuxièmement, ces systèmes puisent les items candidats dans le profil des voisins.

Étant donné le large éventail d'attaques possibles contre la vie privée des utilisateurs des systèmes de recommandation, nous ne pouvons pas toutes les traiter dans cette thèse. Ainsi, $2-step$, la deuxième contribution de cette thèse se concentre sur ce dernier type d'attaque « Sybil » appliquée au filtrage collaboratif à représentation en mémoire. Une raison supplémentaire de ce choix est que ce type d'attaque n'est pas énormément étudié dans la littérature.

Dans cette thèse, nous nous concentrons sur des aspects de vie privée pour les utilisateurs de systèmes de filtrage collaboratif (FC), et notamment les systèmes utilisant une représentation en mémoire pour plusieurs raisons.

Premièrement, le FC est le type de techniques de recommandation le plus populaire de nos jours. Elles permettent de recommander n'importe quel contenu, leurs recommandations peuvent être sérendipiteuses et elles offrent la plus grande précision des notes prédites. Ces techniques peuvent être divisées en deux sous-ensembles : celles à représentation par modèle et celles à représentation en mémoire. Bien que les techniques de FC à représentation par modèle ont la plus haute précision de prédiction, les techniques de FC à représentation en mémoire sont plus simples à mettre en œuvre, à régler précisément et elles peuvent facilement fournir des explications pour les items recommandés, ce qui contribue plus à la satisfaction des utilisateurs qu'une précision optimale. De plus, le FC à représentation en mémoire est particulièrement adapté aux architectures décentralisées, ce qui est une bonne caractéristique pour la vie privée des utilisateurs car cela retire par conception la menace des adversaires « Big Brother ».

Deuxièmement, les systèmes de FC constituent une cible privilégiée pour les adversaires qui cherchent à obtenir des données personnelles d'utilisateurs. Mathématiquement, puisque le FC est la technique la plus utilisée dans les systèmes commerciaux, ce genre de systèmes est le choix le plus efficace pour les adversaires qui nous concernent. De plus, l'aspect collaboratif du FC le rend tout particulièrement attractif pour ces adversaires. En effet, parce que le fondement même du FC consiste en l'exploitation des préférences de certains utilisateurs afin de fournir des recommandations à un autre utilisateur, cela signifie que beaucoup de parties du processus de FC peuvent potentiellement divulguer des données personnelles d'utilisateurs. Comparativement, le fait qu'un utilisateur A se voit recommandé l'item i par un système

fondé sur le contenu lui indique seulement que les mots-clefs ou concepts extraits par le système de la description de i correspondent à ses intérêts. Alors que si on considère le même fait mais dans un système de FC, l'utilisateur A peut potentiellement déduire des informations à propos d'autres utilisateurs car un tel système ne peut pas recommander un item à propos duquel personne n'a exprimé d'opinion.

A.2 Contributions

A.2.1 *Hide & Share*

Notre première contribution est « *Hide & Share* » (*H&S*), un nouveau mécanisme de calcul de similarité qui offre un niveau raisonnable de protection du profil des utilisateurs de systèmes de filtrage collaboratif pair-à-pair. Il est conçu pour nécessiter peu de puissance de calcul afin d'être appliqué aux contextes demandant un grand nombre de calculs de similarité tels que les algorithmes de recherche de KNN pair-à-pair. *H&S* permet aux utilisateurs de trouver leurs KNN sans devoir partager leur profil avec quiconque. *H&S* s'appuie sur une observation simple : les applications de KNN centrées sur l'utilisateur, telles que la recommandation, n'ont pas besoin de connaissances parfaites. Ceci permet à *H&S* d'offrir un gain significatif en protection de la vie privée, contre une réduction minime de la précision des calculs de similarité.

La contribution clef de *H&S* est une nouvelle technique d'approximation par les « landmarks », des profils générés aléatoirement et de manière équitable. Notre solution permet à deux utilisateurs de mesurer indirectement leur similarité en comparant leurs propres profils avec un ensemble de profils générés aléatoirement. La similarité entre le profil d'un utilisateur et un « landmark » est telle une coordonnée dans un système de coordonnées. Les utilisateurs échangent ensuite ces vecteurs de coordonnées, puis calculent une approximation de leur vraie similarité. Cela préserve la vie privée des utilisateurs puisqu'ils n'échangent pas leur profil complet et puisque les vecteurs de coordonnées ne révèlent qu'une petite quantité d'informations sur l'utilisateur.

Nous utilisons trois traces réelles pour évaluer *H&S* en termes de qualité de recommandation, de coûts supplémentaires et de protection empirique de la vie privée. Nous démontrons aussi formellement ses guaranties de protection de la vie privée en calculant une borne supérieure sur la quantité d'informations divulguées. Nos résultats montrent que les KNN générés par *H&S* fournissent un compromis raisonnable entre vie privée et utilité. De plus, *H&S* perturbe les valeurs de similarité, ce qui empêche les adversaires semi-curieux de faire une attaque de reconstruction de profil, mais sans diminuer significativement la qualité des recommandations.

A.2.2 $2-step$

Dans cette thèse, nous considérons la même attaque « Sybil » que dans [21] et montrons que sa performance dépend fortement de la population des utilisateurs, et de la mesure de similarité du système. Nos résultats obtenus avec un cadriciel à la pointe et sur trois jeux de données réelles, montrent une forte corrélation entre la capacité d'une mesure à offrir de bonnes recommandations et sa vulnérabilité à

l'attaque. Puis, nous proposons notre seconde contribution : la mesure de similarité composite $2-step$. Appliquée aux systèmes de filtrage collaboratif à représentation en mémoire, elle sert de contre-mesure à l'attaque sus-citée tout en continuant à fournir de bonnes recommandations aux utilisateurs légitimes.

Nous observons que *Cos-overlap*, une variante de la similarité cosinus, réduit significativement le taux de succès de l'attaque. Cette mesure de similarité considère que deux utilisateurs sont parfaitement similaires dès qu'ils ont donné la même note aux items en commun dans leurs profils. Ce faible pouvoir discriminant rend difficile pour les « Sybils » de distinguer la cible et les autres « Sybils » des alter egos parfaitement similaires à la cible.

Alors que la propriété précédente constitue un atout pour la résistance aux « Sybils », elle entrave clairement la capacité du système de fournir de bonnes recommandations. Considérons un utilisateur A et deux alter egos parfaitement similaires B et C. A et B partagent la même note pour le seul item commun à leurs profils. En revanche, A et C partagent des notes identiques pour plusieurs items communs. Clairement, C ferait un meilleur candidat que B pour fournir des recommandations à A. Mais *Cos-overlap* considèrera B et C comme également bons.

Cette observation nous permet de proposer $2-step$, une nouvelle mesure composite combinant la qualité de recommandation de la similarité cosinus, avec la résistance à l'attaque de *Cos-overlap*. Nos résultats avec $2-step$ combinent un bon score $RMSE$ avec un taux de succès très faible pour les attaquants « Sybils ».

Soit un utilisateur U et un voisin potentiel N, la première étape emploie une bonne mesure de similarité standard (cosinus par exemple), puis modifie la valeur résultante de manière à ce que les voisins potentiels ayant une similarité standard au-dessus d'un certain seuil apparaissent identiques pour U. À la différence de *Cos-overlap*, l'utilisation d'un seuil permet de regrouper les utilisateurs qui fourniront probablement des recommandations de qualités similaires, ce qui rend ardu à un « Sybil » d'obtenir un voisinage contenant l'utilisateur cible.

La seconde étape de la mesure va au-delà du seuil et tente de distinguer parmi les meilleurs voisins potentiels (ceux dont la similarité standard dépasse le seuil) lesquels seraient les plus utiles pour générer des recommandations pour U. Le processus de recommandation consistant à trouver des items pertinents dans le profil des voisins de U, cela implique qu'un voisin qui n'a pas d'items n'apparaissant pas dans le profil de U n'apporte rien. Au contraire, un bon voisin devrait avoir au moins quelques items n'apparaissant pas dans le profil de U. Cependant, il ne devrait pas en avoir trop : les utilisateurs avec de trop grands profils ont tendance à fournir des suggestions moins précises.

La seconde étape différencie donc les voisins potentiels qui dépassent le seuil en tenant compte du nombre d'items dans leur profil n'apparaissant pas dans celui de U. Puisque le système génère les voisinages de tous les utilisateurs, « Sybils » inclus, cette heuristique a l'effet bénéfique de décourager la présence d'autres « Sybils » dans le voisinage d'un « Sybil », rendant ainsi l'attaque encore plus difficile.

En somme, le seuil rend difficile pour un « Sybil » de différencier la cible ou un autre « Sybil » d'autres utilisateurs très similaires. La seconde étape complémente cette caractéristique en préférant les utilisateurs légitimes aux « Sybils ».

A.3 Conclusion

En conclusion, nous avons présenté deux contributions dans cette thèse. Chaque contribution vise à contrecarrer un type différent d'attaque contre la vie privée des utilisateurs de systèmes de recommandation utilisant le filtrage collaboratif.

Premièrement, nous avons proposé « *Hide & Share* » (*H&S*) comme une solution contre les adversaires de type « Little Brothers » dans les systèmes pair-à-pair. *H&S* permet à deux utilisateurs de calculer une approximation de leur similarité sans qu'ils doivent se révéler mutuellement leurs profils. *H&S* rend cela possible en calculant indirectement la similarité de deux utilisateurs, en comparant chaque profil à un ensemble de « landmarks », des profils générés aléatoirement.

Deuxièmement, nous avons proposé 2−*step* comme une solution contre les attaques « Sybil » à base d'informations auxiliaires visant les systèmes de FC, quelle que soit leur architecture. 2−*step* est une mesure de similarité composite réduisant significativement l'efficacité d'une telle attaque, tout en conservant une bonne qualité de recommandation. 2−*step* fonctionne en deux temps. Cette mesure commence par modifier la similarité de tous les utilisateurs au-dessus d'un certain seuil pour les rendre identiques en termes de similarité. Puis elle applique une seconde modification de la similarité des utilisateurs précédemment concernés afin de mettre en valeur ceux qui seraient les plus susceptibles d'améliorer la qualité des recommandations.

Mais il reste d'autres types d'attaque contre la vie privée des utilisateurs de systèmes de recommandation que nous n'avons pas traité dans cette thèse. Ces autres attaques, bien que déjà prisent en compte dans la littérature existante, méritent d'être plus étudiées pour que des solutions pratiques et efficaces soient déployées.

Enfin, les contributions que nous avons proposées peuvent encore être améliorées. Il serait intéressant d'étudier l'applicabilité de 2−*step*, et les adaptations éventuellement requises, aux techniques de FC à représentation par modèle. Une autre piste de recherche serait d'évaluer si *H&S* a des propriétés intrinsèques de résistance aux mêmes attaques « Sybil » dont s'occupe 2−*step*. Une piste également intéressante est : comment évolue le protection, empirique et théorique, offerte par *H&S* quand on considère un adversaire capable de collusion avec d'autres utilisateurs ?

Appendix B

Publications

The contributions in this thesis led to the following publications:

- "Hide & Share: Landmark-based Similarity for Private KNN Computation" by *A. Boutet, D. Frey, R. Guerraoui, A.-M. Kermarrec, A. Rault, F. Taïani, and J. Wang*, published in IEEE/IFIP International Conference on Dependable Systems and Networks 2015 [42].
- "Collaborative Filtering Under a Sybil Attack: Analysis of a Privacy Threat" by *D. Frey, R. Guerraoui, A.-M. Kermarrec, and A. Rault*, published in European Workshop on Systems Security 2015 [41].

Appendix C

Bibliography

[1] Mahout. `https://mahout.apache.org/`.

[2] Under the Hood: Building the App Center recommendation engine, October 2012.

[3] Charu C. Aggarwal. On K-anonymity and the Curse of Dimensionality. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 901–909, Trondheim, Norway, 2005. VLDB Endowment.

[4] Kamal Ali and Wijnand van Stam. TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 394–401, New York, NY, USA, 2004. ACM.

[5] Marko Balabanović and Yoav Shoham. Fab: Content-based, Collaborative Recommendation. *Commun. ACM*, 40(3):66–72, March 1997.

[6] Ranieri Baraglia, Patrizio Dazzi, Matteo Mordacchini, and Laura Ricci. A peer-to-peer recommender system for self-emerging user communities based on gossip overlays. *Journal of Computer and System Sciences*, 79(2):291–308, 2013.

[7] Arnaud Berlioz, Arik Friedman, Mohamed Ali Kaafar, Roksana Boreli, and Shlomo Berkovsky. Applying Differential Privacy to Matrix Factorization. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 107–114, New York, NY, USA, 2015. ACM.

[8] Daniel Julius Bernstein. NaCl: Networking and cryptography library.

[9] Marin Bertier, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. The gossple anonymous social network. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, pages 191–211. Springer-Verlag, 2010.

[10] Marin Bertier, Rachid Guerraoui, Vincent Leroy, and Anne-Marie Kermarrec. Toward personalized query expansion. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 7–12. ACM, 2009.

[11] Smriti Bhagat, Udi Weinsberg, Stratis Ioannidis, and Nina Taft. Recommending with an agenda: Active learning of private attributes using matrix factorization. In *RecSys*. ACM, 2014.

[12] Daniel Billsus, Michael J. Pazzani, and James Chen. A Learning Agent for Wireless News Access. In *Proceedings of the 5th International Conference on Intelligent User Interfaces*, IUI '00, pages 33–36, New York, NY, USA, 2000. ACM.

[13] Simon Blake-Wilson and Alfred Menezes. Authenticated diffe-hellman key agreement protocols. In *Selected Areas in Cryptography*, number 1556, pages 339–361. Springer Berlin Heidelberg, 1999.

[14] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[15] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM Special Interest Group on Algorithms and Computation Theory News*, 15(1):23–27, 1983.

[16] Antoine Boutet, Davide Frey, Rachid Guerraoui, Arnaud Jégou, and Anne-Marie Kermarrec. WHATSUP: A decentralized instant news recommender. In *Parallel and Distributed Processing Symposium, International*, pages 741–752. IEEE Computer Society, 2013.

[17] Antoine Boutet, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Rhicheek Patra. HyRec: Leveraging Browsers for Scalable Recommenders. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, pages 85–96, New York, NY, USA, 2014. ACM.

[18] Antoine Boutet, Davide Frey, Arnaud Jégou, Anne-Marie Kermarrec, and Heverson B. Ribeiro. Freerec: An anonymous and distributed personalization architecture. In *The First International Conference on Networked Systems*, pages 58–73. Springer Berlin Heidelberg, 2013.

[19] John S. Breese, David Heckerman, and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, Madison, WI, USA, July 1998. Morgan Kaufmann.

[20] Brian Morrissey. How the Banner Ad Was Born, April 2013.

[21] J.A. Calandrino, A. Kilzer, A. Narayanan, E.W. Felten, and V. Shmatikov. "You Might Also Like:" Privacy Risks of Collaborative Filtering. In *2011 IEEE Symposium on Security and Privacy (SP)*, pages 231–246, May 2011.

[22] J. Canny. Collaborative filtering with privacy via factor analysis. In *SIGIR*, 2002.

[23] John Canny. Collaborative filtering with privacy. In *SP*. IEEE, 2002.

[24] C. Castelluccia, M.-A. Kaafar, and M.-D. Tran. Betrayed by your ads! In *Privacy Enhancing Technologies*, volume 7384 of *Lecture Notes in Computer Science*. Springer, 2012.

[25] Sonny Han Seng Chee, Jiawei Han, and Ke Wang. RecTree: An Efficient Collaborative Filtering Method. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Data Warehousing and Knowledge Discovery*, number 2114 in Lecture Notes in Computer Science, pages 141–151. Springer Berlin Heidelberg, September 2001. DOI: 10.1007/3-540-44801-2_15.

[26] Rui Chen, Min Xie, and Laks V. S. Lakshmanan. Thwarting passive privacy attacks in collaborative filtering. In *Database Systems for Advanced Applications*, number 8422, pages 218–233. Springer International Publishing, 2014.

[27] R Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 364–369. ACM, 1986.

[28] William W. Cohen. Fast Effective Rule Induction. In *Machine Learning Proceedings 1995*, pages 115–123. Morgan Kaufmann, San Francisco (CA), 1995.

[29] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12):94–101, 2009.

[30] Cyphort Labs. The Rise of Malvertising. Cyphort Labs Special Report, 2015.

[31] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, 2007.

[32] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning*, 3:993–1022, 2003.

[33] Mukund Deshpande and George Karypis. Item-based top-N Recommendation Algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, January 2004.

[34] Tommaso Di Noia, Roberto Mirizzi, Vito Claudio Ostuni, Davide Romito, and Markus Zanker. Linked open data to support content-based recommender systems. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 1–8, New York, NY, USA, 2012. ACM.

[35] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems An Introduciton*. Cambridge University Press, November 2010.

[36] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag.

[37] M.D. Ekstrand, J.T. Riedl, and J.A. Konstan. *Collaborative Filtering Recommender Systems*. Now Publishers, 2011.

[38] A.M. Elmisery and D. Botvich. Private recommendation service for IPTV systems: Protecting user profile privacy. In *2011 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 571–577, 2011.

[39] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[40] Julien Freudiger, Nevena Vratonjic, and Jean-Pierre Hubaux. Towards privacy-friendly online advertising. In *IEEE Web 2.0 Security and Privacy (W2SP)*, 2009.

[41] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Antoine Rault. Collaborative filtering under a sybil attack: Analysis of a privacy threat. In *EuroSec*. ACM, 2015.

[42] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Antoine Rault, Francois Taiani, and Jingjing Wang. Hide & Share: Landmark-Based Similarity for Private KNN Computation. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 263–274, June 2015.

[43] Davide Frey, Arnaud Jégou, and Anne-Marie Kermarrec. Social market: Combining explicit and implicit social networks. In *Stabilization, Safety, and Security of Distributed Systems*, number 6976, pages 193–207. Springer Berlin Heidelberg, 2011.

[44] Robert Garfinkel, Ram D. Gopal, Bhavik K. Pathak, Rajkumar Venkatesan, and Fang Yin. Empirical Analysis of the Business Value of Recommender Systems. SSRN Scholarly Paper ID 958770, Social Science Research Network, Rochester, NY, November 2006.

[45] David Goldberg, David A. Nichols, Brian M. Oki, and Douglas B. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[46] Ken Goldberg, Theresa Roeder, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4:133–151, 2001.

[47] Carlos A. Gomez-Uribe and Neil Hunt. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4):13:1–13:19, 2015.

[48] Rachid Guerraoui, Anne-Marie Kermarrec, Rhicheek Patra, and Mahsa Taziki. D2p: distance-based differential privacy in recommenders. *Proceedings of the VLDB Endowment*, 8(8):862–873, 2015.

[49] Ihsan Gunes, Alper Bilge, and Huseyin Polat. Shilling attacks against memory-based privacy-preserving recommendation algorithms. *TIIS*, 7(5):1272–1290, 2013.

[50] Guy Shani, David Heckerman, and Ronen I. Brafman. An MDP-Based Recommender System. *Journal of Machine Learning Research*, 6:1265–1295, September 2005.

[51] Peng Han, Bo Xie, Fan Yang, and Ruimin Shen. A scalable p2p recommender system based on distributed collaborative filtering. *Expert Systems with Applications*, 27(2):203 – 210, 2004.

[52] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 230–237. ACM, 1999.

[53] T. Ryan Hoens, Marina Blanton, Aaron Steele, and Nitesh V. Chawla. Reliable medical recommendation systems with patient privacy. *ACM Transactions on Intelligent Systems and Technology*, 4(4):67:1–67:31, 2013.

[54] Thomas Hofmann. Latent Semantic Models for Collaborative Filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, January 2004.

[55] Zhengli Huang, Wenliang Du, and Biao Chen. Deriving private information from randomized data. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, page 37–48. ACM, 2005.

[56] Sibren Isaacman, Stratis Ioannidis, Augustin Chaintreau, and Margaret Martonosi. Distributed rating prediction in user generated content streams. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 69–76, New York, NY, USA, 2011. ACM.

[57] Markus Jakobsson and Ari Juels. Proofs of Work and Bread Pudding Protocols. In *Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), September 20-21, 1999, Leuven, Belgium*, pages 258–272, 1999.

[58] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25, August 2007.

[59] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321–2339, 2009.

[60] Michael J. Shaw Hsin-Lu Chang Matthew Nelson Jong Woo Kim, Byung Hun Lee. Application of Decision-Tree Induction Techniques to Personalized Advertisements on Internet Storefronts. *International Journal of Electronic Commerce*, 5(3):45–62, March 2001.

[61] JP Mangalidan. Amazon's recommendation secret. *Fortune*, July 2012.

[62] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Third IEEE International Conference on Data Mining*, pages 99–106. IEEE Computer Society, 2003.

[63] George Karypis. Evaluation of Item-Based Top-N Recommendation Algorithms. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, CIKM '01, pages 247–254, New York, NY, USA, 2001. ACM.

[64] Jae Kyeong Kim, Hyea Kyeong Kim, and Yoon Ho Cho. A user-oriented contents recommendation system in peer-to-peer architecture. *Expert Systems with Applications*, 34(1):300 – 312, 2008.

[65] Brendan Kitts, David Freed, and Martin Vrieze. Cross-sell: A Fast Promotion-tunable Customer-item Recommendation Method Based on Conditionally Independent Probabilities. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 437–446, New York, NY, USA, 2000. ACM.

[66] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[67] Yehuda Koren. Collaborative Filtering with Temporal Dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 447–456, New York, NY, USA, 2009. ACM.

[68] Ugur Kuter and Jennifer Golbeck. SUNNY: a new algorithm for trust inference in social networks using probabilistic confidence models. In *Proceedings of the 22d National Conference on Artificial Intelligence*, volume 2, page 1377–1382. AAAI Press, 2007.

[69] Pierre L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators, 1998.

[70] Friedrich Leisch. A toolbox for K-centroids cluster analysis. *Computational Statistics & Data Analysis*, 51(2):526–544, November 2006.

[71] David D. Lewis, Robert E. Schapire, James P. Callan, and Ron Papka. Training Algorithms for Linear Text Classifiers. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '96, pages 298–306, New York, NY, USA, 1996. ACM.

[72] Dongsheng Li, Qin Lv, Huanhuan Xia, Li Shang, Tun Lu, and Ning Gu. Pistis: A privacy-preserving content recommender system for online social communities. In *WI-IAT*. IEEE, 2011.

[73] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[74] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 17–24, New York, NY, USA, 2007. ACM.

[75] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[76] Frank McSherry and Ilya Mironov. Differentially Private Recommender Systems: Building Privacy into the Net. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 627–636, New York, NY, USA, 2009. ACM.

[77] Bamshad Mobasher, Robin Burke, and Jeff J. Sandvig. Model-based collaborative filtering as a defense against profile injection attacks. In *AAAI*, volume 6, page 1388, 2006.

[78] Cataldo Musto, Fedelucio Narducci, Pierpaolo Basile, Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Cross-Language Information Filtering: Word Sense Disambiguation vs. Distributional Models. In Roberto Pirrone and Filippo Sorbello, editors, *AI*IA 2011: Artificial Intelligence Around Man and Beyond*, number 6934 in Lecture Notes in Computer Science, pages 250–261. Springer Berlin Heidelberg, September 2011. DOI: 10.1007/978-3-642-23954-0_24.

[79] Cataldo Musto, Fedelucio Narducci, Pasquale Lops, Giovanni Semeraro, Marco de Gemmis, Mauro Barbieri, Jan Korst, Verus Pronk, and Ramon Clout. Enhanced Semantic TV-Show Representation for Personalized Electronic Program Guides. In Judith Masthoff, Bamshad Mobasher, Michel C. Desmarais, and Roger Nkambou, editors, *User Modeling, Adaptation, and Personalization*, number 7379 in Lecture Notes in Computer Science, pages 188–199. Springer Berlin Heidelberg, July 2012. DOI: 10.1007/978-3-642-31454-4_16.

[80] Moni Naor. Bit commitment using pseudo-randomness. *Journal of Cryptology*, 4:151–158, 1991.

[81] A. Narayanan and V. Shmatikov. Robust De-anonymization of Large Sparse Datasets. In *IEEE Symposium on Security and Privacy, 2008. SP 2008*, pages 111–125, 2008.

[82] Michael P. O'Mahony, Neil J. Hurley, and Guenole C. M. Silvestre. Recommender Systems: Attack Types and Strategies. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 334–339, 2005.

[83] Michael Pazzani and Daniel Billsus. Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning*, 27(3):313–331, June 1997.

[84] Michael J. Pazzani and Daniel Billsus. Content-Based Recommendation Systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, number 4321 in Lecture Notes in Computer Science, pages 325–341. Springer Berlin Heidelberg, 2007. DOI: 10.1007/978-3-540-72079-9_10.

[85] David Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-Based Approach. In *In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 473–480. Morgan Kaufmann, 2000.

[86] H. Polat and Wenliang Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *Third IEEE International Conference on Data Mining, 2003. ICDM 2003*, pages 625–628, November 2003.

[87] Huseyin Polat and Wenliang Du. Svd-based collaborative filtering with privacy. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, page 791–795. ACM, 2005.

[88] J.A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen, and H.J. Sips. Tribler: A social-based peer-to-peer system. Technical Report 2006-002 (presented at IPTPS'06), Delft University of Technology, Feb 2006. (presented at the 5th Int'l Workshop on Peer-to-Peer Systems (IPTPS)).

[89] PricewaterhouseCoopers. IAB internet advertising revenue report 2012 full year results. IAB internet advertising revenue report, April 2013.

[90] Anand Rajaraman and Jeffrey David Ullman. Data Mining. In *Mining of Massive Datasets*, pages 1–17. Cambridge University Press, 2011. Cambridge Books Online.

[91] Sahin Renckes, Huseyin Polat, and Yusuf Oysal. A new hybrid recommendation algorithm with privacy. *Expert Systems*, 29(1):39–55, 2012.

[92] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, page 175–186. ACM, 1994.

[93] Francesco Ricci, Lior Rokach, and Bracha Shapira, editors. *Recommender Systems Handbook*. Springer US, Boston, MA, 2nd edition edition, 2015.

[94] Cornelis Joost van Rijsbergen. *Information Retrieval*. Butterworths, 2 edition, 1979.

[95] Joseph John Rocchio. Relevance feedback in information retrieval. *The SMART System: Experiments in Automatic Document Processing*, pages 313–323, 1971.

[96] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted Boltzmann Machines for Collaborative Filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[97] Gerard Salton and Michael McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1983.

[98] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of Recommendation Algorithms for e-Commerce. In *Proceedings of the 2Nd ACM Conference on Electronic Commerce*, EC '00, pages 158–167, New York, NY, USA, 2000. ACM.

[99] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. ACM, 2000.

[100] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[101] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-Commerce Recommendation Applications. *Data Mining and Knowledge Discovery*, 5(1-2):115–153, January 2001.

[102] Carlos E. Seminario and David C. Wilson. Attacking item-based recommender systems with power items. In *RecSys*. ACM, 2014.

[103] Carlos E. Seminario and David C. Wilson. Nuke 'Em Till They Go: Investigating Power User Attacks to Disparage Items in Collaborative Recommenders. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 293–296, New York, NY, USA, 2015. ACM.

[104] Upendra Shardanand and Pattie Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[105] Simon Funk. Netflix Update: Try This at Home, December 2006.

[106] Stefanie Olsen. Amazon blushes over sex link gaffe, December 2002.

[107] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. 2009:e421425.

[108] Xiaoyuan Su and T.M. Khoshgoftaar. Collaborative Filtering for Multi-class Data Using Belief Nets Algorithms. In *18th IEEE International Conference on Tools with Artificial Intelligence, 2006. ICTAI '06*, pages 497–504, November 2006.

[109] M. Tada, H. Kikuchi, and S. Puntheeranurak. Privacy-Preserving Collaborative Filtering Protocol Based on Similarity between Items. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 573–578, April 2010.

[110] P. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Addison Wesley, 2005.

[111] Nava Tintarev. Explaining Recommendations. In Cristina Conati, Kathleen McCoy, and Georgios Paliouras, editors, *User Modeling 2007*, number 4511 in Lecture Notes in Computer Science, pages 470–474. Springer Berlin Heidelberg, July 2007. DOI: 10.1007/978-3-540-73078-1_67.

[112] Tom Vanderbilt. The Science Behind the Netflix Algorithms That Decide What You'll Watch Next. *WIRED*, August 2013.

[113] Tore Dalenius. Finding a Needle In a Haystack or Identifying Anonymous Census Records. *Journal of Official Statistics*, 2(3):329–336, 1986.

[114] Koen Verstrepen and Bart Goethals. Unifying Nearest Neighbors Collaborative Filtering. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 177–184, New York, NY, USA, 2014. ACM.

[115] Spyros Voulgaris and Maarten van Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par 2005 Parallel Processing*, number 3648, pages 1143–1152. Springer Berlin Heidelberg, 2005.

[116] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 501–508, New York, NY, USA, 2006. ACM.

[117] Udi Weinsberg, Smriti Bhagat, Stratis Ioannidis, and Nina Taft. BlurMe: Inferring and obfuscating user gender based on ratings. In *RecSys*. ACM, 2012.

[118] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, third edition, September 2007.

VU :

Le Directeur de Thèse
(Nom et Prénom)

VU :

Le Responsable de l'École Doctorale

VU pour autorisation de soutenance

Rennes, le

Le Président de l'Université de Rennes 1

David ALIS

VU après soutenance pour autorisation de publication :

Le Président de Jury,
(Nom et Prénom)