Received: 2 August 2021 | Revised: 10 November 2021 | Accepted: 27 November 2021

DOI: 10.1002/spy2.201

RESEARCH ARTICLE

WILEY

OpenNym: Privacy preserving recommending via pseudonymous group authentication

Alessandro Checco[1] | Lorenzo Bracciale[2] | Douglas J. Leith[3] | Giuseppe Bianchi[2]

[1]Information School, The University of Sheffield, Sheffield, UK

[2]Electronic Engineering Department, University of Rome "Tor Vergata", Rome, Italy

[3]School of Computer Science & Statistics, Trinity College Dublin, Dublin, Ireland

Correspondence
Alessandro Checco, Information School, The University of Sheffield, Sheffield, UK.
Email: a.checco@sheffield.ac.uk

Abstract

A user accessing an online recommender system typically has two choices: either agree to be uniquely identified and in return receive a personalized and rich experience, or try to use the service anonymously but receive a degraded non-personalized service. In this paper, we offer a third option to this "all or nothing" paradigm, namely use a web service with a public group identity, that we refer to as an OpenNym identity, which provides users with a degree of anonymity while still allowing useful personalization of the web service. Our approach can be implemented as a browser shim that is backward compatible with existing services and as an example, we demonstrate operation with the Movielens online service. We exploit the fact that users can often be clustered into groups having similar preferences and in this way, increased privacy need not come at the cost of degraded service. Indeed use of the OpenNym approach with Movielens *improves* personalization performance.

KEYWORDS

privacy, pseudonymous authentication, recommender systems

1 | INTRODUCTION

Recommender systems are becoming pervasive in online services, ranging from relatively simple "people who liked this also liked that" suggestions to personalized advertising, reranking of search results and decision support, for example, route recommendation, health diagnosis. Importantly, existing services and business models are largely based on an all or nothing user participation paradigm, namely users are encouraged to supply personally identifiable information (typically via use of a login account linked to their identity) in return for a personalized service. The, intentionally unattractive, alternatives offered are either to receive a degraded service when accessing without logging in or to simply not use the service.

The current online-tracking arms race between browsers and online advertisers exacerbates this ambivalence, as solutions like Google FLoC[1] will gradually reduce the possibility of indirect user behavior tracking, pushing web services to require forms of authentication for the simplest of functionalities. However, centralizing all user tracking and profile management could increase users' sensitivity toward the requirement of using login accounts linked to their identity.

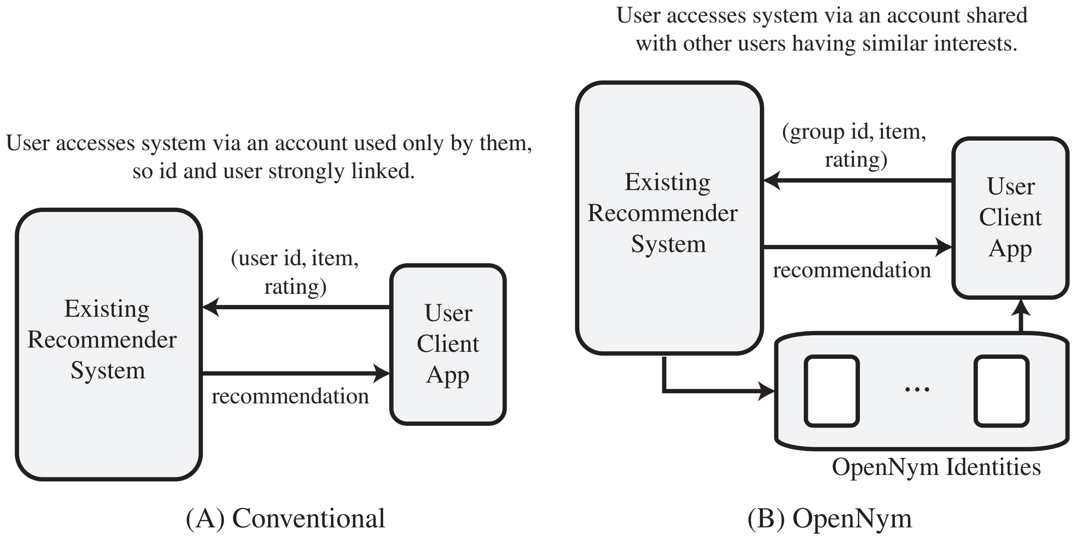

User accesses system via an account used only by them, so id and user strongly linked.
Existing Recommender System
(user id, item, rating)
recommendation
User Client App
(A) Conventional
User accesses system via an account shared with other users having similar interests.
Existing Recommender System
(group id, item, rating)
recommendation
User Client App
...
OpenNym Identities
(B) OpenNym

FIGURE 1 Overview of OpenNym approach

Our aim in this paper is to establish whether a usable middle ground exists whereby users gain good quality personalized recommendations while retaining a meaningful degree of privacy. We argue that a key requirement for practically useful approaches is support for incremental rollout and backward compatibility with existing online services that provide recommendations. We emphasize that while powerful cryptographic approaches to privacy have previously been proposed, these are essentially clean slate designs that are not compatible with existing services without major changes and so too easily dismissed/blocked by incumbents. Alternative approaches to privacy based on adding "noise" to user ratings are potentially backward compatible with existing systems and have been considered, for example, in the context of differential privacy,[2] but often the addition of noise incurs a significant degradation in predictive accuracy plus the nature of the privacy protection provided may be difficult for relatively unsophisticated users to fully understand.

The basic idea behind our approach is illustrated in Figure 1. Instead of users accessing the recommender system individually using an account linked to them, they access the system by selecting from a set of available group accounts, which we refer to as OpenNym identities (the nym terminology is motivated by the use of pseudonyms). Group accounts are shared by many users, so providing a form of "hiding in the crowd" indistinguishability that we will make crisp shortly. By only submitting ratings via a group account, the link between a user's ratings and their individual identity is much reduced. Provided the users sharing an OpenNym identity have similar preferences, a useful degree of personalization can still be provided.

In this paper, we first demonstrate that it is indeed possible to provide a useful degree of personalization when using group accounts with an existing production recommender service, namely Movielens. Secondly, we show how users can select in a private manner an OpenNym that is suitably aligned with their preferences. We do this by carrying out the selection locally within the user's browser via comparison of public OpenNym preference profiles with the user's private locally stored profile. While OpenNym identities with specified profiles can be created in an offline manner, for example, using demographic information, thirdly we show that they can also be learned in an online manner. Namely, we consider the repeated process of (a) users selecting the OpenNym identity most closely aligned with their preferences and submitting ratings using this and (b) adaptation of the recommendations for each OpenNym identity based on these submitted ratings. We show that under mild conditions this process is convergent and results in users being clustered according to their preferences. Importantly, we demonstrate that this can be achieved in a privacy-preserving manner while remaining backward compatible with existing systems: we show that an appropriate profile selection policy can guarantee convergence without (a) the need to access the online recommender system, and (b) the knowledge of the individual user identities nor their ratings by the online recommender system.

Even in the worst-case scenario where an attacker already knows the OpenNym identity associated to a user (eg, after a side-channel attack), the user can keep a so-called τ-hiding property,[3] keeping the adversary's maximum probabilistic belief that the user rated a given item lower than τ.

We implement a proof-of-concept prototype service based on these ideas. Users employ a browser plug-in to select an appropriate OpenNym identity for accessing the Movielens recommender system. Since the identity used for access is determined by the cookies supplied to the recommender system, the plugin essentially provides a cookie-management service, while the identity repository on a third-party server is able to perform the clustering and the update of the OpenNym identities. We demonstrate tools for private automated selection of the most appropriate identity and also support for incremental rollout by demonstrating operation with the existing, unmodified Movielens recommender service and user interface. Note that the Movielens service therefore includes a mix of existing individual users and users accessing the system via OpenNym identities, a key aspect of support for incremental rollout.

The rest of the paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we introduce the threat model. In Section 4, we introduce the family of systems OpenNym and discuss its architecture. In Section 5, we describe the specific profile association policy implemented in our prototype. In Section 6, we measure the recommending performance of our prototype on the Movielens web system. In Section 7, we analyze the privacy performance of the proposed system. In Section 8, we present the theoretical foundations on which OpenNym is based, and identify the family of association policies and loss functions under which convergence can be guaranteed. Finally in Section 9, we draw the conclusions.

2 | RELATED WORK

Early work on private recommender systems focused on anonymous trust authentication. Bussard et al.[4] uses a blind signature scheme to establish trust and anonymity in an online system while Quercia et al.[5] introduces a blinded threshold signature technique to achieve private trusted authentication. Crites and Lysyanskaya[6] propose the so-called mercurial signature to provide anonymous credentials by means of signatures that can be privately linked from one pseudonym to another.

Unfortunately these techniques, while robust to some attacks (eg, Sybil and shilling attacks), are not by themselves able to protect against de-anonymization attacks when the recommendation database is compromised. For example, Aggarwal[7] shows how high dimensional data can often easily be de-anonymized in the case of a leak and Narayanan and Shmatikov[8] demonstrated how the Netflix dataset could be de-anonymized using publicly available side information. More generally, Datta et al.[9] shows how large datasets can be easy to de-anonymize. This is arguably the main vulnerability of a recommender system, and this kind of attack constitutes the main threat considered in the present work (see Section 3).

There have been a number of previous studies on privacy in recommender systems, but none of them are able to provide at the same time backward compatibility, good prediction accuracy, and implementation simplicity. Many existing approaches are based on encryption and multi-party computation and are essentially not backward compatible with existing systems. Nikolaenko et al.[10] proposes a solution where a trusted third party (a cryptographic service provider) executes an encrypted version of the recommending task. Aimeur et al.[11] imagines a solution where the user data is split between the online system and a semi-trusted third party. Li et al.[12] provides a solution where content ratings and item similarity data are determined via distributed cryptographic multi-party computation, recommendations are then further personalized locally. In the special case of advertisement systems, Guha et al.[13] considers a web system that supplies a large set of possible adverts to the user which the user's browser then privately filters, locally making the decision about which advert to display. Other solutions propose privacy-preserving schemes for a collaborative filtering algorithm using synchronized cryptographic computation[14] or homomorphic encryption.[15-17] Another approach to implementing a private recommender system is to perturb ratings with noise, an approach that often suffers from performance loss. Initially proposed by Agrawal & Srikant[18] it is applied to collaborative filtering.[19-21]

Federated learning approaches innovate the field by training machine-learning models without exposing user's private data.[22] It has been extensively studied in the case of private matrix factorization.[23,24] Federated learning has also been proposed by Google Federated Learning of Cohorts (FLoC)[1] as an alternative to HTTP cookies and related tracking techniques.

Perhaps the closest piece of literature to our work is the one from Checco et al.,[25] where a private clustering-based recommender system is devised building on existing matrix factorization approaches, without the need for sophisticated cryptographic methods. The main idea is to average the user ratings inside a cluster and iteratively learn the optimal cluster choice. They show that often the increased privacy does not come at the cost of reduced recommendation accuracy. However, this work is limited to recommender systems that use a specific form of matrix factorization, and requires the recommender service to implement a series of changes to ensure compatibility. For this reason, the applicability to existing web systems has not been demonstrated. Our work builds on these ideas from a system perspective, extending the class of recommender systems considered (not just matrix factorization recommenders, but rather a vast class of recommenders satisfying very mild conditions) thus ensuring backward compatibility with existing web services. Moreover, we define the implementation details of the client-server architecture needed to guarantee backward compatibility, and show that the recommender system does not need to be aware of the presence of such a system, which can thus effectively piggyback on existing systems. Finally, our experiments on a live web system are an important step toward the implementation of such techniques in the real world.

3 | THREAT MODEL

The privacy disclosure threat we consider arises naturally from attacks which have already occurred on production recommender systems. A user interacts with the recommender system through an authentication-based system, sending ratings and receiving recommendations. The user ratings are stored in the recommender system database. The sensitive asset is the stream of unique IDs and ratings composing this database, which might be released either as a result of a hack/leak or intentionally by the recommender system operator[*] Thus, the attacker can be any actor that has access to this database, even the recommender system itself. We do not consider any type of online/active attack (eg, malicious recommender system/users or DoS attacks) focusing on the potential privacy issue resulting from leakage of the recommender database, as our solution is meant to mitigate these type of vulnerability. However, in Section 3.3, we briefly discuss the most relevant linking attacks for these type of systems. Motivations behind this kind of attacks are case-specific but potentially very damaging,[26] since recommender systems are employed in e-commerce, video streaming, dating applications, and other fields where user ratings can be considered a private asset.

3.1 | Attack definitions

We consider two main types of attack/threat:

Attack 1. The attacker seeks to learn the true user identity associated with a user account or ID. This can be of interest in its own right: even if the user has not disclosed any rating, the mere fact of being associated to a specific ID can have many different implications. Examples of such attacks are described in Reference 9,7,8.

Attack 2. The attacker knows already the association of a user with an ID and seeks to discover the user ratings.[†]

In conventional recommender systems each ID in the database is associated with a single user. In this case, Attack 1 might proceed by attempting to correlate the collection of (item, rating) pairs associated with each ID with known side information about a target user or users. When such an attack is successful then the ratings submitted by the user are simultaneously discovered, making Attack 2 automatically successful too.

When multiple users are associated with an ID (as with OpenNym) then the collection of (item, rating) pairs associated with that ID is now a mixture of ratings from multiple different users. Provided the users sharing an ID submit sufficiently different sets of (item, rating) pairs then this attack can be expected to involve a harder inference task than when an ID is used by a single user. This is because the (item, rating) pairs submitted by the other users sharing the ID now act as “noise” that tends to mask the pattern of (item, rating) pairs submitted by the user being targeted. Of course, if the users sharing an ID submit sets of rating which are too similar to each other then the protection against de-anonymization provided by this direct mixing mechanism may be reduced, in which case additional measures can be envisioned, for example, inserting dummy ratings or other “noise” to increase diversity.

Regarding Attack 2, de-anonymization attacks have extensively proven their ability in attacking also large data set.[27] Such attacks usually rely upon an extra knowledge (side-channel) which correlates with the allegedly anonymous data, to reveal precious information. Importantly, our approach can, at least in part, protect from this kind of attack: when more than one user is sharing an ID, then linking a user to the set of items rated is also significantly harder: an attacker can only have certainty that an item has been rated by a user when all users sharing the ID have rated that item. In general, the probability that a user rated an item can be taken as roughly proportional to the fraction of users employing the shared ID that have rated the item. Further, even when the attacker knows that a user has rated an item, the attacker will not be able to know the value of the user’s rating, unless all of the users employing the shared ID also gave exactly the same rating for that item. This form of "hiding in the crowd" indistinguishability/plausible deniability can be further strengthened by allowing users to submit dummy ratings, in which case users can additionally claim that a submitted rating linked to them was in fact dummy rather than real.

3.2 | Privacy metrics

We define a privacy metric for each attack, which will then be evaluated on a live recommender system in Section 7.

3.2.1 | Attack 1

Lacking additional side information the attacker can try to guess the correct nym by exploiting the fact that some nyms might contain more users than others, namely the nym with most users is the best guess for the attacker. In the worst case for the attacker (when all nyms have the same number of users) the user has an indistinguishability probability of $1/p$, where p is the number of nyms, that is, we have a form of "hiding in the crowd" anonymity.

In general, the probability that a user is using a certain nym is equal to the number of users in the that nym divided by the total number of users (equivalent to anonymity set size[3]), so the best guess for the attacker is to choose the nym with the largest number of users.

3.2.2 | Attack 2

In the worst case scenario, knowing for instance all the ratings of an user but one, an attacker can at most calculate the nym associated with a user (eg, pretending to be the user and letting the algorithm choose the right nym which will be probably the same nym of the targeted user). We then consider this worst case scenario, where an attacker seeks to learn which items a user has rated given knowledge of the nym to which the user belongs.

We define the following privacy measure:

$$P(u, v \mid u \text{ is associated to nym } a) \triangleq p(u, v) = \frac{\mid \mathcal{O}_{av} \mid}{\mid \mathcal{U}(a) \mid},$$

where a is the nym index, v the item index, u is a given user, $\mathcal{U}(a)$ is the set of users sharing the nym a, and $\mathcal{O}_{av}$ the set of users selecting account a and rating item v. We refer to $p(u, v)$ as the *association probability* and it is an estimate of the probability that a user rated movie v, given that the user chose nym a. Hence, smaller values of $p(u, v)$ correspond to increased privacy in the sense that it is harder for an attacker to learn that a specific user in nym a rated item v, whereas when $p(u, v) = 1$ then every user in nym a has rated item v and so an attacker can be certain that a target user in nym a has rated that item.

More rigorously, the anonymity property of a user-item pair can be expressed in terms of *hiding property*,[3] which is a measure for accounting the adversary's maximum probabilistic belief that a user is the sender (rater) of a given message (rated item). According to such metric, the pair user-item (u, v) is assumed to be hidden if the probability is smaller than a threshold τ.[28] In our case the system provides all the necessary information to enforce such threshold in two ways: users know both $\mid \mathcal{O}_{av} \mid$ and $\mid \mathcal{U}(a) \mid$ and thus can avoid joining a specific nym or rating a given item when such ratio is greater than τ; nym providers can decide to join nyms or prevent nym splitting to enforce on their side the threshold as well, for all the possible items. More formally, a user u has a τ-hiding property[‡] if:

$$\text{priv}_{\text{HP}} \equiv \tau, \quad \text{where } \forall v : p(u, v) \leq \tau.$$

The threshold can in practice be very small, given that usually the preference matrix are very sparse.

Note that the attack considered here does not reveal the numerical rating submitted by a user for an item, only the likelihood that a user rated the item. Hence the attack is rather limited in nature and a more powerful, and significantly more difficult, attack would be needed in order to estimate the numerical ratings, given that individual user ratings are not stored anywhere apart from user clients.

3.3 | Other attacks

3.3.1 | Sybil attacks

Attacks by dishonest users who submit false ratings in an attempt to manipulate the recommendations made by the system are an important challenge for all recommender systems.[29,30] Typically such attacks are mitigated by forcing users to login to an account linked to an external identity, for example, a credit card, or by restricting the rate at which ratings can be submitted, for example, by restricting ratings to people who have paid for an item. By weakening the link between rating submissions and user identity the use of OpenNym therefore potentially facilitates Sybil attacks. Nevertheless, there is a number of ways that retain user anonymity while restricting the rate at which ratings can be submitted. For example, users can authenticate against the system to obtain tokens/e-cash. This step is not anonymous and since authentication is required the number of tokens generated per user can be restricted. The tokens thus obtained can then be used when submitting ratings via an OpenNym identity. This step can be carried out in an anonymous fashion using a variety of cryptographic techniques. In more detail, using the approach of Chaum et al.,[31] each user mints a number of session tokens (with associated serial number), blinds them with a secret blinding factor and forwards them to the recommender system through a non-secure channel. The system then signs the tokens with its private key, without knowledge of the serial number associated with the tokens. On receiving the signed tokens back from the recommender system, the user can remove the blinding factor and use the tokens to submit ratings to the system anonymously. Double use of tokens is prevented by the system maintaining a database of the serial numbers of all tokens that have been issued. This token approach therefore allows restricting use of the recommender system to pre-authenticated users and also limiting of the rate of submission of ratings by each user. It does this while retaining the anonymity and privacy provided by use of OpenNym when submitting ratings and obtaining recommendations.

3.3.2 | Additional linking attacks

The OpenNym approach is a technique for ensuring that the content of the messages exchanged between client browser and the service provider provides unlinkability between user ratings and their individual identity. Of course this leaves open the possibility of other vectors for carrying out linking attacks.

One such approach in web or mobile applications is for the service provider to attempt to place secondary cookies or third-party tracking content on the web pages viewed by a user. Within the EU the GDPR rules require that users be explicitly informed of such actions and must take a positive step to opt in, hence attempts at such tracking seem like a relatively minor concern. Outwith the EU, existing tools for blocking third-party trackers can be used, leaving the setting of unique identifying first party cookies as the main concern. This can be mitigated by standard approaches for example, by activists maintaining lists of cookies that can be safely used (similar to existing lists of malware sites, trackers, and so on) and users blocking the rest.

Another possible vector of attack is to record the IP address of the user browser, and thereby try to link the ratings back to the individual user. However, due to the widespread use of NAT and other middleboxes use of IP addresses as identifiers is often ineffective.[32] Users also have the option of using tools such as TOR to further conceal the link between the IP address revealed to the server and the user's identity. A more sophisticated approach is to try to use the timing and ordering of ratings submissions to cluster and link them to an individual user. There are two main mitigations that can be used against timing attacks. One is to inject dummy rating submissions so as to disrupt timing and ordering information. The other is to buffer ratings and delay their submission, again so as to disrupt timing and ordering information. When applied to individual users in isolation both of these mitigations potentially significantly impact on user experience and network/server load. Of course timing based attacks are not confined to recommender systems and there is, in particular, a growing literature on such attacks against HTTPS traffic and on defenses. Borrowing from this literature, when user traffic can be aggregated, for example by use of a shared VPN, then relatively low amounts of buffering and injection of dummy traffic are sufficient to provably disrupt timing-based attacks[33] and such approaches might usefully be adopted in the present context.

4 | SYSTEM ARCHITECTURE

In this section, we introduce the system architecture to enable the use of OpenNym shared identities with existing web-based recommender systems. As already introduced in Section 1 and with reference to Figure 1B), the proposed architecture is comprised of three entities: User client, Recommender system (RecSys) and OpenNym repository.

4.1 | User client

The client's main responsibility is emitting requests for recommendation of given items and obtain back the responses as recommended values. In traditional system, as shown in Figure 1A), this is done by implementing two set of operations:

1. send item ratings to the RecSys;
2. obtain recommendations for a set of given items.

Conversely, in the proposed system, the user client needs to be more complex and to perform the following activities:

1. fetch information about available nyms by querying the OpenNym repository;
2. select a profile with a *group profile estimation*;
3. send item ratings to the OpenNym repository and store them privately on a local database;
4. connect with the recommender system using the selected nym account to obtain recommendations.

While the group profiles stored in the OpenNym repository are visible to all users of the service, the individual ratings history of each user is stored within the User client and never leaves the user's browser. The client (which can be for instance implemented as a browser plugin) is therefore a trusted element in the architecture and user confidence in this plugin might be ensured by, for example, making it available in open source form.

4.2 | Recommender system

This solution does not imply any change in the existing recommender systems that will work as usual. Our solution is agnostic to the underlying implementation of the recommender system, but we refer to Section 8 for a theoretical analysis of the (relatively mild) recommender systems constraints under which we obtain convergence guarantees.

4.3 | OpenNym provider

The OpenNym provider is one or more servers which provide and collect information regarding the nyms. In particular, each provider will:

1. keep a list of available nyms;
2. for each nym, provide the sum of all the ratings for a given item together with the number of ratings (in order to let the User client compute the average value);
3. update such information when a user rates a new item;
4. provide splitting/joining functionality to split/merge nyms according to the evolution of the system and the accuracy/privacy thresholds. The OpenNym repository can decide to split one nym in two when the predictive performance falls below a certain threshold, provided that the computed privacy metrics constraints are satisfied, as explained in details in Section 3.

As shown in Figure 2 and discussed in more details in the next section, these functionalities can be implemented with an OpenNym profile estimator and an OpenNym repository.

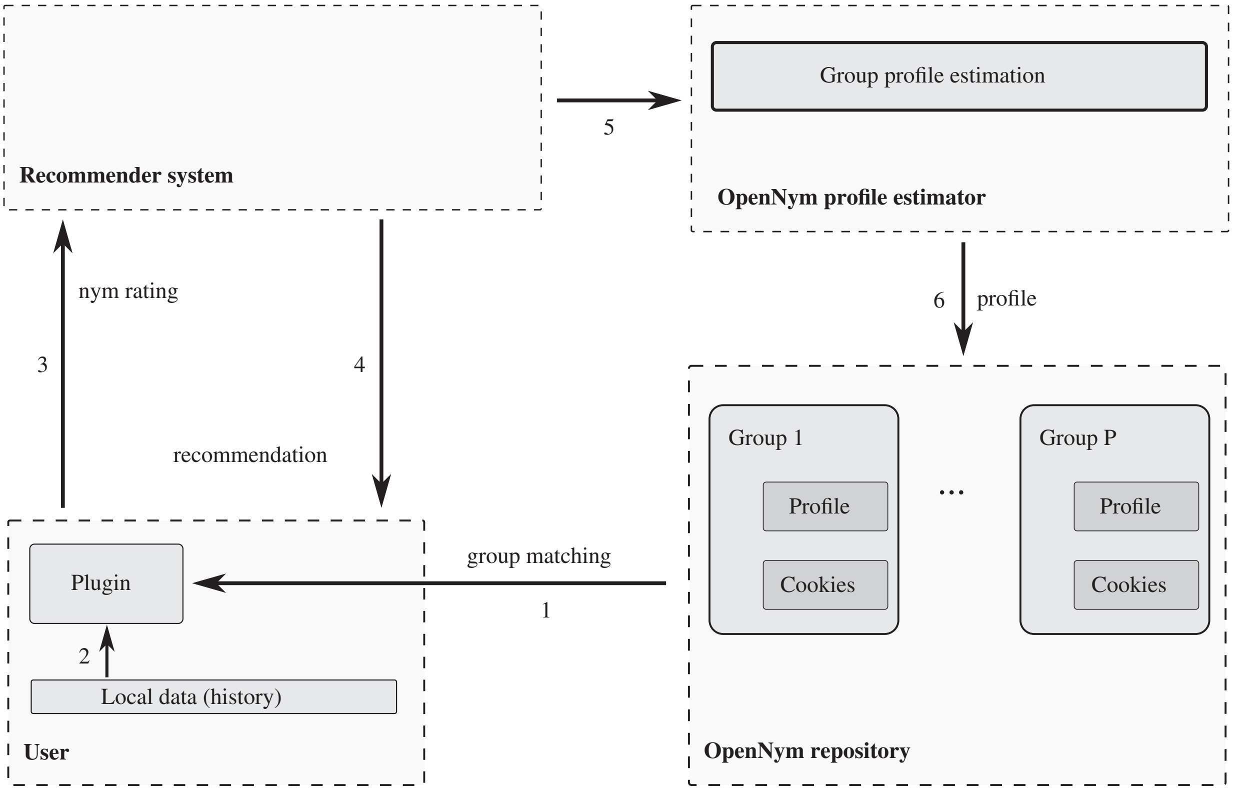


FIGURE 2 OpenNym workflow to interact with existing systems

5 | PROTOTYPE IMPLEMENTATION

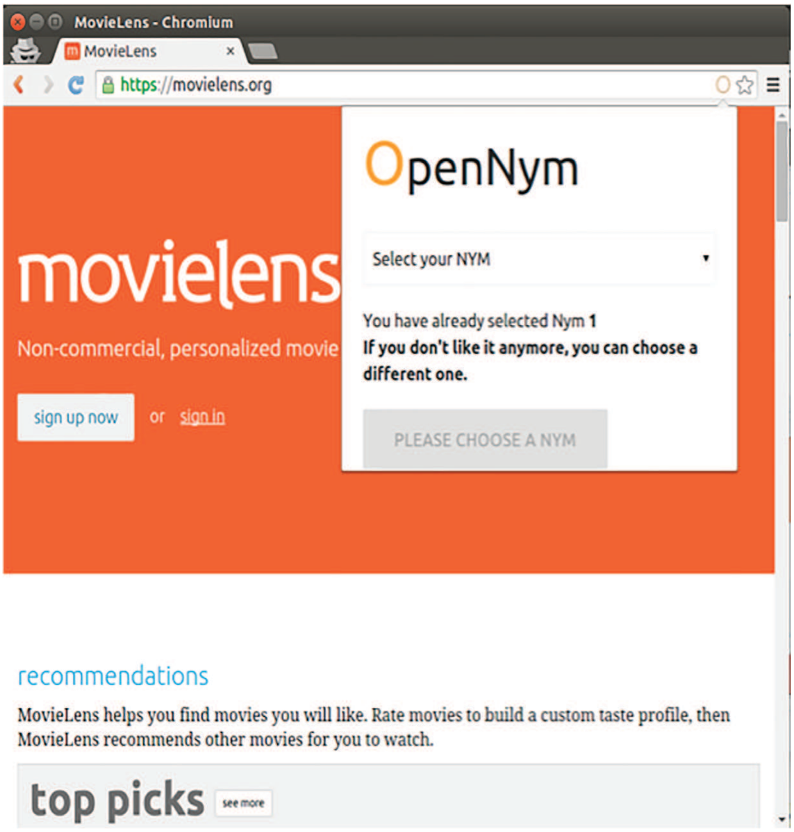
In this section, we describe the implementation of an OpenNym prototype for a web applications, by defining a simple nym association policy (adaptive nym selection). In such kind of applications, cookies are currently the primary approach for identifying a user accessing an online service. Users can therefore access a service using the same OpenNym identity by submitting the same application cookies to the service. This approach requires no changes to existing web systems, and existing systems for use of cookies/identifiers and rating of items can be retained. Changes are, however, needed in the user web browser in order to manage the application cookies and an OpenNym repository is needed that provides group information and exports the APIs needed to update group characteristics according to the new elements rated by users. Alternative implementations of OpenNym on other recommender systems based on this work are presented in Reference 34.

5.1 | Operations

In this section, we present the operations with reference to the workflow depicted in Figure 2, describing how OpenNym can function on top of existing web-based applications which use a recommender system.

5.1.1 | Group selection (steps 1 and 2)

The User client selects an appropriate group identifier either via manual user selection or by comparing the public group profiles stored in the OpenNym repository with the user's individual ratings history, the latter being stored privately within the browser and never leaving the user's device. An example user interface for manual group selection is shown in Figure 3 (for use with Movielens).



You have already selected Nym 1
If you don't like it anymore, you can choose a different one.

MovieLens helps you find movies you will like. Rate movies to build a custom taste profile, then MovieLens recommends other movies for you to watch.

FIGURE 3 Illustration of the OpenNym plugin login functionality

5.1.2 | Recommender system access (steps 3 and 4)

The plug-in intercepts user communication with the recommender web service, inserts the cookies for the selected identity and then forwards the communication to the recommender web service. The only information sent between the user browser and the web service is the group identity and any ratings submitted, there is no need for the user to disclose their individual identity. For recommender systems which do not support submission of multiple ratings for the same item then the submitted user ratings are also modified by the plug-in, namely such that the submitted rating for an item is the average of the existing ratings and the newly submitted rating (as explained in detail in Section 8).

5.1.3 | Group profile estimator (steps 5 and 6)

A profile estimator is needed which constructs/updates the profiles of the shared accounts (stored in the OpenNym repository and used to assist users to select the OpenNym identity that most closely matches their personal preferences), steps 5 and 6 in Figure 2. This can be implemented either by accessing the recommender system using each shared account and deriving the information needed to build/update the account profile, or by directly collecting data from the recommending system (eg, by web scraping or via API) and constructing/adjusting the shared account profiles accordingly. This task must be executed periodically in order to keep the shared account profile information fresh.

5.2 | Nym repository implementation

Each nym repository contains an *internal status* for each nym it serves. The status is composed by two variables which are the number of ratings received N_m and the sum of the ratings received R_m, for each item m.

The nym repository exposes an API with the following three functions:

addRating: this function increments (decrements) by one the number of ratings N_m and update the score R_m for a given item m, and it is implemented by just updating $R_m = R_m + r$ and $N_m = N_m + 1$.

rmRating: this function is implemented by just updating $R_m = R_m - r$ and $N_m = N_m - 1$. The function is called when a user leaves a nym.

getRating: this function returns the number of ratings N_m and the current score R_m for a given item-nym pair.

The nym repository has an internal logic for creating new nyms or for merging different nyms. This is an adaptive logic that should follow the evolution of the data provided by the users. In the simulation presented in Section 6, we implemented the split ability by just duplicating nyms when the number of collected ratings is above a given threshold.

Depending on the legacy recommender system architecture, the repository might expose a function *getCookie*, that returns the session cookie for a given shared account/nym and website.

5.3 | Client implementation

As a proof of concept of the feasibility of the proposed approach with existing web services we implemented OpenNym as a Google Chrome extension[§] for use with Movielens,[¶] a popular website for movie recommendations. We evaluate its performance below but here we briefly describe the implementation.

The extension implements OpenNym as a *page action*[**] so that an icon appears in the address bar only when the user is visiting a certain website (as shown in Figure 3). By clicking on that icon the user can manually select the shared account with which to access Movielens. Underneath, when Movielens is accessed the related session cookie is downloaded from the OpenNym server and installed on the user browser. The user can at any time either manually change the shared account used or the process can be automated based on the ratings history of the user. To allow automatic selection of the shared account to use the plugin stores all the scores locally on the user browser. This data can also be stored in encrypted form on a remote server and downloaded by the client on demand, to support the usage of multiple web browsers for the same user (eg, mobile and laptop).

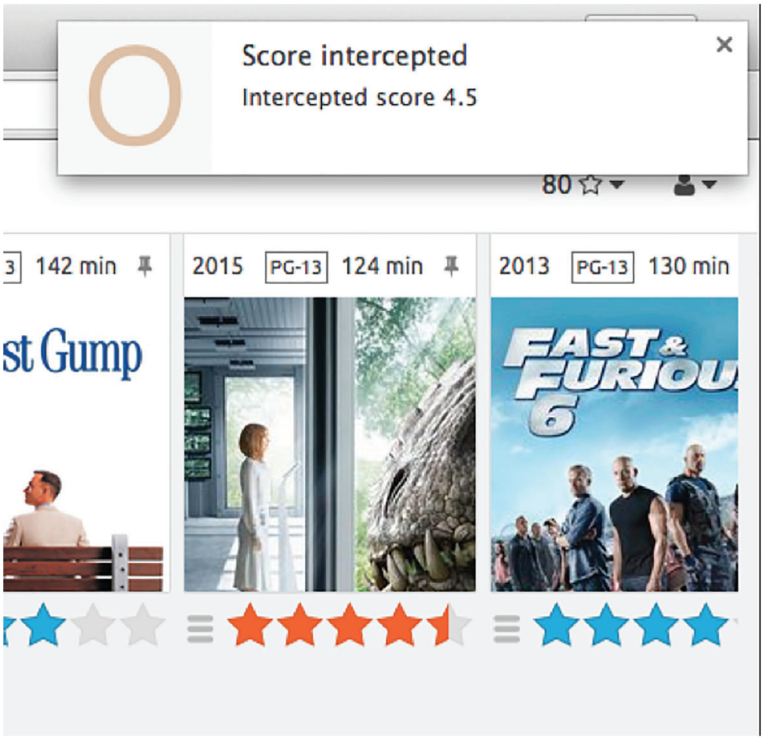
The plugin also launches a background process responsible for intercepting the ratings submitted by the user. This traps the message to the server and changes the rating according to the average of the selected shared account as shown in Figure 4.

Operation of the extension is facilitated by an OpenNym repository that (a) stores the list of shared accounts and the related session cookies for each website and (b) for each movie and shared account keeps track of the number of scores and their average value. The User client may contain a logic for choosing the right nym adaptively. In the simulation described in Section 6, we adopt the following logic. For each available nym and for all items already rated by user u, compute the prediction error as the difference between the rating predicted by Movielens and the user rating r. Then:

- Select the nym with the minimum prediction error.
- If the selected nym is different from the current nym, remove all the ratings by this user from the current nym (by decrementing each item rating by r and each counter by one in the nym internal status) and then join the new nym by updating its internal status by bulk adding the rating of all the rated items.

The pseudocode of this simple policy is reported in Algorithm 1 which makes use of three functions that, in turn, call the APIs exposed by the repository. Specifically:

- *calculateError*: calculate the error according to a metric (eg, RMSE) between the scores given by a user (stored locally), and the ratings predicted by the nym (using the *getRatings* API);
- *join*: iteratively uses the *addRating* to populate a nym repository with all the ratings belonging to the user
- *leave*: iteratively uses the *rmRating* to remove the ratings belonging to the user from the nym repository.



Score intercepted

Intercepted score 4.5

FIGURE 4 Illustration of the OpenNym plugin score interception functionality

Algorithm 1. Adaptive nym selection

```
bestNym ← NULL
bestErr ← MAX_VALUE
for $n$ in NYMS do
    err ← calculateError(myScores, getRatings($n$))
    if err < bestErr then
        bestNym ← $n$
        bestErr ← err
    end if
end for
if bestNym ≠ currentNym then
    leave(currentNym)
    join(bestNym)
    currentNym ← bestNym
end if
```

6 | PERSONALIZATION PERFORMANCE

In this section, we use the proof of concept OpenNym implementation to evaluate the personalization performance when using shared identities to access the Movielens recommendation service,[††] that is, we evaluate the loss of personalization, if any, incurred by using shared rather than individual identities. We first assume that identities have already been defined and roughly corresponds either to a system where these identities have been crafted manually, for example, using demographic and other public information, or where the system is already operating in steady state. We then relax this assumption, and analyze the cold start properties in Section 6.4.

TABLE 1 RMSE tested on Movielens website with “Wizard” engine

ID	Nym size	Size of ratings per user group	Number of users in nym	RMSE (OpenNym)	RMSE (individual accounts)	RMSE (naive)	Performance gain in using OpenNym [%]
1	Smallest	20	466	**1.496**	1.507	1.551	0.739
2	Smallest	30	466	1.272	**1.260**	1.325	−0.908
3	Smallest	134	466	1.358	**1.304**	1.338	−4.161
4	Median	20	1724	**0.993**	1.005	1.232	1.240
5	Median	41	1724	**0.934**	0.945	1.185	1.188
6	Median	272	1724	**0.870**	0.874	1.133	0.470
7	Biggest	20	5442	**0.670**	0.734	1.081	8.720
8	Biggest	157	5442	0.839	**0.812**	1.071	−3.368
9	Biggest	2898	5442	**0.789**	0.792	0.960	0.414

Note: The best result for each dataset is indicated in bold.

6.1 | Movielens recommendation service

Movielens has several recommending engines. The “Wizard”‡‡ is a matrix factorization recommender based on Reference 35 and Reference 36, using 50 features, 125 training epochs per feature, and subtracting the user-item personalized mean prior to factorizing the matrix. Since this is both the most accurate engine and that most preferred by users, as shown in Reference 37, we focus on testing using this engine. To explore the sensitivity of our results to the choice to engine we also collected data for an engine called the “Peasant”, a user-item personalized mean algorithm partially described in Reference 37.

6.2 | User ratings data

In order to focus on personalization performance we used the Movielens 20 M dataset§§ (with 26 744 movies, 138 493 users and 20 million ratings) and BLC[25] to perform offline clustering of users into 64 nyms¶¶: in that work, this number of nyms has been proven sufficient to jointly achieve personalization and privacy. The use of online clustering is evaluated in detail in Section 8. To minimize the load and potential perturbation of ratings of real users on the Movielens website, we selected 3 of these 64 clusters for study: that containing the smallest number of users, that with the median number of users and that with the largest number of users, with the aim of gaining a fair idea of the range of behaviors. For each of these three nyms, we created three datasets by sorting users by the number of ratings each submitted and grouping these into subsets of users with the smallest, median and largest numbers of ratings (relative to the nym). In this way, we obtained a total of nine sub-datasets consisting of (user, movie, rating) triples, as summarized in Table 1.

6.2.1 | Measuring performance

For evaluating performance we split each of these datasets into a training and a test set. The Movielens most advanced recommender system (the “Wizard”) needs 15 ratings to being able to give predictions and so the training set consists of 15 ratings per user, and the test set consists of the rest of the ratings for each user.

For a sample of users*** (32 on average) in each training dataset, we: (a) submit the user ratings using a separate account for each user in the Movielens website (in total, 4320 ratings are submitted to the online system), then (b) acquire the predicted ratings for all the test movies in the dataset. In total, 36 304 queries are made. We then repeat (a) for each training dataset but now using a shared account whose ratings for a movie are the average ratings among all the users using that account and then (b) compare each user test rating with the Movielens predictions for this shared account. We automated the submission of ratings and collection of predictions by using the (undocumented) rest APIs of Movielens.†††

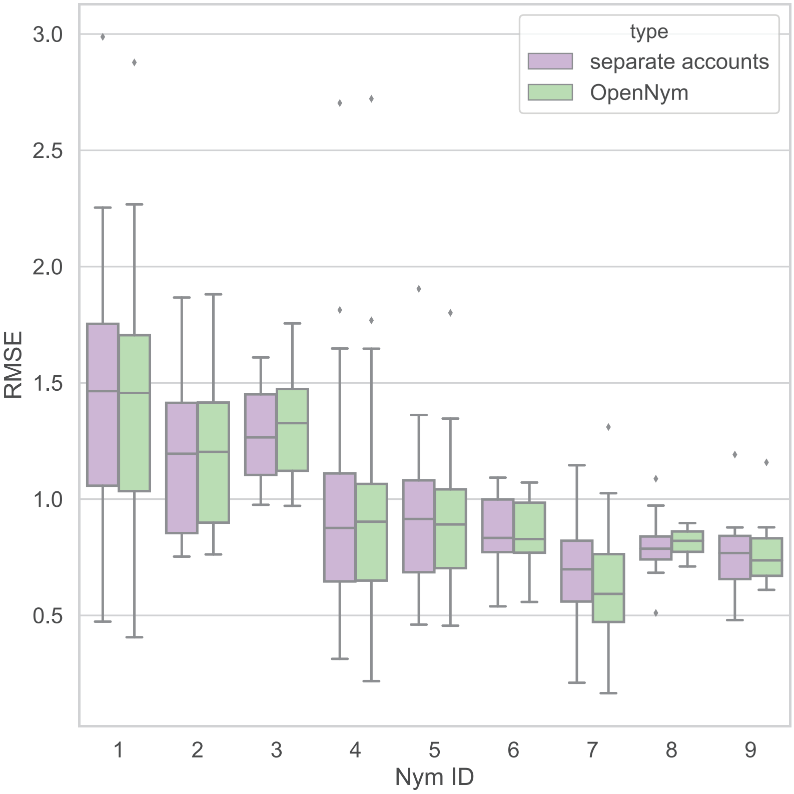


FIGURE 5 Per user RMSE between predicted ratings and test ratings for Movielens website with "Wizard" engine

The metric used to assess the performance is the root mean square error (RMSE) between the ground truth and the predicted ratings (either using OpenNym or separate accounts).

6.3 | Results

6.3.1 | "Wizard" recommender engine

In Table 1 we report the measured performance when users access the system using a shared account (ie, an OpenNym identity) and when they access the system using individual per-user accounts. These results are for the "Wizard" recommender engine of Movielens. We also show, as sanity-check, the measured performance of a naive technique where for each movie we use the global average rating over the training set as prediction, to verify that a more complex recommender system is indeed helpful. It can be seen that using OpenNym leads to an average per-user RMSE decrease of 2.7% (two-sided Wilcoxon $P < .05$ after Bonferroni-Holm adjustment) compared to use of individual per-user accounts. This is a clear indication that OpenNym does not cause a loss in prediction accuracy: indeed the prediction accuracy slightly improves because the recommender system can leverage the additional information from the other users sharing an account.

In Figure 5 the per-user distribution of the RMSE is shown. Broadly speaking, it can be seen that the distributions are much the same when using both OpenNym and individual per-user accounts. This is consistent with the results in Table 1. Two trends are also apparent for both methods. Firstly, it can be seen that the users that have been clustered in a smaller nym tend to achieve worse predictive performance (compare the left-hand box plots against those on the right hand of the plot). Since the users have been clustered by BLC into a small nym presumably these users have specialized preferences and this is also why Movielens is less able to predict their ratings accurately. Secondly, it can be seen that as the number of ratings submitted by users increases the variance of the prediction error tends to fall (in each group of three box plots in Figure 5 when moving from left to right the variance decreases), although the mean error stays approximately unchanged.

6.3.2 | "Peasant" recommender engine

We repeated the experiment using the "Peasant" recommender engine of Movielens. Since this engine does not need many ratings before providing recommendations we rearranged the training and test sets such that only 10 ratings per user are used as the training set, and the rest of the available ratings are used as the test set (Figure 6).

The full results are presented in Table 2 and are qualitatively comparable with the ones already presented for the "Wizard" engine. This lack of sensitivity to the choice of engine is encouraging as it suggests that the OpenNym architecture is robust. For this recommender system, using OpenNym leads to an average per-user RMSE decrease of 1.0% (two-sided Wilcoxon $P < .05$ after Bonferroni-Holm adjustment) compared to use of individual per-user accounts.

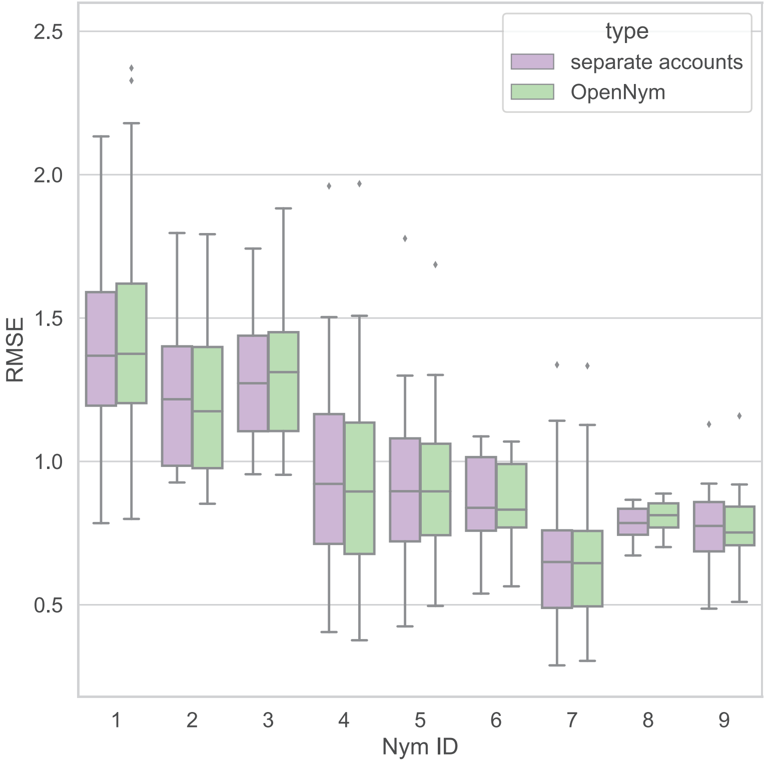


FIGURE 6 Per user RMSE between predicted ratings and test ratings for Movielens website with "Peasant" engine

TABLE 2 RMSE tested on Movielens website with "Peasant" engine

ID	Nym size	Size of ratings per user group	Number of users in nym	RMSE (OpenNym)	RMSE (individual accounts)	RMSE (naive)	Performance gain in using OpenNym (%)
1	Smallest	20	466	1.583	**1.468**	1.474	0.387
2	Smallest	30	466	1.374	**1.261**	1.280	1.468
3	Smallest	134	466	**1.325**	1.375	1.331	−3.321
4	Median	20	1724	1.173	**0.967**	0.983	1.661
5	Median	41	1724	1.160	**0.936**	0.945	0.961
6	Median	272	1724	1.133	**0.872**	0.878	0.714
7	Biggest	20	5442	0.990	**0.700**	0.703	0.478
8	Biggest	157	5442	1.087	**0.833**	0.812	−2.637
9	Biggest	2898	5442	0.954	**0.797**	0.798	0.147

Note: The best result for each dataset is indicated in bold.

6.3.3 | Top-k Kendall-tau distance

As already remarked in Reference 38, the RMSE is not necessarily the best metric for evaluating real world recommender systems since a user often will not query the predicted rating of a specific item, but rather will passively receive a target recommendation ("you might be interested in … "). Moreover, recommender systems can be used in search engines, and also there, typically the "first page" of results is the only one that matters. For this reason we follow the approach in Reference 39 and use the top-k Kendall-tau distance, a generalization of the Kendall-tau distance able to consider partially matching lists.

We computed the top-50 Kendall-tau distance (with parameter $p = 0.5$ as defined in Reference 39) between the predicted ratings and the test ratings for both OpenNym and when individual per-user accounts are used. Interestingly, we found that for 98.6% of the users OpenNym preserves the ordering of the predicted ratings, and consequently the Kendall-tau distance is equal for the majority of the cases (98.9%). This means that while OpenNym exhibits slightly better performance in terms of RMSE, the top-50 recommended items would almost always be the same when using the two mechanisms. Again, this is encouraging as it demonstrates in a concrete way that increased privacy (which we discuss next) need not come at the cost of degraded predictive performance.

6.4 | Nym learning and cold start

The results obtained so far assessed the performance of the OpenNym system when an appropriate user-nym matching is known. Now we focus on the issue of nym learning from a cold start: nyms need to be created/updated, and users need to be (dynamically) associated to the right nym.

We begin from the same settings described in Section 6.2 and select the four nyms (labeled A, B, C, D) with highest Euclidean distance from each other in the latent space: the goal is to see how the transient period compares to the steady state, when the users and the OpenNym repository starts with zero knowledge. We took a training set of 2520 ratings made by 121 unique users to 587 unique movies equally sampled from nyms A–D . At each update, we compute the RMSE of each user, using a (small) test set comprised of three ratings for each user.‡‡‡

To test the whole bootstrap process, we made the simulation start from a clean slate where only one initially empty nym is available; then after considering respectively 10, 20, and 30 ratings, we introduce into the system new nyms, passing from one initially active nym to four active nyms after 30 ratings. A new nym is introduced by cloning the last-activated nym: at the next rating, users on the nym that has been cloned will approximately split in half in the new one.

6.4.1 | Experimental setup

We generate a sequence of triples (user, movie, and rating) from the aforementioned dataset.

We implemented the system as described in Section 5, with the nym selection policy defined by Algorithm 1. Each nym has an *internal status* containing, for each movie, the number of ratings received N and the sum of the ratings received R. For each triple (u, m, r), we perform the following actions:

- If r is the first rating of user u, choose a random nym n.
- User u impersonates nym n, adds r to R and increments N by one.
- Update Movielens rating m to be equal to $\left\lceil \frac{R}{N} \right\rceil$.
- For each available nym and for all movies already rated by user u, compute the prediction error as the difference between the rating predicted by Movielens and the user rating.
- Select the nym with the minimum prediction error.
- If the selected nym is different from the current nym, remove all the ratings by this user from the current nym (by decrementing each movie rating by r and each counter by one in the nym internal status) and then join the new nym by updating its internal status and updating any rating in Movielens for which the corresponding $\left\lceil \frac{R}{N} \right\rceil$ has changed.

When a new nym enters the system, it is created by the duplication of a nym already active in the system. Every 360 ratings, the performance for all the active users (ie, the users that rated at least one movie since the beginning of the simulation) is evaluated against a test set composed of three values for each user.

Figure 7 shows that the ratio of nym changes decreases over the time and tends to stabilize below 20% after 1500 ratings. Figure 8 shows how the transitory phase tends to converge to the steady-state RMSE, containing most of the difference in the order of 10% right after 1000 ratings. The measured aggregated RMSE as the system evolves is shown in Figure 9.

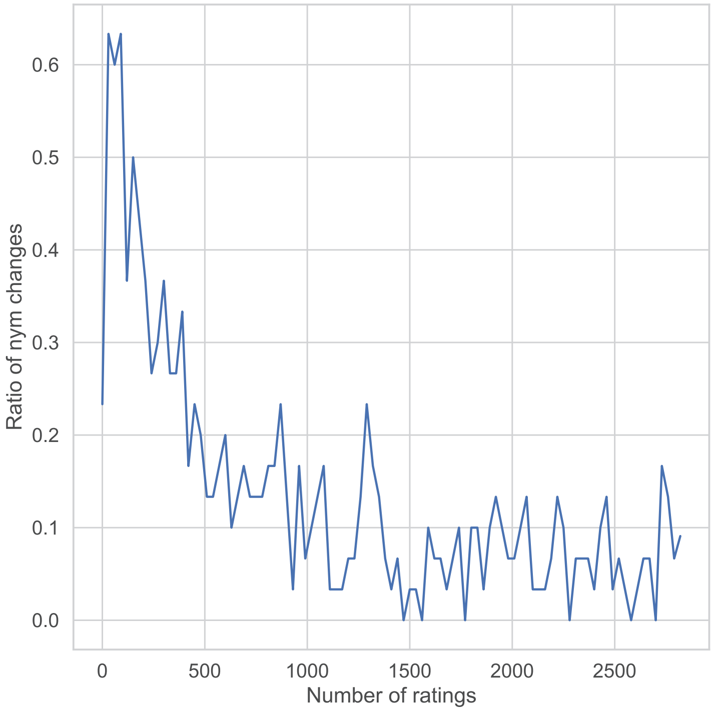


FIGURE 7 Proportion of nym changes from cold start, using a window of 30 ratings

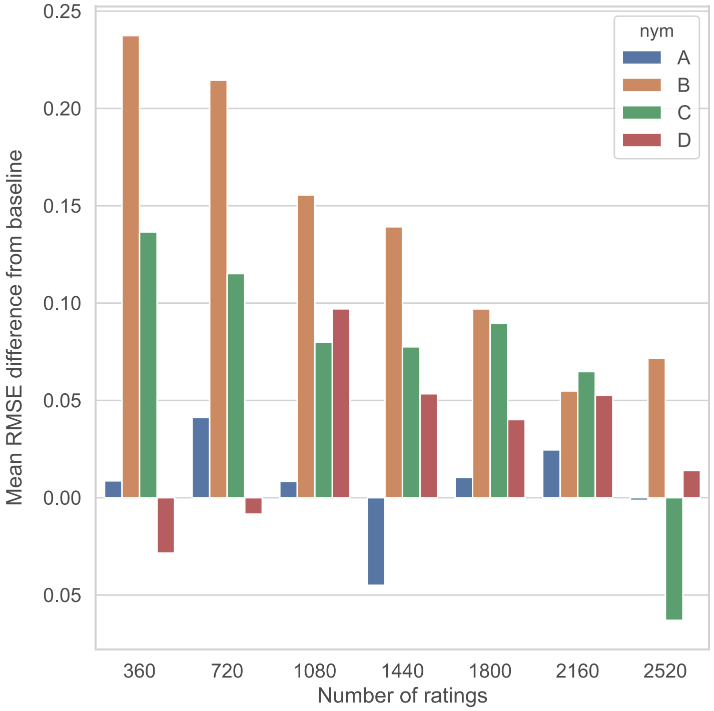


FIGURE 8 Mean user RMSE difference from steady state one over time

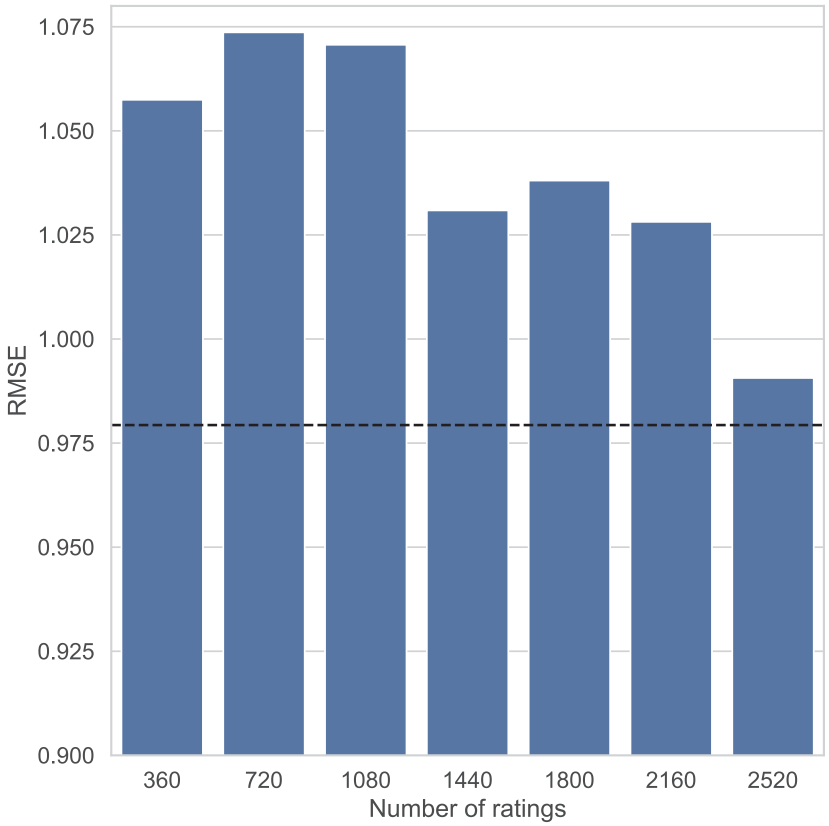


FIGURE 9 RMSE over time from cold start. The dotted line is the computed steady state from previous sections

6.4.2 | Vulnerability assessment

This proof-of-concept implementation is open to attacks, which we briefly discuss here.

Batch rating

When a user change a nym might needs to send a lot of ratings, risking a temporal correlation attack. This attack can be neutralized by introducing artificial random delays in the ratings and using mitigation techniques such as the ones described in Section 3.3.2 to prevent linkability.

DoS attack

A malicious agent can act as a user, and providing ratings in a malicious way to, for instance, erase the status of all the nyms. We refer to Section 3.3.1 for a more detailed discussion of this attack.

7 | PRIVACY PERFORMANCE

As discussed in Section 3, the main attacks of interest are those which seek to learn: (a) which nym a user belongs to, and (b) which items a user has rated, given knowledge of the nym to which the user belongs.

7.1 | Defending against attack 1

For the Movielens setup described in the preceding section, the best guess for an attacker on which nym a user belongs to corresponds to a probability of about 3.9% (calculated as the ratio of users in the biggest nym over the total number of users). We argue that for this example the use of nyms therefore provides a reasonable level of protection against this first type of attack.

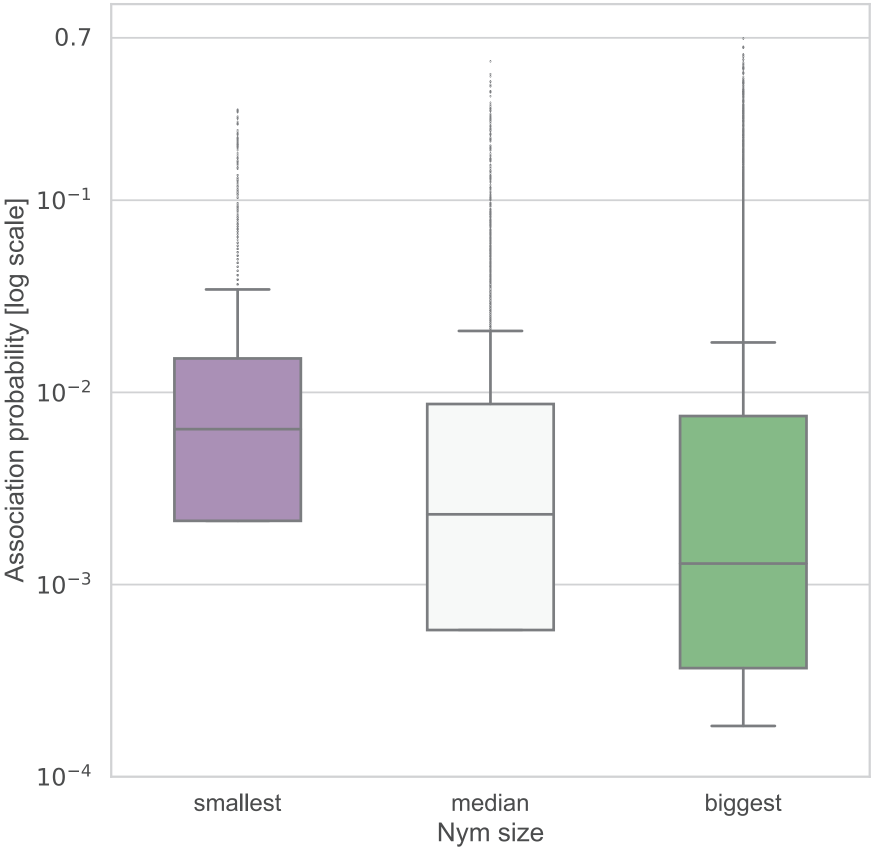


FIGURE 10 Measured movie association probability distribution for the three nyms considered

7.2 | Defending against attack 2

In Figure 10 we show the measured distribution of the movie association probability $p(u, v)$ for the three nyms (smallest, median, and largest). The median of these values is consistently below 1%, indicating a generally high level of privacy. That said, it can also be seen that a small fraction of movies that are rated by many of the users sharing a nym (the dashed lines in Figure 10 indicating outlier points extend to a value of almost 0.7). As discussed in Section 3, these outliers could be easily avoided by the user clients, by computing $p(u, v)$ and avoiding to rate an item when such probability is greater than their (possibly item-specific) threshold.

8 | THEORETICAL ANALYSIS

We now demonstrate that OpenNym enables a private use of existing non-private online recommender systems, and can guarantee convergence without (a) the need to access the online recommender system, and (b) the knowledge of the individual user identities or ratings by the online recommender system.

By moving from the conventional architecture (Figure 1A), to the OpenNym architecture (Figure 1B) the overall system is performing two types of optimizations: (a) the optimization performed by the (legacy) recommender systems, which typically aims to reduce a loss function on the user-item rating[40]; and (b) the optimization defined by the association policy of the nyms, which leads to private users clustering. We now present the conditions under which the independent *combination* of such two optimizations (ie, using OpenNym) converges to the same result we would have by using the conventional recommender system (ie, without OpenNym).

In other words, the basic approach to online clustering that we describe is alternating minimization[41]: the repeated process of (i) users selecting the OpenNym identity most closely aligned with their preferences and submitting ratings using this and (ii) adaptation of the recommendations for each OpenNym identity based on these submitted ratings. We show that under mild conditions this process is convergent and results in users being clustered according to their preferences. Importantly, this is achieved in a privacy-preserving manner since ratings are only ever submitted to the system in step (i) using group identities. It is also backward compatible with existing systems since any suitable recommender can be used in step (ii).

8.1 | Formal setup

We proceed by expressing the above clustering approach more formally as follows. We have a set of user accounts $\mathscr{A} := \{1, \dots, a\}$, a set of items $\mathscr{V} := \{1, \dots, m\}$ and a sequence of user-item ratings (a_k, v_k, r_k), $k = 1, 2, \dots$ with user account $a_k \in \mathscr{A}$, item $v_k \in \mathscr{V}$, and corresponding rating $r_k \in \mathbb{R}$. Let $\mathscr{O} \subset \mathscr{A} \times \mathscr{V}$ be the subset of item-user account pairs for which ratings have been submitted and gather the user-item ratings into matrix $R \in \mathbb{R}^{a\times m}$ with entry R_{av} containing the rating of item v submitted by user account a, $(a, v) \in \mathscr{O}$ (so this matrix has many missing values).

Let $\mathscr{U} := \{1, \dots, n\}$ denote the set of individual human users and suppose, for now, that each user $u \in \mathscr{U}$ employs account $u_a \in \mathscr{A}$. We can capture the mapping from users to user accounts using matrix $P \in \{0, 1\}^{n\times a}$. Namely, row P_u of P has a 1 in the column corresponds to account a employed by user u and zeroes in all other columns.

Online clustering aims is to jointly find an assignment of users to accounts and ratings predictions for each account such that

$$P, \hat{R} \in \arg \min_{\hat{R}\in\mathcal{R}, P\in\{0,1\}^{n\times a}: P1=1} \ell(R, P\hat{R}, \mathscr{O}) \tag{1}$$

where $\ell(R, \hat{R}, \mathscr{O})$ is a non-negative loss function and 1 denotes the all ones column vector (so $P1 = 1$ ensures that each row of P has a single non-zero element).

8.2 | Online clustering

We can solve for P and $\hat{R}$ in alternating fashion, namely by solving

$$P_k \in \arg \min_{P\in\{0,1\}^{n\times a}: P1=1} \ell \left(R, P\hat{R}_{k-1}, \mathscr{O}\right) \tag{2}$$

$$\hat{R}_k \in \arg \min_{\hat{R}\in\mathcal{R}} \ell \left(R, P_k\hat{R}_{k-1}, \mathscr{O}\right) \tag{3}$$

in turn for $k = 1, 2, \dots$ until converged. Since each update is a descent step it is easy to show that these alternating updates will converge to a stationary point, although this may of course only be a local minimum. Indeed, for convergence all we need is that each of these updates is a descent step and so we can relax update (2) to

$$P_k \in \left\{P \in \{0, 1\}^{n\times a} : P\mathbb{1} = \mathbb{1} \text{ and } \ell \left(R, P\hat{R}_{k-1}, \mathscr{O}\right) < \ell \left(R, P_{k-1}\hat{R}_{k-1}, \mathscr{O}\right)\right\}. \tag{4}$$

The importance of this observation in the present context is that we do not require that the loss function used to update $\hat{R}_k$ be the same as the loss function used to update P_k, namely we can select

$$P_k \in \arg \min_{P\in\{0,1\}^{n\times a}: P\mathbb{1}=\mathbb{1}} \hat{\ell} \left(R, P\hat{R}_{k-1}, \mathscr{O}\right) \tag{5}$$

provided $\hat{\ell}$ is compatible with loss function ℓ in the sense that it satisfies descent requirement (4). This not only provides great flexibility but when updating P_k also avoids the need to know the internal details of the mechanism used to update $\hat{R}_k$. In other words, we can use any existing recommender system to perform the $\hat{R}_k$ update, augment it with group selection update P_k having compatible choice of loss function $\hat{\ell}$ and in this way obtain a new OpenNym recommender system with group-based user accounts.

8.3 | User rating obfuscation

We can usually assume that a $\hat{\ell}(R, P\hat{R}, \mathcal{O})$ with the user-separable form $\sum_{u\in\mathcal{U}} \hat{\ell}_u \left(R_u, P_u\hat{R}, \mathcal{O}_u\right)$ can be used, where R_u denotes the row of matrix R containing submitted ratings by user u, P_u is the row of P associated with user u (since P_u has a single unit non-zero element in column a then $P_u\hat{R}$ is row a of $\hat{R}$) and $\mathcal{O}_u$ the set of items rated by user u. For example, when $\hat{\ell}$ is a least squares or top-k loss function then it has this form. Similarly for item-based approaches, and also user-based approaches where the "users" are user accounts.

When this holds then in update (5) each row of P_k can be updated separately (and in parallel) by each user u. Namely, each user u selects the account a to use by individually solving

$$P_{u,k} \in \arg \min_{P_u\in\{0,1\}^a : P_u\mathbb{1}=\mathbb{1}} \hat{\ell}_u \left(R_u, P_u\hat{R}_{k-1}, \mathcal{O}_u\right). \tag{6}$$

This selection can be carried out privately within the user's browser with the ratings R_u submitted by user u not being revealed to the recommender system. Instead of seeing individual user ratings the recommender system only sees the collection of ratings submitted to each account $a \in \mathcal{A}$, and when multiple users share account a direct linkage of these ratings to a specific user u is broken.

8.4 | Backward compatibility

When $\ell(R, P\hat{R}, \mathcal{O})$ in addition has the item separable form $\sum_{v\in\mathcal{V}} \sum_{u\in\mathcal{U}} \ell \left(R_{uv}, \left(P_u\hat{R}\right)_v, \mathcal{O}_{uv}\right)$, for example when a least squares loss is used, then the $\hat{R}_k$ update can be decomposed as

$$\hat{R}_k \in \arg\min_{\hat{R}\in\mathcal{R}} \sum_{v\in\mathcal{V}} \sum_{u\in\mathcal{U}} \ell \left(R_{uv}, \left(P_u\hat{R}\right)_v, \mathcal{O}_{uv}\right) \tag{7}$$

$$= \arg\min_{\hat{R}\in\mathcal{R}} \sum_{v\in\mathcal{V}} \sum_{a\in\mathcal{A}} \sum_{u\in\mathcal{U}(a)} \ell \left(R_{uv}, \hat{R}_{av}, \mathcal{O}_{uv}\right) \tag{8}$$

where $\mathcal{U}(a) := \{u \in \mathcal{U} : P_{ua} = 1\}$ is the set of users sharing account a. Letting $\overline{R}_{av} = \frac{1}{|\mathcal{U}(a)|} \sum_{u\in\mathcal{U}(a)} R_{uv}$ be the average rating of item v by users sharing account a and $\mathcal{O}_{av}$ the set of users selecting account a and rating item v, then for some classes of loss functions the update further simplifies to

$$\hat{R}_k = \arg\min_{\hat{R}\in\mathcal{R}} \sum_{v\in\mathcal{V}} \sum_{a\in\mathcal{A}} \mid \mathcal{U}(a) \mid \ell \left(\overline{R}_{av}, \hat{R}_{av}, \mathcal{O}_{av}\right). \tag{9}$$

This holds, for example, when ℓ is the least squares loss (see Reference 25 Lemma 4.1).

Although it can be seen that stronger conditions are required than for the rest of the analysis above, when they hold the advantage is that it is sufficient to submit an average rating $\overline{R}_{av}$ to account a rather than the collection of individual ratings of the users sharing that account. This is helpful when, for example, using a recommender system for the $\hat{R}_k$ update that takes only the latest rating of each item by a user, such as Movielens.

Finally, if the number of users sharing an account $\mid \mathcal{U}(a) \mid$ is not known, (9) can be approximated by

$$\hat{R}_k = \arg\min_{\hat{R}\in\mathcal{R}} \sum_{v\in\mathcal{V}} \sum_{a\in\mathcal{A}} \ell \left(\overline{R}_{av}, \hat{R}_{av}, \mathcal{O}_{av}\right). \tag{10}$$

This approximation is required when we want to update $\hat{R}_k$ using an existing recommender system that expects a one-to-one mapping between users and accounts.

9 | CONCLUSION

In this work, we introduce OpenNym: a framework where users access online recommender systems by selecting from a set of available shared accounts (OpenNym identities). Group accounts are shared by many users, so providing a form of "hiding in the crowd" indistinguishability. We demonstrate that, by using an appropriate profile association policy, accurate user recommendations can be generated in a privacy preserving manner, without the need for a user's individual ratings to leave their browser/client, while remaining backward compatible with existing systems.

We implement a proof-of-concept prototype service based on these ideas. Users employ a browser plug-in to select an appropriate OpenNym identity. We demonstrate tools for private automated selection of the most appropriate identity and also support for incremental rollout by demonstrating operation with the existing, unmodified Movielens recommender service and user interface. Use of the OpenNym approach with Movielens yields an average per-user RMSE decrease of 2.7% for the "Wizard" recommender system, and of 1.0% for the "Peasant" recommender system (two-sided Wilcoxon $P < .05$ after Bonferroni-Holm adjustment). That is, performance is improved while providing a form of "hide in the crowd" deniability from attacks aimed at discovering which item have been rated by the targeted user, providing a median association probability of less than 1% for all cases.

Finally we analytically demonstrate and experimental verify the convergence of OpenNym over a set of legacy recommender systems: the system tends to converge to the steady-state (±10 %) after a relatively small number of ratings.

CONFLICT OF INTEREST

The authors declare no potential conflict of interest.

ACKNOWLEDGMENTS

This work is supported by the Science Foundation Ireland (SFI) grant 16/IA/4610.

ENDNOTES

[*] Even if IDs are not directly linked to user identities, they can be considered as quasi-identifiers, that is, they can potentially be linked with side information to re-identify all or some of the users.[8]

[†] Hypothetically, an attacker might be interested in which items have *not* been rated by a user. However, usually in a recommending systems the user ratings are extremely sparse (of the order of few percentage points). Therefore absence of a rating is largely uninformative and can be excluded from the attack.

[‡] While we define this property at the user level, it can also be defined at the item level, to allow more stringent thresholds on sensitive items.

[§] Such an architecture do not need a trusted third party to be implemented.

[¶] http://movielens.org.

[**] https://developer.chrome.com/extensions/pageAction.html.

[††] https://movielens.org.

[‡‡] http://lenskit.org/documentation/algorithms/svd/.

[§§] https://grouplens.org/datasets/movielens/.

[¶¶] github.com/AlessandroChecco/clustering-for-OpenNym.

[***] To keep the perturbation on the online system to a minimum.

[†††] Partially described in https://www.npmjs.com/package/node-movielens.

[‡‡‡] The small size of the test set was necessary not to overburden the very frequent user RMSE computation on the Movielens website.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in Clustering for OpenNym at github.com/AlessandroChecco/clustering-for-OpenNym.

ORCID

Alessandro Checco https://orcid.org/0000-0002-0981-3409

REFERENCES

1. Langheinrich M. To FLoC or not? *IEEE Pervasive Comput*. 2021;20(2):4-6.
2. McSherry F, Mironov I. Differentially private recommender systems: building privacy into the Netflix prize contenders. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY: ACM; 2009;627-636.
3. Wagner I, Eckhoff D. Technical privacy metrics: a systematic survey. *ACM Comput Surv*. 2018;51(3):1-38.

4. Bussard L, Roudier Y, Molva R. Untraceable secret credentials: trust establishment with privacy. *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. Piscataway, NJ: IEEE; 2004;122-126.
5. Quercia D, Hailes S, Capra L. Tata: Towards anonymous trusted authentication. *International Conference on Trust Management*. New York, NY: Springer; 2006;313-323.
6. Crites EC, Lysyanskaya A. Mercurial signatures for variable-length messages. *Proc Priv Enh Technol*. 2021;4:441-463.
7. Aggarwal CC. On k-anonymity and the curse of dimensionality. *Proceedings of the 31st International Conference on Very Large Data Bases*. New York, NY: ACM; 2005;901-909.
8. Narayanan A, Shmatikov V. How to break anonymity of the netflix prize dataset. *arXiv Preprint cs/0610105*. 2006;1–10.
9. Datta A, Sharma D, Sinha A. Provable de-anonymization of large datasets with sparse dimensions. *International Conference on Principles of Security and Trust*. New York, NY: Springer; 2012;229-248.
10. Nikolaenko V, Ioannidis S, Weinsberg U, Joye M, Taft N, Boneh D. Privacy-preserving matrix factorization. *ACM SIGSAC Computer & Communications Security (CCS'13)*. New York, NY: ACM; 2013.
11. Aïmeur E, Brassard G, Fernandez JM, Onana FSM. ALAMBIC: a privacy-preserving recommender system for electronic commerce. *Int J Inf Secur*. 2008;7(5):307-334.
12. Li D, Lv Q, Xia H, Shang L, Lu T, Gu N. Pistis: a privacy-preserving content recommender system for online social communities. *IEEE Int. Conf. on Web Intelligence and Intelligent Agent Technology*. Piscataway, NJ: IEEE; 2011;79-86.
13. Guha S, Cheng B, Privad FP. Practical privacy in online advertising. *8th USENIX NSDI Symposium*. Berkeley, CA: USENIX; 2011.
14. Canny J. Collaborative filtering with privacy. *Proceedings 2002 IEEE Symposium on Security and Privacy*. Piscataway, NJ: IEEE; 2002; 45-57.
15. Badsha S, Yi X, Khalil I, Bertino E. Privacy preserving user-based recommender system. *37th International Conference on Distributed Computing Systems (ICDCS)*. Piscataway, NJ: IEEE; 2017;1074-1083.
16. Kim J, Koo D, Kim Y, Yoon H, Shin J, Kim S. Efficient privacy-preserving matrix factorization for recommendation via fully homomorphic encryption. *ACM Trans Priv Secur*. 2018;21(4):1-30. doi:10.1145/3212509
17. Fu A, Chen Z, Mu Y, Susilo W, Sun Y, Wu J. Cloud-based outsourcing for enabling privacy-preserving large-scale non-negative matrix factorization. *IEEE Trans Serv Comput*. 2019;1-1. doi:10.1109/TSC.2019.2937484
18. Agrawal R, Srikant R. Privacy-preserving data mining. *ACM Sigmod Rec*. 2000;29(2):439-450.
19. Huang Z, Du W, Chen B. Deriving private information from randomized data. *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. New York, NY: ACM; 2005;37-48.
20. Kargupta H, Datta S, Wang Q, Sivakumar K. On the privacy preserving properties of random data perturbation techniques. *Third IEEE International Conference on Data Mining, ICDM 2003*. Piscataway, NJ: IEEE; 2003;99-106.
21. Polat H, Du W. SVD-based collaborative filtering with privacy. *Proceedings of the 2005 ACM Symposium on Applied Computing*. New York, NY: ACM; 2005;791-795.
22. Li Q, Wen Z, Wu Z, Hu S, Wang N, Li Y, Liu X, He B. A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. *IEEE Transactions on Knowledge and Data Engineering*. 2021;1-1. http://dx.doi.org/10.1109/tkde.2021.3124599
23. Chai D, Wang L, Chen K, Yang Q. Secure federated matrix factorization. *IEEE Intell Syst*. 2021;36(5):11-20. doi:10.1109/MIS.2020.3014880
24. Flanagan A, Oyomno W, Grigorievskiy A, Tan KE, Khan SA, Ammad-Ud-Din M. Federated multi-view matrix factorization for personalized recommendations. *arXiv Preprint arXiv:2004.04256*. 2020;1–16.
25. Checco A, Bianchi G, Leith DJ. BLC: private matrix factorization recommenders via automatic group learning. *ACM Trans Priv Secur*. 2017;20(2):4-25.
26. Singel R. *Netflix Spilled your Brokeback Mountain Secret, Lawsuit Claims*. San Francisco, CA: Threat Level (blog), Wired; 2009.
27. Narayanan A, Shmatikov V. Robust de-anonymization of large sparse datasets. *IEEE Symposium on Security and Privacy (sp 2008)*. Piscataway, NJ: IEEE; 2008;111-125.
28. Tóth G, Hornák Z, Vajda F. Measuring anonymity revisited. *Proceedings of the Ninth Nordic Workshop on Secure IT Systems*. Espoo, Finland: Helsinki University of Technology; 2004;85-90.
29. Margolin NB, Levine BN. Quantifying resistance to the sybil attack. *Financial Cryptography and Data Security*. New York, NY: Springer; 2008;1-15.
30. Rowaihy H, Enck W, McDaniel P, La Porta T. Limiting sybil attacks in structured p2p networks. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications*. Piscataway, NJ: IEEE; 2007;2596-2600.
31. Chaum D, Fiat A, Naor M. Untraceable electronic cash. *Proceedings on Advances in Cryptology*. New York, NY: Springer-Verlag New York, Inc; 1990;319-327.
32. Park H, Shin S, Roh B, Lee C. Identification of hosts behind a NAT device utilizing multiple fields of IP and TCP. *2016 International Conference on Information and Communication Technology Convergence (ICTC)*. Piscataway, NJ: IEEE; 2016;484-486.
33. Feghhi S, Leith DJ. An efficient web traffic defence against timing-analysis attacks. *arXiv:1610.07141*. 2018;10(05):557-570.
34. Mohamed Y. Privacy Enhanced Recommendations Using Group Clustering Techniques. Master's thesis. Trinity College Dublin. College Green, Dublin 2, Ireland; 2018.
35. Funk S. *Netflix Update: Try This At Home*; 2006. Accessed December 2006. http://sifter.org/simon/journal/20061211.html.
36. Paterek A. Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*; New York, NY: ACM; 2007;5-8.
37. Ekstrand MD, Kluver D, Harper FM, Konstan JA. Letting users choose recommender algorithms: an experimental study. *Proceedings of the 9th ACM Conference on Recommender Systems*. New York, NY: ACM; 2015;11-18.

38. Yang X, Steck H, Guo Y, Liu Y. On top-k recommendation using social networks. *Proceedings of the Sixth ACM Conference on Recommender Systems*. New York, NY: ACM; 2012;67-74.
39. Fagin R, Kumar R, Sivakumar D. Comparing top k lists. *SIAM J Discret Math*. 2003;17(1):134-160.
40. Koren Y, Bell R, Volinsky C. Matrix factorization techniques for recommender systems. *Computer*. 2009;42(8):30-37.
41. Jain P, Netrapalli P, Sanghavi S. Low-rank matrix completion using alternating minimization. *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. New York, NY: ACM; 2013;665-674.

How to cite this article: Checco A, Bracciale L, Leith DJ, Bianchi G. OpenNym: Privacy preserving recommending via pseudonymous group authentication. *Security and Privacy*. 2021;e201. doi: 10.1002/spy2.201