Matrix Factorization Without User Data Retention

David Vallet, Arik Friedman, and Shlomo Berkovsky

NICTA, Australia
{david.vallet,arik.friedman,shlomo.berkovsky}@nicta.com.au

Abstract. Recommender systems often rely on a centralized storage of user data and there is a growing concern about the misuse of the data. As recent studies have demonstrated, sensitive information could be inferred from user ratings stored by recommender systems. This work presents a novel semi-decentralized variant of the widely-used Matrix Factorization (MF) technique. The proposed approach eliminates the need for retaining user data, such that neither user ratings nor latent user vectors are stored by the recommender. Experimental evaluation indicates that the performance of the proposed approach is close to that of standard MF, and that the gap between the two diminishes as more user data becomes available. Our work paves the way to a new type of MF recommenders, which avoid user data retention, yet are capable of achieving accuracy similar to that of the state-of-the-art recommenders.

Keywords: data retention, matrix factorization, recommender systems

1 Introduction

An increasing number of users are nowadays facing the information deluge, which hinders the discovery of content and services of interest. Recommender systems address this problem through the provision of personalized recommendations to users. Personalization, however, entails another problem: potential leak of sensitive user data [10]. Indeed, accurate recommendations inherently require a large volume of personal user data, which may concern some users to the extent that they refrain from using the recommender despite its benefits [1].

While encryption technologies can mitigate the risk of eavesdropping when data is transferred to the recommender system, the retention of the data by the recommender exposes the users to additional privacy risks. Even seemingly innocuous preference information, like that typically stored by recommender systems, introduces privacy risks, as demonstrated in scenarios such as 1) inference of undisclosed sensitive personal information, e.g., relationship status or sexual orientation, through access to public user data [14, 20, 4, 9]; 2) trading or brokering of personal data to an untrusted third party;[1] or 3) disclosure of sensitive personal information to law enforcement agencies, even without a warrant [6].

[1] http://gizmodo.com/5991070/big-data-brokers-they-know-everything-about-you-and-sell-it-to-the-highest-bidder

These issues fuel a growing concern that motivates keeping personal user data away from a centralized storage, thereby mitigating data disclosure risks.

In this work, we leverage Matrix Factorization (MF) techniques [8] to investigate the use of a semi-decentralized recommender, which retains neither the user ratings nor the latent user vectors. Instead, user ratings are stored on the user side, e.g., on a device owned and carried by the user or on a secure cloud-based service with a personal account. When a user rates a new item or a recommendation is required, the user's ratings are provided to the MF recommender, which derives the latent user vector and updates the latent item matrix. Then, item ratings are predicted and the recommendations are generated. When the interaction with the user is concluded, the recommender discards both the ratings and the latent user vector. This semi-decentralized setting mitigates the above risks by not retaining permanently any user-specific data that could be exploited for the inference of sensitive personal information, while allowing service providers to retain content-specific data, crucial for the provision of recommendations.

We carried out an experimental evaluation, considering scenarios in which the recommender transitions from the centralized to the semi-decentralized model, as well as using the latter from the outset. The results demonstrate that the semi-decentralized model achieves accuracy that is close to that of the centralized recommender, and the gap between the two diminishes over time, as more ratings become available to the system. To summarize, the main contributions of this paper are two-fold. Firstly, we propose a novel semi-decentralized variant of MF without user data retention. Secondly, we evaluate several variants of the algorithm and perform a time-based simulation that demonstrates the applicability of the algorithm to a large-scale real-life MF recommender.

2 Related Work

Several studies focused on the privacy implications of access to user ratings. In the Netflix Prize competition,[2] data anonymization was used to protect user ratings. However, Narayanan and Shmatikov [14] demonstrated that it was possible to de-anonymize the ratings with a little background knowledge, and that sensitive and private information could be inferred from these ratings. In a later work, Calandrino et al. [2] demonstrated that it was possible to infer which items had been rated by a user by sampling changes to aggregated publicly available item recommendations. Weinsberg et al. [20] showed that demographic data could be derived from user ratings. The experiments showed that gender and other sensitive private information (age, ethnicity, political orientation) could be accurately inferred. Chaabane et al. [4] demonstrated a similar problem in information pertaining to users' interests. By accessing supposedly harmless music interests of users, the authors were able to infer sensitive undisclosed information about user gender, age, country, and relationship status. Similarly, Kosinski et al. [9] were able to uncover sensitive information referring to sexual orientation, political alignment, religion, and race, based on Facebook "likes."

[2] http://www.netflixprize.com/

Decentralization of personal user data was previously studied in the context of online advertising, which entails gathering browsing and behavioral data. Adnostic [19] and Privad [7] are two privacy preserving systems, which offer advertisement services while storing private user data on the user side. This is achieved by pushing parts of the advertisement selection process to the client. However, this solution is inapplicable to MF and Collaborative Filtering recommenders, which are too computationally intensive to run on the client. Even if that was feasible, many recommenders would need to push to the user side additional information, such as item descriptions, similarity scores, or latent factors. Not only may this information be deemed too sensitive by service providers, it could also be used maliciously, e.g., for inferring other users' data [2].

Our semi-decentralized variant of MF, which retains no user data, shares similarities with prior works on online MF [16, 15, 18, 5, 12]. These works leveraged incremental re-training of user/item vectors (e.g., through *folding-in* [16]) when new ratings become available, to avoid the cost incurred by running the full factorization process, and improve system scalability and efficiency. In contrast to these approaches, we present a novel solution that relies neither on a centralized storage nor on the availability of past user ratings.

3 Semi-Decentralized MF Recommender

In this section we present how a MF-based recommender can be modified to operate without retaining user ratings, profiles, or latent vectors.

3.1 Preliminaries and Notation

Consider a set U of n users, who assign ratings to a set I of m items. We denote by S the set of available ratings, where each element is a triplet (u, i, r_{ui}), and r_{ui} denotes the rating that user u assigned to item i. We denote by $S_v = \{(u, i, r_{ui}) \in S | u = v\}$ the set of known ratings of user v. The recommendations are generated by predicting the values of unknown ratings. MF achieves this by learning two low-rank matrices with d latent factors: $P_{n\times d}$ for users and $Q_{m\times d}$ for items, where a row p_u in P pertains to user u and a row q_i in Q pertains to item i. The predicted rating of user u for item i is then computed by $\hat{r}_{ui} = p_u q_i^\intercal$. Given the known ratings in S, the latent matrices P and Q are obtained by solving the optimization problem

$$\arg\min_{P,Q} J_S(P, Q), \tag{1}$$

where the loss function $J_S(P, Q)$ with regularization parameter γ is given by

$$J_S(P, Q) := \sum_{r_{ui}\in S} [(r_{ui} - p_u q_i^\intercal)^2 + \gamma(\|p_u\|^2 + \|q_i\|^2)] \ . \tag{2}$$

We consider the Stochastic Gradient Descent (SGD) technique for empirical risk minimization, with a learning rate λ [8]. In this process, P and Q are

```
Algorithm 1: GetRecommendations($S_u$, $Q$)
  Input:
    $S_u$ – ratings of user $u$
    $Q$ – the latent item matrix maintained by the recommender
  User $u$ sends $S_u$ to the recommender
  Given $S_u$ and $Q$, the recommender solves $\arg\min_{p_u} J_{S_u}(p_u, Q)$ to derive $p_u$
  The recommender predicts ratings $\hat{r}_{ui} = p_u q_i^\intercal$ and generates recommendations
  The recommender discards $S_u$ and $p_u$
```

modified for each $r_{ui} \in S$ using the following update rules, which are applied iteratively until both P and Q converge:

$$p_u := p_u + 2\lambda[(r_{ui} - p_u q_i^\intercal)q_i - \gamma p_u] \tag{3}$$

$$q_i := q_i + 2\lambda[(r_{ui} - p_u q_i^\intercal)p_u - \gamma q_i] \tag{4}$$

3.2 Maintaining Decentralized User Profiles

One way to avoid centralized retention of user profiles by a MF recommender is to off-load them to the user side. Predicting unknown scores $\hat{r}_{ui}$ for all unrated items of user u requires the latent user vector p_u and the latent item matrix Q. In a semi-decentralized setting, u can send the user vector p_u to the system, which holds the item matrix Q. The system then computes $\hat{r}_{ui}$ and provides the recommendations to u. Once the interaction with u is concluded, the system discards p_u, as it is not needed for recommendations for other users. This eliminates the need for a permanent centralized retention of all ratings of all users, and mitigates privacy concerns related to misuse of personal data “at rest”.

One drawback of this setting is that the user's p_u may not reflect recent ratings provided by other users. In MF, p_u is affected indirectly by ratings of other users, as new ratings gathered by the recommender affect Q and, in turn, affect vectors of users, who previously rated those items. Fixing latent user vectors and keeping them on the user side hinders these updates of p_u, and can cripple the accuracy of the system. Folding-in techniques [16] circumvent this drawback, using an up-to-date matrix Q to re-evaluate the user vector p_u by solving

$$\arg\min_{p_u} J_{S_u}(p_u, Q) \quad . \tag{5}$$

This optimization problem can be solved with the SGD technique by fixing Q and only applying update rule (3). Algorithm 1 shows the modified semi-decentralized recommendation process that recomputes p_u from S_u.

3.3 Maintaining the Item Matrix

In addition to generating recommendations, the system needs to update the latent item matrix Q based on new ratings. In a standard MF setting, the user and item vectors are typically updated simultaneously. But this cannot be done

```
Algorithm 2: UpdateItem($S_u$, $r_{ui}$, $Q$, $\gamma$, $\lambda$)
  Input:
      $S_u$ – past ratings of user $u$
      $r_{ui}$ – new rating from user $u$
      $Q$ – the item-factor matrix maintained by the recommender
      $\gamma$ – regularizer
      $\lambda$ – learning rate
1:  User $u$ sends $r_{ui}$ and $S_u$ to the recommender
2:  $S_u := S_u \cup r_{ui}$
3:  Given $S_u$ and $Q$, the recommender solves $\arg\min_{p_u} J_{S_u}(p_u, Q)$ to derive $p_u$.
4a: $q_i := q_i + 2\lambda[(r_{ui} - p_u q_i^\intercal)p_u - \gamma q_i]$  OR  4b: for each $r_{uj} \in S_u$ do
        $q_j := q_j + 2\lambda[(r_{uj} - p_u q_j^\intercal)p_u - \gamma q_j]$
5:  The system discards $p_u$ and $S_u$.
```

in the semi-decentralized setting, as no user ratings are retained. Alternatively, following the approaches outlined in [15, 16], folding-in can be applied to update the item vectors by fixing the user matrix P, and retraining an item vector q_i through applying SGD and update rule (4) to all the available ratings. While effective, this iterative update of item factors is also inapplicable unless all the ratings for i are available, and another solution is needed.

Algorithm 2 presents a modified process for updating Q that leverages the fact that for each new rating r_{ui}, the update of q_i requires access only to the vector p_u of the user who provided the rating. First, the user u sends r_{ui} along with previous ratings in S_u to the system, which reconstructs p_u. Then, update rule (4) is applied to the newly added rating and, finally, the system keeps the updated latent item vector and discards both p_u and S_u.

We consider two variants of this scheme. In the first variant, denoted by *new*, we perform an update only with the newly added rating r_{ui}, i.e., apply the update in instruction 4a of Algorithm 2. In the second variant, denoted by *rated*, we take advantage of the availability of the entire set of ratings S_u and apply an iterative update rule in instruction 4b of Algorithm 2 to the item vectors of all the items that were rated by u. We note that unlike the *new* variant, which updates one latent vector for every new rating, the *rated* variant updates the vectors of all previously rated items, which may bias the latent item matrix Q towards ratings of highly active users, who provided many ratings. However, this bias can be mitigated, e.g., by weighting the update according to the number of ratings that a user provided.

Note that the semi-decentralized setting can be extended to support more complex variants of MF. For example, a common practice is to incorporate user bias $b_u = \sum_{r_{ui} \in S_u} r_{ui}/|S_u|$ to normalize user ratings. The user bias is discounted from the gathered ratings prior to computing P and Q, and re-introduced when the predictions are computed: $\hat{r_{u,i}} = b_u + p_u q_i^\intercal$. This bias can be stored at the user side, or be directly recomputed by the system from the gathered ratings, similarly to the user latent vector p_u.

4 Experimental Evaluation

In order to evaluate the proposed semi-decentralized variants of MF, we consider the following scenario. The recommender collects user ratings until a cut-off time t_s, after which it transitions to the semi-decentralized model. All the data collected before t_s is considered as the initial *static* data. Then, the recommender retains the latent item matrix Q_s derived at t_s and discards the latent user matrix P_s. As discussed in Section 3, from time t_s and onwards the recommender retains neither the user ratings nor the latent user vectors.

The item matrix Q, initialized to Q_s, is updated as outlined in Algorithm 2, when new ratings become available. We posit that the resulting Q will deviate from the hypothetical Q^* that could be derived if the recommender retained user ratings as in the centralized MF. That is, the proposed approach can only approximate Q^*, since not all user ratings are available when Q is updated. A "good" approach for updating Q will be one that generates predictions as close as possible to those of the centralized MF that continuously updates P and Q.

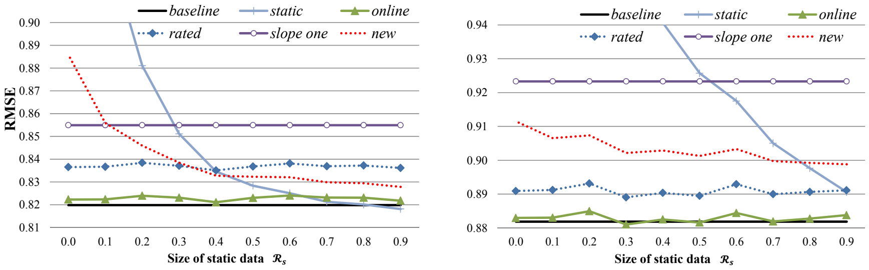
We denote by $\mathcal{R}_s$ the ratio between the number of ratings collected before time t_s, i.e., ratings used to derive Q_s, and the overall number of ratings processed in the evaluation. $\mathcal{R}_s$ represents the relative volume of the static initialization data. A special case of $\mathcal{R}_s = 0$ reflects a scenario in which the semi-decentralized recommender has no prior user data and starts from the outset with a randomly initialized Q.

In the evaluation, we use two public datasets: MovieLens and Netflix. Since the temporal information of MovieLens ratings was deemed unreliable (many ratings had the same timestamp), we use MovieLens to produce a random split of data and Netflix to produce a time-based split. Specifically, we use the Movielens 10M dataset to perform a 10-fold cross validation for randomly split non-overlapping training and test sets. For the Netflix dataset, we randomly sample 10% of users to obtain a comparable dataset of 10M ratings. For the time-based split we select the most recent 10% of ratings given by each user as the test set. We use RMSE to assess the performance of the proposed approach and the Wilcoxon's signed-rank test to validate statistical significance [17].

We tune the MF parameters in an offline evaluation that is omitted due to space limitation. We set the number of factors to $d = 10$. When training with static data, we use SGD with regularization factor $\gamma = 0.1$, learning rate $\lambda = 0.01$, and $n = 200$ iterations. We use the user and item biases as presented in Section 3.3. When deriving p_u (Equation 5), we run $n = 10$ iterations of SGD. When updating Q (algorithm 2), the learning rate is set to $\lambda = 0.005$ and $\lambda = 0.05$ for the *rated* and *new* approaches, respectively.

4.1 Overall Performance

In this section, we assess which of the proposed variants of the semi-decentralized MF performs better and how they compare to the state-of-the-art MF that retains user data.



(a) MovieLens 10M, random split (b) Netflix 10M, time-based split

Fig. 1: Comparison between the semi-decentralized MF approach and the state-of-the-art. The differences are significant at $p < 0.01$.

Figures 1a and 1b depict the accuracy achieved by the recommenders for the random and time-based split of training and test data, respectively. We compare the two proposed variants of the semi-decentralized MF, which update the latent item vectors of either all the rated items (*rated*) or of the recently rated items only (*new*), to four baseline approaches:

Baseline Standard centralized implementation of MF, using SGD and retaining all the available ratings [8].

Static Similar to baseline, but using only the static ratings available at t_s, i.e., only Q_s and P_s, to generate recommendations. This mimics a "lazy" approach, where the recommender disregards new ratings and does not update P and Q after t_s.

Online Centralized online MF proposed by Rendle and Schmidt-Thieme in [15], which allows an incremental update of Q by iterating over all the stored ratings. We tuned the parameters using an offline evaluation to $d = 10$, $\lambda = 0.01$, and $n_{iter} = 200$.

Slope-one A simplified item-based Collaborative Filtering (CF) that assumes a linear relationship between the items [11]. This algorithm provides a realistic lower bound for acceptable performance of a personalized recommender. This approach also retains all the available user ratings.

We observe that for the random split (Figure 1a), the *new* variant outperforms the *rated* variant for $\mathcal{R}_s \geq 0.4$, but its error is substantially higher for lower ratios of static data. This is due to the fact that *new* introduces fewer updates to the reasonably stable Q_s than *rated*, and, just like the *static* approach that does not update Q at all, yields good performance for high values of $\mathcal{R}_s$. Conversely, the *rated* variant, which entails more updates, results in a stronger deviation of Q from Q_s that has already stabilized for high $\mathcal{R}_s$, and this leads to less accurate predictions. The performance of the *rated* variant is, however, superior to that of *new* and *static* for lower ratios of $\mathcal{R}_s$, due to the repetitive vector updates of the rated items.

In the more realistic time-based split scenario portrayed in Figure 1b, *rated* outperforms *new* across the board: the repetitive updates of the latent vectors of all the rated items in S_u keep Q closer to Q^* as rating of users are processed

Dataset-Split	*static*	*rated*	*new*	*online*	*slope one*
MovieLens-Random	3.22%	2.09%	1.98%	0.37%	4.28%
Netflix-Time	6.75%	1.02%	2.33%	0.12%	4.70%

Table 1: Difference between the evaluated approaches and the MF baseline averaged for $\mathcal{R}_s \in [0, 0.9]$.

over time. In contrast, the more conservative update process of *new* performs worse when ratings are fed in temporal order, since it is slower to incorporate ratings of new items, which are more important to accurately predict ratings in this setup (this is also shown in the *static* approach).[3] We conclude that, when taking a realistically behaving temporal factor of ratings into consideration, the *rated* variant is the best performing semi-decentralized MF approach, and its performance is stable across various values of $\mathcal{R}_s$.

As expected, standard centralized MF achieves the highest accuracy. However, MF is not sufficiently scalable and cannot be deployed in a practical Web-based recommender, since it requires the latent vectors to be re-trained for every new rating. In the following analysis, we consider the RMSE of standard MF as the lower bound for error achievable by other approaches and refer to its accuracy as the baseline.

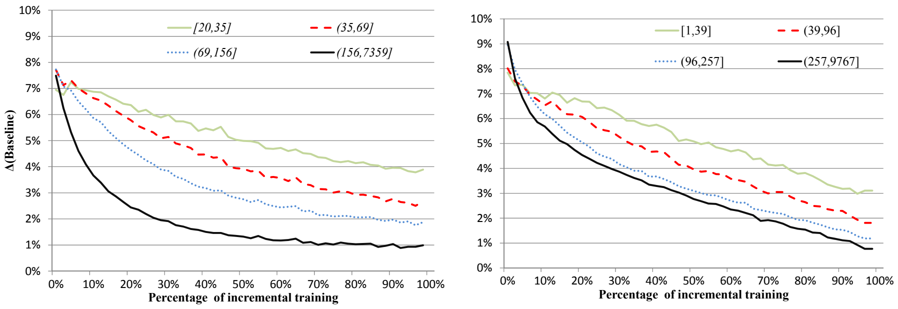
Table 1 summarizes the performance of the evaluated approaches with respect to the centralized MF baseline. The performance is quantified by averaging the errors obtained for ten values of $\mathcal{R}_s$, ranging from 0 to 0.9. As can be seen, the average error of the *online* approach is only 0.12% higher than that of the MF baseline for the realistic time-based split. The *rated* and *new* semi-decentralized variants are inferior to MF by 1.02% and 2.33%, respectively. Finally, *slope one* and *static* demonstrate substantially higher error rates than MF.

Out of the two semi-decentralized MF variants, *rated* again outperforms *new* and comes closer to the baseline MF. *Online* is very close to MF and is consistently superior to our best performing *rated* variant. However, it should be noted that *online* benefits from access to all the available ratings and can re-train the latent factors, which explains its performance. Placing *rated* on the range between the upper bound of accuracy set by the baseline MF and the lower bound set by *slope one*, we highlight that *rated* is much closer to MF than to *slope one*. We conclude that *rated* offers a reasonable compromise, as it performs online updates of the latent vectors, without retaining user data.

4.2 Impact of the Availability of User Ratings

In this section we investigate how the number of available user ratings affects the performance of the proposed approach. We do this by measuring the fluctuations in the accuracy of the semi-decentralized *rated* variant, as a function of the volume of incrementally added training data. We fix the static data ratio to

[3] Random split of Netflix yielded results similar to those in Figure 1a, highlighting the importance of the temporal data. Due to space limitations we excluded this chart.



(a) MovieLens 10M, random split

(b) Netflix 10M, time-based split

Fig. 2: Relative difference between the *rated* variant and the MF baseline as a function of the volume of the incremental data. Each line pertains to a set of users with a given number of ratings. The differences are significant at $p < 0.01$.

$\mathcal{R}_s = 0.3$, and then gradually increase the number of ratings added on top of the static data from 0% to 100% of the remaining training ratings. We split the users into four equal-sized buckets according to the number of their ratings, and average the user-based accuracy of the *rated* variant across the buckets.

Figures 2a and 2b show the relative difference between the RMSE of the *rated* variant and that of the baseline MF, respectively for the random split and the time-based split, both averaged on a bucket basis for an increasing number of incrementally added ratings. Generally, the performance of *rated* converges to MF as more ratings become available. For the random split, there is an evident difference between the buckets, and the difference with respect to MF gets smaller with the number of user ratings. This is expected, as predictions based on many ratings are generally more accurate than those based on a few ratings. The convergence rate observed in the temporal split evaluation is slower than that observed in the random split evaluation, since the temporal split emphasizes prediction of ratings for new items, and in the early stages of the evaluation there is no sufficient training data pertaining to new items to perform such predictions accurately. The difference between the buckets is less pronounced for the time-based split, and the behavior of the top two buckets, corresponding to users with more than 96 ratings, is similar.

Hence, we conclude that when users have a moderate amount of ratings, the prediction accuracy converges to the baseline MF, while users with fewer ratings will suffer a slight impact on performance.

4.3 Time-Based Simulation

Finally, we investigate the performance of the proposed approach in a realistic large-scale time-aware scenario. We perform this experiment using the complete Netflix 100M ratings dataset, to mimic the evolution of a Web-scale recommender system that transitions to the semi-decentralized MF model. We compare the accuracy of the two semi-decentralized variants that do not retain user data

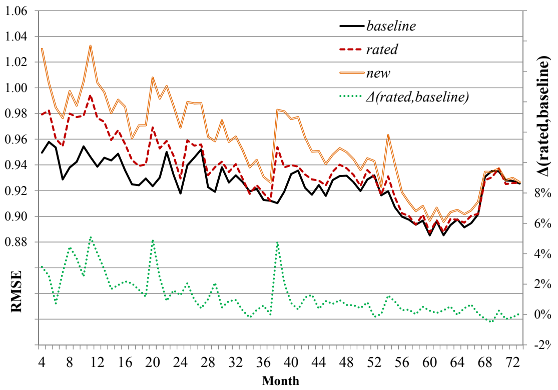


Fig. 3: Difference between the *rated* and *new* variants and the MF baseline. The bottom line shows the difference between *rated* and MF.

with the baseline MF. In this experiment, we sort all the available Netflix ratings according to their timestamps.[4] We then use the increasing-time window evaluation methodology with a window size of 1 month [3]. That is, we use the data of the first $n-1$ months as the training set, predict the ratings of the n-th month, and compute the RMSE for that month. Then, we add the data of the n-th month to the training set and predict the ratings of the $n+1$-th month, and so on. The overall span of Netflix ratings is 72 months and we initialize the training set with the first four months of data.

Figure 3 shows the RMSE of the *new*, *rated*, and MF recommenders (values on the left axis), as well as the relative difference between the *rated* variant and the MF baseline (right axis). Initially (months 4 to 24), the difference hovers around the 2% mark, and it steadily diminishes later on, as the training window size increases. Eventually, the difference becomes smaller than 1% around month 55 and virtually disappears from month 68 onward.[5] We note some lose of accuracy (spikes in the relative difference) up to months 38-40. These spikes are due to the introduction of a large number of new items into the system paired with a lower number of users in the system. This makes our algorithms more prone to error, as their update process is slower than that of the baseline (the *new* approach has a higher spike due to an even slower update cycle).

This large-scale simulation clearly demonstrates that the performance of the proposed approaches comes very close to that of the baseline MF, as more ratings become available to the recommender.[6] This result is encouraging, as it shows that although avoiding user data retention comes at the cost of accuracy, the cost diminishes over time. Also, while *rated* outperforms *new* at most stages

[4] As the timestamps of Netflix rating are truncated to days, we randomize all the ratings with the same timestamp.

[5] Similar results are obtained in an experiment using the increasing dataset size methodology with quanta of 1M ratings (ratings are sorted, and chunks of 1M ratings are predicted and then added). Due to space limitations we exclude this chart.

[6] The processing of ratings (re-computing the factors and the recommendations) was done at a rate of 900 ratings/second, using one core of AMD Opteron 4184.

of the simulation, their performance converges at the later stages, as more ratings are included, implying that *new* could also be viable when enough training data is available. Overall, this simulation demonstrates that the proposed semi-decentralized MF variant can be deployed in a practical Web-scale recommender.

5 Conclusions and Discussion

In this work we present a novel semi-decentralized variant of MF that retains neither user ratings nor user latent factors. This way, we mitigate the risk of undesired access to personal data by malicious third parties and hinder the use of the data for purposes other than the recommendations. Our evaluation indicates that the proposed approach performs on a par with the state-of-the-art MF approaches, thus offering a viable alternative and moving a step forward towards privacy-aware recommender systems. At the same time, our approach allows service providers to keep control of their domain knowledge, encapsulated in the latent item factors matrix.

Note that even if the service providers avoid user data retention, this does not prevent possible leakages of item-related data, which is retained by the service providers. While item-related data does not contribute directly to user privacy risks, it could still be abused to derive personal information, as implied by Calandrino et al. [2], and addressing this risk would require the use of complementing privacy mechanisms, such as differential privacy [13]. However, when compared to accessing readily-available user data, such attacks require more effort, depend on access to auxiliary data, and are harder to execute at scale.

The mitigation of privacy risks relies on the service provider discarding user data. One could argue that no privacy gain is achieved, as user ratings are still accessible to service providers when the recommendations are generated and the users have no means to verify that their data is discarded. Furthermore, since the recommendations requires access to all the user's ratings, service providers could still mine personal information when the ratings are sent to the system. However, such misconduct may be detrimental to user trust and to the reputation of the service. While service providers need to balance user privacy with the benefits of acquiring personal data, it is worth to note the increased awareness of privacy, such that privacy is often advocated as a competitive advantage of services.[7] This incentivizes service providers to conform to privacy regulations and to embrace privacy preserving algorithms. We also note that our approach does not preclude law enforcement agencies from activating "wiretaps" that record the data of certain users. In fact, as made evident in the mandatory data retention initiative,[8] and the recent revelations about the NSA and the PRISM program [6], service providers may be forced to put such mechanisms in place. However, we posit that requiring an explicit wiretap activation, as opposed to grabbing data that is already stored in the system, raises the bar for user privacy,

[7] http://techcrunch.com/2013/04/22/microsoft-launches-new-online-privacy-awareness-campaign/

[8] https://www.eff.org/issues/mandatory-data-retention

and reduces the risk of users being subject to passive surveillance. In addition, cryptographic techniques could be leveraged for maintaining the confidentiality of user input, and pose an interesting future research direction.

References

1. S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *RecSys*, pages 9–16, 2007.
2. J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. "you might also like:" privacy risks of collaborative filtering. In *IEEE Symposium on Security and Privacy*, pages 231–246, 2011.
3. P. Campos, F. Dez, and I. Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *UMUAI*, 2013.
4. A. Chaabane, G. Acs, and M. Kaafar. You are what you like! information leakage through users' interests. In *Proc. NDSS*, 2012.
5. P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, pages 39–46, 2010.
6. G. Greenwald and E. MacAskill. NSA Prism program taps in to user data of Apple, Google and others. *The Guardian*, June 2013.
7. S. Guha, A. Reznichenko, K. Tang, H. Haddadi, and P. Francis. Serving ads from localhost for performance, privacy, and profit. In *HotNets*, 2009.
8. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.
9. M. Kosinski, D. Stillwell, and T. Graepel. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences*, March 2013.
10. S. K. Lam, D. Frankowski, and J. Riedl. Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. In *ETRICS*, pages 14–29, 2006.
11. D. Lemire and A. Maclachlan. Slope one predictors for online rating-based collaborative filtering. *Society for Industrial Mathematics*, 5:471–480, 2005.
12. X. Luo, Y. Xia, and Q. Zhu. Incremental collaborative filtering recommender based on regularized matrix factorization. *Know.-Based Syst.*, 27:271–280, Mar. 2012.
13. F. McSherry and I. Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *KDD*, pages 627–636, 2009.
14. A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, pages 111–125, 2008.
15. S. Rendle and L. Schmidt-Thieme. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *RecSys*, 2008.
16. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28, 2002.
17. G. Shani and A. Gunawardana. Evaluating recommendation systems. In *Recommender Systems Handbook*, pages 257–297. Springer, 2011.
18. G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *JMLR*, 10:623–656, 2009.
19. V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *NDSS*, 2010.
20. U. Weinsberg, S. Bhagat, S. Ioannidis, and N. Taft. BlurMe: inferring and obfuscating user gender based on ratings. In *RecSys*, pages 195–202, 2012.