SoCRATe: a Framework for Compensating Users Over Time with Limited Availability Recommendations

(Discussion Paper)

Davide Azzalini[1], Fabio Azzalini[1], Chiara Criscuolo[1], Tommaso Dolci[1], Davide Martinenghi[1], Letizia Tanca[1] and Sihem Amer-Yahia[2]

[1]*Politecnico di Milano – Dipartimento di Elettronica, Informazione e Bioingegneria*
[2]*Centre National de Recherche Scientifique (CNRS) – Université Grenoble Alpes*

Abstract
We present our preliminary ideas for developing SoCRATe, a framework and an online system dedicated to providing recommendations to users when items' availability is limited. SoCRATe is relevant to several real-world applications, among which movie and task recommendations. SoCRATe has several appealing features: it watches users as they consume recommendations and accounts for user feedback in refining recommendations in the next round, it implements loss compensation strategies to make up for sub-optimal recommendations, in terms of accuracy, when items have limited availability, and it decides when to re-generate recommendations on a need-based fashion. SoCRATe accommodates real users as well as simulated users to enable testing multiple recommendation choice models. To frame evaluation, SoCRATe introduces a new set of measures that capture recommendation accuracy over time as well as throughput and user satisfaction. All these features make SoCRATe unique and able to adapt recommendations to user preferences in a resource-limited setting.

Keywords
Recommender Systems, Dynamic User Preferences, Compensation Strategies

1. Introduction

The goal of a recommendation system is to produce a set of items that are highly relevant to a user. A typical recommendation workflow takes historical user data, applies a strategy that produces relevant recommendations, and measures accuracy [1, 2, 3, 4, 5, 6, 7, 8]. The decision of refining the recommendation logic is usually made offline based on how well a given strategy performs, and based on the users' explicit and implicit feedback. In this paper, we advocate for an iterative approach that benefits from observing users as they consume proposed items, and creating a feedback loop to adapt recommendations to individual users on the fly in the next iteration. This is particularly important when items have limited availability such as product recommendation where only a fixed number of copies of an item is available, and in crowdsourcing where tasks are to be completed by a fixed number of workers. We are developing SoCRATe, a framework and an online system dedicated to providing adaptive

SEBD 2022: The 30th Italian Symposium on Advanced Database Systems, June 19-22, 2022, Tirrenia (PI), Italy
davide.azzalini@polimi.it (D. Azzalini); fabio.azzalini@polimi.it (F. Azzalini); chiara.criscuolo@polimi.it (C. Criscuolo); tommaso.dolci@polimi.it (T. Dolci); davide.martinenghi@polimi.it (D. Martinenghi); letizia.tanca@polimi.it (L. Tanca); sihem.amer-yahia@cnrs.fr (S. Amer-Yahia)

recommendations to users. SoCRATe is applicable to both real and simulated users, and is relevant to various applications ranging from movie recommendation to crowdsourcing.

1.1. Challenges

Formalizing adaptive recommendations and developing an online system that enables them must address several new challenges. The first challenge **C1** is to model user behavior, i.e., how users consume recommendations, and capture both explicit and implicit feedback. Explicit feedback represents the actions taken by the users when provided with recommendations. Examples of actions include rating movies, and completing a task. When users are simulated, different assumptions can be made to reflect how they choose their recommendations. When users are real, their explicit choice of items out of recommended ones needs to be captured. Additionally, several implicit signals can be represented, including the order and speed at which they consume recommendations. The second challenge **C2** is to determine at what moment and for which users recommendations are re-optimized based on their behavior and feedback. The third challenge **C3** is to leverage the observed feedback to refine recommendations in the next iteration. This challenge is coupled with the need to be adaptive regardless of the recommendation strategy. This allows us to deploy a generic solution on top of any recommender. It also raises the question of managing how users compete on recommendations. Indeed, when the number of instances of an item is limited (referred to as *availability*), some users may not get an item even if it is in their top choices. This may happen, e.g., in book recommendation, in case a limited number of copies of a physical book exists, or in crowdsourcing, where the number of workers who complete the same task is limited. Addressing these challenges will enable us to develop SoCRATe, a framework and a system to produce recommendations, evaluate them, and refine them on the fly in order to comply with user behavior.

1.2. Contributions

SoCRATe has several appealing features: it watches recommendation consumption and decides when to apply a loss compensation strategy to make up for sub-optimal recommendations due to limited item availability.

To address **C1**, we introduce an objective function in the form of a linear combination of several dimensions that capture users' strategic behaviors when consuming recommendations. For instance, when presented with a list of movie recommendations, users may choose highly relevant items (utility-based) while also favoring items that are different from each other (diversity). In the case of crowdsourcing, users may pick tasks that are relevant to their profile (utility-based) while also looking for tasks posted by different requesters (diversity) and with high compensation (payment). We devise a generic objective function where each dimension receives a weight according to a user's choice model. When users are simulated, SoCRATe assumes that they choose k out of N provided recommendations according to one of three common choice models [9, 10, 11]: *random*, where users are assumed to choose items randomly; *utility-based*, where users are assumed to choose items in decreasing order of their utility wrt their profile; and *rank-based*, where users are assumed to choose their items in the order in which they are ranked by the objective function.

To address **C2**, we develop three temporal granularities: *fixed*, where recommendations of all users are re-optimized at fixed time periods; *single-user*, where the best moment for re-optimizing recommendations is determined for each user separately as a function of when the user is done and ready to receive new recommendations; and *user-group*, where the best moment is determined for a group of users as a function of when they are all ready.

To address **C3**, SoCRATe captures each user's feedback as a weight vector, where each entry represents the preference of a user for an optimization dimension. This representation is generic enough to capture a variety of applications. To refine weight vectors, SoCRATe runs a regression that compares the k chosen recommendations against the remaining $N - k$. Refined weight vectors are used to re-rank the recommendations produced for each user to match their feedback. Due to limited item availabilities in some applications, users may compete for some items and not receive their optimal recommendations. Hence, we model the loss for a user as the difference between the proposed recommendations and the optimal recommendations that would have been returned to the user if other users were not considered. This loss is leveraged in the next iteration to compensate users, either individually or in groups according to the temporal granularity. We develop two loss compensation strategies: *preference-driven*, where users are compensated in decreasing loss order by receiving recommendations in preference order before the user with the next smaller loss is served, and *round-robin*, where users are considered in decreasing loss order but items are recommended to them in round-robin with respect to item availability.

2. Related work

Our work relates to sequential recommendations [12, 13, 14] where the order in which items are consumed impacts the prediction of the next items. Our work also relates to session-based recommendations that aim to capture short-term dynamic user feedback to provide more timely and accurate recommendations [15]. Sequential recommenders differ from session-based recommenders in that the order of individual items matters and is used to predict the next item a user is most likely to consume. Our work differs from sequential and session-based recommendations in more than one way: we introduce dynamic time splitting and compensation strategies to make up for the loss incurred in previous sessions; we combine historical user data with dynamic user feedback based on various choice models.

Our work also relates to observing users and modeling an objective function accordingly. We propose to generalize the work in [16], whose purpose was to quantify workers' motivation on a crowdsourcing platform by observing their task adoption in multiple sessions. Our generalization consists in capturing users' feedback in a variety of applications by contrasting users' choices of recommended items against all available recommendations. Multiple machine learning approaches could be used to do that, including a simple regression on individual optimization dimensions.

3. Framework of SoCRATe

We present the data model of SoCRATe and the problem variants we plan to tackle.

3.1. Data model of SoCRATe

We are given a set of users U and a set of items I. Each item $i \in I$ has an availability, $avail(i, t)$, that represents the number of copies left of item i at time t. At a given timestamp t, a list of N recommendations $R_u^t \subseteq I$ is provided to a user $u \in U$. The list of optimal recommendations, in terms of accuracy, a user u could get at time t is referred to as $OptR_u^t$. Due to the presence of other users and limited item availability, R_u^t may differ from $OptR_u^t$. When user u receives a list of recommendations R_u^t, u may choose to adopt a list $S_u^t \subseteq R_u^t$ of size $k_u \leq N$ to consume. The user choice incurs the computation of a weight vector w_u^t that represents the importance the user gives to the dimensions O of an objective function f. The lists R_u^t and $OptR_u^t$ are used to compute $loss_u^t$, a vector of losses (one entry per dimension in O) for user u at time t.

3.2. Problem variants in SoCRATe

Our framework raises several problem variants we plan to tackle in the future. We may formulate the problem of determining the best time granularity and compensation strategy that reduce cumulated loss. We may formulate another problem where the optimization goal is to reduce users' idling time. We may again formulate another problem where the goal is to optimize item throughput. All these problem variants give rise to hard problems for which efficient greedy solutions will be designed.

For instance, the problem of refining recommendations in such a way that previously incurred loss is minimized can be expressed as: for each user, find a list R_u^t, s.t.:

$$argmin \sum_{u \in U} w_u^{t-1} \times loss_u^{t-1}$$

Our problem is a variant of the Knapsack Problem [17]. Each item has a value v that corresponds to its contribution to the objective function and a weight (in our case the weight vector w_u^{t-1} of the user), and we want to find N items for each user that maximize the sum of values $\sum v_i$ under a capacity constraint N. Additionally, as the value of recommending an item to a user evolves over time as other users consume items, we need to account for that dynamicity. We will see in Section 4.2 how we solve this problem with greedy solutions.

4. The SoCRATe system

4.1. Architecture of SoCRATe

The architecture of SoCRATe is provided in Figure 1. The ORCHESTRATOR (1) is in charge of determining the moment at which recommendations are re-generated for individuals or for user groups. It monitors when users are done consuming recommendations from the previous iteration and uses that to determine when to re-generate recommendations and start the next iteration. It implements our three time granularity splits: *fixed*, *single-user*, and *user-group*. It makes use of the recommendations adopted by users S_u^t (2), and triggers the WEIGHT UPDATE (3) module to obtain the new weights. Weight computation admits S_u^t, R_u^{t-1}, $OptR_u^{t-1}$ and returns the updated weights W_u^t. Following that, the INDIVIDUAL RECOMMENDER module (4) is

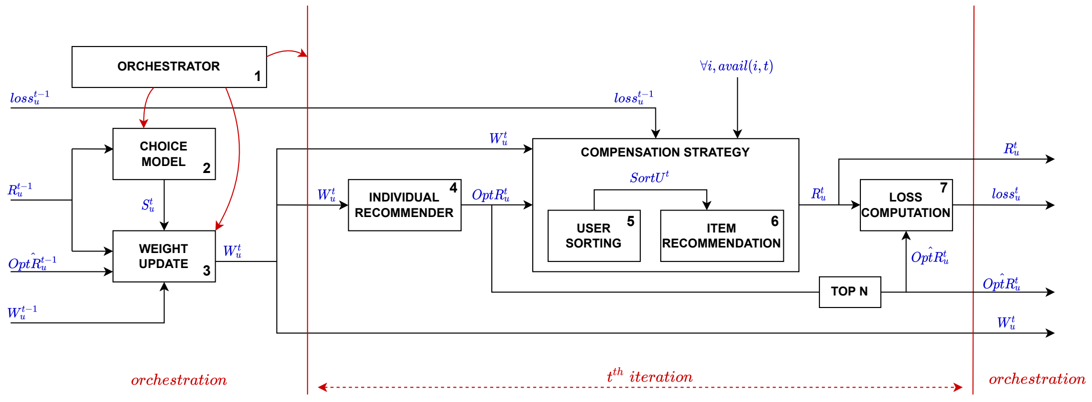

Figure 1: Architecture of SoCRATe

called to produce new recommendations based on the weights associated to each user. This module returns $OptR_u^t$, which is fed to the Compensation Strategy module. Here the User Sorting module (5) sorts users according to their loss and calls the Item Recommendation module (6). This module implements two strategies: *preference-driven* and *round-robin*. The output is a set of new recommendations that are fed to the Loss Computation module (7) to compute each user's incurred loss. SoCRATe then iterates by calling the Orchestrator.

4.2. Algorithms of SoCRATe

Algorithm 1 is generic. It contains the main steps of our solution regardless of the problem we tackle (Section 3.2). It is illustrated for *fixed* time, the same applies to *single-user* and *user-groups*. It takes as input t, the moment where a new iteration starts according to the Orchestrator, and for each user u, $loss_u^{t-1}, R_u^{t-1}, OptR_u^{t-1}, W_u^{t-1}$. It outputs for each user $loss_u^t$, R_u^t, $OptR_u^t$, W_u^t. The line numbers in the overall algorithm reflect the module numbers in Figure 1).

Algorithm 1 SoCRATe Algorithm

```
Input: $\forall u, loss_u^{t-1}, R_u^{t-1}, OptR_u^{t-1}, W_u^{t-1}$      Output: $\forall u, loss_u^t, R_u^t, OptR_u^t, W_u^t$
1: for all $u \in U$ do
2:     $S_u^t \leftarrow$ ChoiceModel($R_u^{t-1}$)
3:     $W_u^t \leftarrow$ WeightUpdate($OptR_u^{t-1}, R_u^{t-1}, W_u^{t-1}, S_u^t$)
4:     $OptR_u^t \leftarrow$ IndividualRecommender($W_u^t$)
5:     $SortU^t \leftarrow$ UserSorting($W_u^t, loss_u^{t-1},$)
6:     $R_u^t \leftarrow$ ItemRecommendation($SortU^t, OptR_u^t, \forall i, avail(i,t)$)
7:     $loss_u^t \leftarrow$ LossComputation($OptR_u^t, R_u^t$)
8: end for
```

We now focus on lines 5 and 6 which correspond to the compensation strategy. Each strategy receives a list of users sorted in decreasing loss, $SortU^t$, the optimal recommendations, in terms of accuracy, for each user, $OptR_u^t$, the availability for each item, $avail(i,t)$, and produces a list of recommended items for each user, R_u^t.

When the preference-based compensation strategy is adopted (see Algorithm 2), N items are recommended to the first user in the sorted list of users before moving to the next one in the list. The round-robin variant (see Algorithm 3) recommends one item at a time to each user (according to the order dictated by the sorted list of users), until N items are recommended to each user.

Choosing the right compensation strategy is crucial when there is a limited set of items that are very popular among all users and when the availability of these items is limited and not sufficient to satisfy all the users. The preference-based compensation strategy is expected to be more effective at maximizing the satisfaction of the first users in the list. As a result, this strategy should be preferred when the system detects that some users are unsatisfied with the recommendations they are getting, and thus may be prone to abandon the system, thereby affecting overall user retention. On the other hand, the round-robin compensation strategy should be preferred when the main concern is to consume as many items as possible. In fact, when adopting this compensation strategy, it is very likely that all users get at least a few items they like, which will affect their consumption rate.

Algorithm 2 Preference-Based Strategy

```
Input: $SortU^t, OptR^t_u, avail(i,t)$
 1: for all $u \in SortU^t$ do
 2:     for $n \leftarrow 1$ to $N$ do
 3:         $i \leftarrow \text{pop}(OptR^t_u)$
 4:         while $i$ is available do
 5:             $i \leftarrow \text{pop}(OptR^t_u)$
 6:         end while
 7:         $R^t_u \leftarrow R^t_u \cup i$
 8:         $avail(i,t+1) \leftarrow avail(i,t) - 1$
 9:     end for
10: end for
11: return $\forall u, R^t_u$
```

Algorithm 3 Round-Robin Strategy

```
Input: $SortU^t, OptR^t_u, avail(i,t)$
 1: for $n \leftarrow 1$ to $N$ do
 2:     for all $u \in SortU^t$ do
 3:         $i \leftarrow \text{pop}(OptR^t_u)$
 4:         while $i$ is available do
 5:             $i \leftarrow \text{pop}(OptR^t_u)$
 6:         end while
 7:         $R^t_u \leftarrow R^t_u \cup i$
 8:         $avail(i,t+1) \leftarrow avail(i,t) - 1$
 9:     end for
10: end for
11: return $\forall u, R^t_u$
```

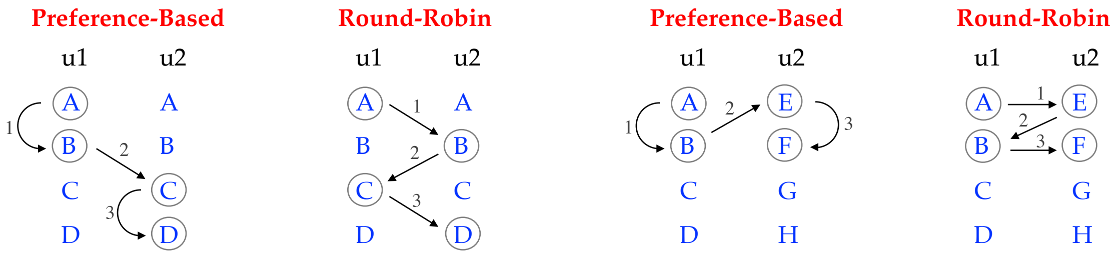

Figure 2: Illustration of our compensation strategies. The list of optimal recommendations of each user is shown in decreasing relevance. The numbers on arrows reflect the order on which items will be recommended. On the left, the two users compete as they have similar recommendations. On the right, the two users do not compete

The example in Figure 2 illustrates the difference between our compensation strategies. In the left part, users have similar recommendations (i.e., similar item rankings) and item availability is low (= 1). In the right part, their recommendations differ. We can see that on the left, preference-based favors u_1 and round-robin treats u_1 and u_2 similarly. On the right, the two compensation strategies behave similarly. A deeper exploration of compensation is therefore warranted.

5. Perspectives

We presented our plan to develop SoCRATe, a framework and online system that watches users as they choose and consume items and refines upcoming recommendations accordingly. SoCRATe raises three novel challenges: how to model user behavior, when and for whom the recommendations should be re-generated, and how to leverage user feedback to refine recommendations and compensate users for the loss incurred in previous iterations. In this paper, we describe our preliminary solution, including a formalization, an architecture, a set of algorithms, and a revised set of evaluation measures to address our challenges.

Our work opens many new perspectives. First, we plan to formalize a series of optimization problems following the discussion in Section 3.2. In doing so, we will strive to preserve the genericity of our solution (Algorithm 1). That will allow us to explore new opportunities in addressing challenges **C2** and **C3**. One compensation strategy we want to formalize is to account for consumer fairness by adding notions such as statistical parity [18] or disparate impact [19], two mitigation procedures that are currently being explored in recommendation systems.

Second, we have started implementing a first in-memory version of our online system SoCRATe. We are currently investigating an in-DBMS implementation of the different components of SoCRATe. Being able to create and maintain dynamic user profiles in a DBMS has many appealing features, including the ability to query evolving user profiles. This will require to incorporate learning and optimizing user recommendations as a foundational extension of the relational algebra. Queries over profiles will be handled within the relational query runtime without exfiltrating data into memory. This will also open new directions in DB/ML cross-optimizations, such as pushing predicates and model inlining. Existing frameworks such as Raven and its runtime environment ONNX [20] appear as good starting points.

Our third endeavor is to complete the implementation of our system and design an evaluation framework. To evaluate recommendations in SoCRATe, we must revisit traditional accuracy measures and combine them with user-aware measures. A particular focus must be put on capturing users' satisfaction by measuring precision and recall as well as user retention. Precision will be revisited to measure the number of items the user appreciated over all adopted items. Appreciation may correspond to a high rating in the case of movie recommendation or a good quality contribution (wrt a ground-truth) in the case of task recommendation. Recall will be measured as a function of the number of adopted items over all recommended items. Retention will measure the percentage of users who leave after a certain number of iterations. Other measures such as item throughput will capture the speed at which items are consumed wrt their availabilities. These measures can be computed for individual users or for user groups; not all measures will be possible, depending on whether we work with simulated or real users. We are indeed considering a novel setting where semi-synthetic data will be generated using as input real users' data and expanding it to simulate different users and choice models. This will allow us to develop a preliminary benchmark for adaptive recommendations.

References

[1] F. Ricci, L. Rokach, B. Shapira, Recommender systems: introduction and challenges, in: Recommender systems handbook, Springer, 2015, pp. 1–34.

[2] Y. Koren, Factorization meets the neighborhood: a multifaceted collaborative filtering model, in: ACM SIGKDD, ACM, 2008, pp. 426–434.

[3] J. B. Schafer, D. Frankowski, J. Herlocker, S. Sen, Collaborative filtering recommender systems, in: The adaptive web, Springer, 2007, pp. 291–324.

[4] M. J. Pazzani, D. Billsus, Content-based recommendation systems, in: The adaptive web, Springer, 2007, pp. 325–341.

[5] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, et al., Analysis of recommendation algorithms for e-commerce, in: EC, 2000, pp. 158–167.

[6] C. Kim, J. Kim, A recommendation algorithm using multi-level association rules, in: Proceedings of IEEE/WIC, IEEE, 2003, pp. 524–527.

[7] Y. Hu, Y. Koren, C. Volinsky, Collaborative filtering for implicit feedback datasets, in: 2008 Eighth IEEE International Conference on Data Mining, Ieee, 2008, pp. 263–272.

[8] S. Rendle, C. Freudenthaler, Z. Gantner, L. Schmidt-Thieme, Bpr: Bayesian personalized ranking from implicit feedback, in: Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence, AUAI Press, 2009, pp. 452–461.

[9] S. Yao, Y. Halpern, N. Thain, X. Wang, K. Lee, F. Prost, E. H. Chi, J. Chen, A. Beutel, Measuring recommender system effects with simulated users, CoRR abs/2101.04526 (2021).

[10] A. J. B. Chaney, Recommendation system simulations: A discussion of two key challenges, CoRR abs/2109.02475 (2021). URL: https://arxiv.org/abs/2109.02475. arXiv:2109.02475.

[11] N. Hazrati, F. Ricci, Recommender systems effect on the evolution of users' choices distribution, Information Processing & Management 59 (2022) 102766.

[12] S. Wang, L. Hu, Y. Wang, L. Cao, Q. Z. Sheng, M. A. Orgun, Sequential recommender systems: Challenges, progress and prospects, CoRR abs/2001.04830 (2020).

[13] M. Quadrana, D. Jannach, P. Cremonesi, Tutorial: Sequence-aware recommender systems, in: S. Amer-Yahia, M. Mahdian, A. Goel, G. Houben, K. Lerman, J. J. McAuley, R. Baeza-Yates, L. Zia (Eds.), WWW, ACM, 2019, p. 1316.

[14] H. Fang, D. Zhang, Y. Shu, G. Guo, Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations, ACM Trans. Inf. Syst. 39 (2020) 10:1–10:42.

[15] S. Wang, L. Cao, Y. Wang, Q. Z. Sheng, M. A. Orgun, D. Lian, A survey on session-based recommender systems, ACM Comput. Surv. 54 (2022) 154:1–154:38.

[16] J. Pilourdault, S. Amer-Yahia, S. B. Roy, D. Lee, Task relevance and diversity as worker motivation in crowdsourcing, in: IEEE ICDE, 2018, pp. 365–376.

[17] C. Chekuri, S. Khanna, A polynomial time approximation scheme for the multiple knapsack problem, Society for Industrial and Applied Mathematics Journal on Computing 38 (2006).

[18] G. Frisch, J. Léger, Y. Grandvalet, Co-clustering for fair recommendation, in: International Workshops of ECML PKDD 2021, volume 1524, Springer, 2021, pp. 607–630.

[19] M. D. E. et al., All the cool kids, how do they fit in?: Popularity and demographic biases in recommender evaluation and effectiveness, in: FAT, volume 81, PMLR, 2018, pp. 172–186.

[20] K. Karanasos, et al., Extending relational query processing with ml inference, arXiv preprint arXiv:1911.00231 (2019).