Pick & Merge: An Efficient Item Filtering Scheme for Windows Store Recommendations

Adi Makmal
Microsoft R&D, Israel

Jonathan Ephrath
Microsoft R&D, Israel

Hilik Berezin
Microsoft R&D, Israel

Liron Allerhand
Microsoft R&D, Israel

Nir Nice
Microsoft R&D, Israel

Noam Koenigstein
Microsoft R&D, Israel
Tel-Aviv University, Israel

ABSTRACT

Microsoft Windows is the most popular operating system (OS) for personal computers (PCs). With hundreds of millions of users, its app marketplace, Windows Store, is one of the largest in the world. As such, special considerations are required in order to improve online computational efficiency and response times.

This paper presents the results of an extensive research of effective filtering method for semi-personalized recommendations. The filtering problem, defined here for the first time, addresses an aspect that was so far largely overlooked by the recommender systems literature, namely effective and efficient method for removing items from semi-personalized recommendation lists.

Semi-personalized recommendation lists serve a common list to a group of people based on their shared interest or background. Unlike fully personalized lists, these lists are cacheable and constitute the majority of recommendation lists in many online stores.

This motivates the following question: can we remove (most of) the users' undesired items without collapsing onto fully personalized recommendations?

Our solution is based on dividing the users into few subgroups, such that each subgroup receives a different variant of the original recommendation list. This approach adheres to the principles of semi-personalization and hence preserves simplicity and cacheability. We formalize the problem of finding optimal subgroups that minimize the total number of filtering errors, and show that it is combinatorially formidable. Consequently, a greedy algorithm is proposed that filters out most of the undesired items, while bounding the maximal number of errors for each user. Finally, a detailed evaluation of the proposed algorithm is presented using both proprietary and public datasets.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

E-commerce, Recommender Systems, Personalization Systems.

RecSys '19, September 16–20, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3347005

ACM Reference Format:
Adi Makmal, Jonathan Ephrath, Hilik Berezin, Liron Allerhand, Nir Nice, and Noam Koenigstein. 2019. Pick & Merge: An Efficient Item Filtering Scheme for Windows Store Recommendations. In *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3298689.3347005

1 INTRODUCTION

Modern recommender systems such as Netflix, Amazon, and Windows Store, offer a diverse set of different recommendation lists. Some of these lists are fully personalized, while some are non-personalized or semi-personalized. Fully personalized lists are unique lists prepared per user, e.g., by ranking items to each user based on collaborative filtering [5, 8]. In contrast, non-personalized lists are computed once for all users. For example, the lists of popular items or an editor's recommendation list. Semi-personalized lists are computed for subsets of users sharing common interests. For example, in music recommendations genre-based lists can be served to all users who showed interest in a specific genre. Another example is showing users popular items in their geographical location, e.g., "popular in your country". In addition, many types of clustering algorithms can be used for generating semi-personalized lists [7].

Most literature on recommender systems deals with fully personalized recommendation lists [2] as they are more algorithmically challenging. However, in many online stores, such as the Windows Store, semi-personalized lists account for the majority of recommendations presented to users. The popularity of the semi-personalized lists is arguably the result of online experiments that often expose the superior KPIs of these lists (e.g., "Most Popular") when compared to fully personalized lists. Moreover, an essential advantage of such lists is that they are easier to compute, store and deploy.

Most of the recommender systems literature concerns with the problem of finding the best items that *should* be presented to the users. However, there are many cases in which it is worthwhile to detect items that *should not* be presented and remove them from the users' lists. One obvious example is the desire to remove items that the user already purchased. As another example, consider the case of negative correlations between items, e.g. when users who acquired item A tend not to acquire item B: then it may be beneficial to mark item B as an "undesired item" for all the users who acquired item A. In other cases, we may wish to remove potentially inappropriate content for certain groups (e.g. children). Last, as also noted in [6], it may be constructive to filter out recommended items that already had several unsuccessful impressions. While the logic for detecting undesired items greatly varies between different

recommender systems, almost all real-world recommenders employ some sort of filtering mechanism.

This work is not concerned with choosing the specific items to be filtered. Instead, we assume that such filtering is required and propose a method to perform it efficiently.

A naive approach might be to remove the undesired items for each user individually. However, this solution suffers from a major drawback, as it leads to a fully personalized system rather than a group-based one, thereby losing the simplicity and cacheability advantages of the latter. To be more concrete, filtering undesired items for each user can be done either offline, by generating all users' lists in advance, or online, by generating each user's list on-demand. An offline calculation would require storing a huge amount of lists, of the order of the number of users, which prohibits caching and thereby tremendously impacting response times. On the other hand, online calculation would require both reading the user's set of undesired items (non-cached) as well as CPU time for filtering the list on-demand. In both cases, the I/O process of reading a list per user may be prohibitive, particularly in large scale systems such as Windows Store which serves $O(10^8)$ users and handles $O(10^4)$ requests per second.

This observation motivated our search for efficient filtering methods, the results of which we present in this paper.

We focus on a semi-personalized approach that allows to remove undesired items by precomputing a modest number of variants of the original recommendation list with different subsets of items removed from each one. The optimization objective of finding these lists and assigning the users is formalized in Section 2. A key advantage of this scheme is the fact that recommendations remain fully cacheable due to the small number of precomputed lists, and requires no additional CPU or I/O online. This allows us to have the cake and eat it too: the algorithm removes most of the undesired items while maintaining a semi-personalized approach. Furthermore, a recommender system can use a semi-personalized algorithm (for example, clustering [7]) and on top of it to use our approach to remove undesired items which will yield even better KPIs. In the remaining part of the paper we present and evaluate the Pick & Merge algorithm which accomplishes this task. Employing the Pick & Merge algorithm in Windows Store demonstrated a dramatic reduction in both computation and space complexities.

2 OUR APPROACH

Our solution is based on dividing the users into subgroups, according to the users' set of undesired items. Then, a set of items is removed from the recommendations of all the users in each subgroup. This semi-personalized approach implies the existence of two types of errors: (1) *false negative items* - desired items that are erroneously removed from the list; (2) *false positive items* - failing to remove an undesired item. The challenge is to minimize the total number of these mismatches, while bounding them for *each user* individually.

2.1 Formalizing The Filtering Problem

For a list of size k, there are 2^k subsets of undesired items (each item is either desired or not). Each subset can be represented by a bit-string of length k, where 0 and 1 indicate desired and undesired item respectively. The set of all possible bit-strings is denoted by $F_{\text{all}} = \{0, \ldots, 2^k - 1\}$ and each user is associated with the unique bit-string that encodes the user's set of undesired items. We denote by a_i the number of users that are associated with the bit-string $i \in F_{\text{all}}$ and mark the number of false-negative and false-positive errors for user u by $fn(u)$ and $fp(u)$, respectively.

Next,we turn to formalize the filtering problem at hand: Given an ordered recommendation list of size k, a set of undesired items for each user, a maximal number of subgroups (M), and the maximal allowed number of false-negative (n) and false-positive (p) errors, the filtering problem seeks to find:

(1) A set of M subgroups $F_M = \{f_0, \ldots, f_{M-1}\} \subseteq F_{\text{all}}$.
(2) A $map : F_{\text{all}} \rightarrow F_M$,

such that: (a) There are at most n false negative items for each user: $fn(u) \leq n, \forall u$; (b) There are at most p false positive items for each user: $fp(u) \leq p, \forall u$; and (c) It minimizes $v(u) = fn(u) + fp(u)$, the number of item violations for user u, over all users, i.e. it minimizes:

$$\sum_u v(u) = \sum_{i \in F_{\text{all}}} a_i D(i, map(i)), \tag{1}$$

where $D(i, j)$ is the Hamming distance between bit-strings i and j. Note that the choice of the input parameters depends, largely, on the specific domain and the particular business needs. For example, the choice of the p parameter can be dictated by an answer to the following question: "what is the maximal number of undesired items in the individual user's list, beyond which the user's trust is expected to deteriorate?".

To appreciate the combinatorial challenge of the filtering problem, consider the case of a recommendation list of size $k = 10$ and $M = 16$ allowed subgroups, where there are $\binom{2^k}{M} = \binom{1024}{16} > 10^{34}$ options to choose the M subgroups. This shows that even for relatively short lists, the task of finding a small number of optimal subgroups is formidable and can not be done in an exhaustive manner.

We thus adopted an approximated approach, based on a greedy algorithm, which we present next.

2.2 The Pick & Merge Algorithm (P&M)

The proposed algorithm, denoted Pick & Merge (P&M), is inspired by the notion of a "walk on a hypercube", see e.g. [3]. A hypercube of dimension k is an undirected graph of 2^k nodes (see Fig. 1 for the case $k = 3$). Each node can be associated with a bit-string of size k, and the graph is structured such that two nodes are connected by an edge if and only if their bit-strings differ by exactly one bit. In the context of the filtering problem each node represents a possible users' subgroup and merging a node with a "right" ("left") neighboring node increases the number of false negative (positive) items by 1. The bounds on the number of false negative and positive items then have a simple visualization: each node is allowed to "walk" at most p edges to the "left" and at most n edges to the "right" on the hypercube.

The P&M algorithm iteratively picks a subgroup i and merges it onto another subgroup j, such that subgroup i is removed from the set of possible subgroups, and the size of subgroup j increases by a_i, the number of users in subgroup i. In order to minimize the loss in Eq. 1, subgroup i is chosen to be the smallest one, i.e., with the smallest number of users a_i, whereas subgroup j is chosen to be its closest and largest neighbor (a point on the hyper-cube). In addition,

to fulfill the conditions on maximal false negative and false positive items, a single subgroup i is allowed to be merged onto a subgroup $j \neq i$, and we write "$i \rightarrow j$ is allowed", if and only if: (1) bit-string j has at most n 1's that are not in i; and (2) bit-string i has at most p 1's that are not in j. We denote by $H(i)$ the set of subgroups that were so far merged onto subgroup i and note that when merging the subgroup i onto j, it is further required that for all subgroups $h \in H(i)$ it holds that $h \rightarrow j$ is allowed. This iterative procedure, whose pseudo-code is given in Alg. 1, continues until there are only M subgroups left or until no further merging is allowed, in which case a set of more than M subgroups is returned. This constitutes the first phase of the algorithm.

Algorithm 1: Pick & Merge (phase I)

Input: M, k, n, p
Output: A set $F_{M'} = \{f_0, f_1, \ldots, f_{M'-1}\}$ of M' bit-strings
- Initialize: $F_0 = \{0, \ldots, 2^k - 1\}$
- Initialize: $F_1 = \emptyset$
- Initialize: $H(i) = i, \ \forall i \in F_0$
while $(|F_0 + F_1| > M)$ and $(F_0 \neq \emptyset)$ **do**
  - Find bit-string $i = argmin_f \ a_f$, s.t. $f \in F_0$
  - Find $J \subseteq F_0$, s.t. $\forall h \in H(i), \forall \ j \in J, h \rightarrow j$ allowed
  if $J == \emptyset$ **then**
    $F_0 = F_0 \setminus i$
    $F_1 = F_1 \cup i$
  else
    - $d = min_j D(i, j)$, s.t. $j \in J$
    - Find bit-string $j = argmax_f \ a_f$, s.t. $f \in J$, $D(i, f) = d$
    - Merge $i \rightarrow j$:
      $a_j = a_j + a_i$
      $F_0 = F_0 \setminus i$
      $H(j) = H(j) \cup H(i)$
  end if
end while
Return $F_{M'} = F_0 \cup F_1$

Once the set $F_{M'}$ of all the subgroups are fixed, the algorithm enters its second phase, where it maps all the users onto these subgroups. In particular, it performs the mapping:

$$map(i) = argmin_j D(i, j), \ \ i \in F_{\text{all}}, j \in F_{M'}, i \rightarrow j \text{ allowed}, \qquad (2)$$

which minimizes the Hamming distance between i and j, for all allowed subgroups j.

2.3 Toy Example

To facilitate a better understanding of the P&M algorithm, Fig. 1 shows a particular example for a fictitious user distribution over a hypercube of dimension $k = 3$. It indicates that 400 users have no undesired items and are thus associated with the subgroup 000, whereas for 85 users the second item in the list is undesired (010), etc. In this particular distribution a total of 1065 undesired items are presented to the users[1], following the loss calculation of Eq. 1. For the case of $M = 2, n = 0$, and $p = 2$, the algorithm returns the subgroups 000 and 010, marked with full circles. In the first iteration, subgroup 011, which is the smallest subgroup, with only 50 users, is merged onto its closest and largest left neighbor, the 010 subgroup. The iterative process ends after six iterations, with a total loss of 725[2].

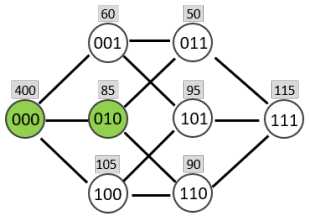

Figure 1: (color online) A fictitious user distribution spread over a hypercube of dimension 3. Numbers in boxes represent the number of users associated with the corresponding bit-strings. For the parameters: $M = 2, k = 3, n = 0$, and $p = 2$ the P&M algorithm returns the subgroups 000 and 010, marked by filled, green, circles.

3 EVALUATION AND DISCUSSION

We evaluate the performance of the P&M algorithm on both public and proprietary datasets. First, we consider the MovieLens [4] and the Netflix [1] datasets and then present the practical impact of the algorithm on propriety data from Windows Store. The MovieLens [4] and Netflix [1] datasets specify dated users' ratings for movies. The MovieLens dataset is composed of 138K users, 27K items, and 20M rating; and the Netflix dataset accounts for more than 480K users, 17K items, and 100M ratings. Items are ranked based on their total sum of ratings.

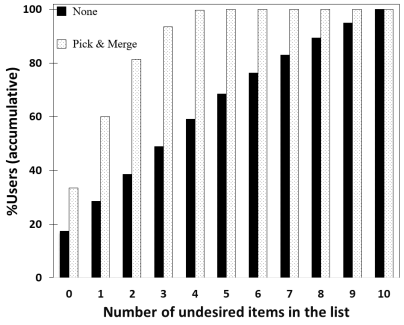

Figure 2: Accumulative percentage of Netflix users with x or less undesired items.

The P&M algorithm requires several input parameters (M, k, n, p). As mentioned above, the choice of these input parameters greatly depends on the specific business requirements. We have tested many different parameter configuration, but encountered no qualitative differences in the algorithm behavior, other than those expected, such as: *increasing the value of sub-groups M reduces the number of undesired items in the users' list.* Thus, to avoid redundancy and due to scope limitations, we stick throughout this section to a single choice of parameters, namely: $k = 10, n = 0, p = 5$. In addition, M was set to the minimal value found by the algorithm: $M = 28$ for the MovieLens dataset and $M = 26$ for the Netflix dataset.

[1] 400*0+(60+85+105)*1+(50+95+90)*2+115*3=1065.

[2] 400*0+60*1+85*0+105*1+50*1+95*2+90*1+115*2=725.

We begin by comparing the original unfiltered lists to the new filtered lists, generated by the P&M algorithm, in terms of the number of undesired items presented to the users. Fig. 2 shows the accumulative percentage of users that are presented with x or less undesired items, before and after applying the algorithm on the Netflix dataset. P&M increases the percentage of users with no undesired items by ~1.5 times. Furthermore, up to $x = 4$, the number of users with as few as x undesired items is almost twice compared to the original list. Finally, while in the original list more than 35% of the users have more than five undesired items, P&M diminishes this number to zero (since $p = 5$ was used). This example showcases the effect of the P&M algorithm on the final recommendation lists. In particular, it shows that the algorithm succeeds in removing a large portion of undesired items from the users' lists while using only a small number of subgroups. Moreover, it ensures a good-enough users' experience for *all users*, including more that 35% of users that were otherwise neglected. The same analysis on the MovieLens dataset yielded very similar trends (not shown).

Next, we compare P&M with two base-line filtering methods: the *RandGroups* approach simply chooses M subgroups at random, whereas the *MaxGroups* method chooses the M most populated subgroups. Clearly, both methods do not adhere to the constraints on the maximal number of false negative and positive items. To facilitate yet a meaningful comparison, we enforce zero number of false negative items ($n = 0$)[3]. The second phase of the algorithm, i.e. mapping the users to the M subgroups, remains otherwise the same as in the P&M algorithm.

Tables 1 and 2 summarize the performance of each method for MovieLens and Netflix respectively, specifying the following: (a) The mean-absolute-error (MAE), i.e. the mean number of undesired items users receive in their list; (b) The mean-squared-error (MSE), the average of the squared number of undesired items. MSE gives a higher weight to users with many undesired items; (c) The percentage of users with more than two undesired items; and (d) The percentage of users with more than $p = 5$ undesired items.

Method	MAE	MSE	%users with $fp(u)>2$	%users with $fp(u)>p$
None	4.2	27.3	65%	36%
RandGroups	2.0	5.7	37%	0.4%
MaxGroups	1.58	5.7	31%	2.4%
P&M	1.62	4.1	28%	0

Table 1: Comparing filtering methods - MovieLens.

Method	MAE	MSE	%users with $fp(u)>2$	%users with $fp(u)>p$
None	3.9	27.3	61%	31.5%
RandGroups	2.1	7.7	38%	4.7%
MaxGroups	1.5	5.3	26%	3.3%
P&M	1.3	3.3	19%	0

Table 2: Comparing filtering methods - Netflix.

Both tables indicate that all the filtering approaches improve the quality of the recommendations by decreasing the number of undesired items. However, P&M provides the largest improvement under most of the evaluated criteria. The MaxGroups method performs the best out of the baselines and in the case of the MovieLens dataset it even achieved the best MAE (marginally better than P&M).

[3]This is attainable via using the 0 . . . 0 subgroup by default.

Naturally, this comes at the cost of violating the condition on the number of false positive items. Note, that P&M leaves no users with more than $p = 5$ unfiltered items.

The practical impact of applying P&M in Windows Store is shown next. We consider the "most popular" items list of size $k = 10$ and our test-set includes a random sample of 42K users that acquired an item in the store during a single day in March 2018. On average, each user had about 1.5 previously purchased items that needed filtering. To fill up the void left by the filtered items, lower ranked items were pushed towards the top. Our evaluation focuses on the canonical recall metric that indicates the average conditional probability that an item resides in the recommendation list, given that the user later acquires it.

Figure 3 shows the recall in Windows Store as a function of the number of recommendations when no filtering is applied (solid, squares, black), when fully personalized filtering is applied (dotted, stars, blue), and when P&M is applied (dashed, circles, red)[4]. As expected, fully-personalized filtering does increase the recall. For short list's size, P&M follows the full personalization curve almost perfectly. It deviates from it as the list's size increases. Here, the P&M algorithm found a minimal number of $M = 13$ subgroups. Clearly, employing a larger number of M subgroups would decrease the gap between P&M and fully-personalized filtering even further. In fact, applying $M = 100$ leads to a recall curve that is hardly distinguishable from the fully personalized one (not shown). This analysis shows that P&M, despite providing merely semi-personalized filtering, succeeds in restoring most of the fully-personalized filtering, and that such a filtering has a great potential to improve the performance of the recommendation lists.

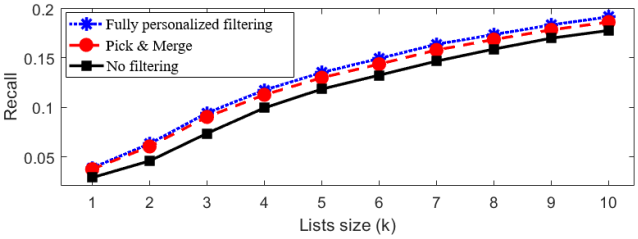

Figure 3: (color online) Windows Store dataset; Recall as a function of list's size.

4 SUMMARY

In this paper, we turned a spotlight towards a concept that so far received very little attention in the literature, namely the importance of marking items as undesired and filtering them out of recommendation lists efficiently. In particular, we motivated a semi-personalized approach for item filtering which enables cacheability. We formalized the problem and presented the Pick & Merge algorithm which removes a large portion of undesired items while bounding the number of false negative and false positive errors per user. Employing the Pick & Merge algorithm in the Windows Store marketplace demonstrated a dramatic reduction in both computation and space complexities. We believe that the algorithm and insights presented in this paper could be beneficial to other repartitioners working on large-scale recommender systems.

[4]For confidentiality, all curves were scaled by a constant.

REFERENCES

[1] J. Bennett and S. Lanning. The Netflix Prize. In *Proc. KDD Cup and Workshop*, 2007.

[2] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109 – 132, 2013.

[3] I. A. Campbell, J. M. Flesselles, R. Jullien, and R. Botet. Random walks on a hypercube and spin glass relaxation. *Journal of Physics C: Solid State Physics*, 20(4):L47, 1987.

[4] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, Dec. 2015.

[5] Y. Koren and R. M. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 77–118. 2015.

[6] O. Sar Shalom, N. Koenigstein, U. Paquet, and H. P. Vanchinathan. Beyond collaborative filtering: The list recommendation problem. In *Proceedings of the 25th International Conference on World Wide Web*, WWW'16, pages 63–72, 2016.

[7] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the fifth international conference on computer and information technology*, volume 1, pages 291–324, 2002.

[8] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, Jan. 2009.