On Competitive Recommendations

Jara Uitto[1] and Roger Wattenhofer[2]

[1] ETH Zurich, Switzerland juitto@tik.ee.ethz.ch
[2] Microsoft Research rogerw@microsoft.com

Abstract. We are given an unknown binary matrix, where the entries correspond to preferences of users on items. We want to find at least one 1-entry in each row with a minimum number of queries. The number of queries needed heavily depends on the input matrix and a straightforward competitive analysis yields bad results for any online algorithm. Therefore, we analyze our algorithm against a weaker offline algorithm that is given the number of users and a probability distribution according to which the preferences of the users are chosen. We show that our algorithm has an $\mathcal{O}(\sqrt{n}\log^2 n)$ overhead in comparison to the weaker offline solution. Furthermore, we show that the corresponding overhead for any online algorithm is $\Omega(\sqrt{n})$, which shows that the performance of our algorithm is within an $\mathcal{O}(\log^2 n)$ multiplicative factor from optimal.

Keywords: Learning, Online, Recommendation, Algorithms

1 Introduction

Among the most important keys to success when tackling a machine learning problem are the quality and especially the quantity of training data. After all, the very definition of machine learning is to study a given or a previously observed set of samples to produce useful predictions about identities or properties of unseen samples.

In this paper, we study a purely algorithmic learning process that starts out with zero knowledge. Given an unknown arbitrary binary $n \times m$ matrix, how many times do we have to query (probe) single entries in the input matrix until we find a 1-entry in each row? Clearly the answer to this question depends on the matrix. If all the entries of the matrix are 1, the task is trivial. On the other hand, if there is only one 1-entry in each row at a random position, the task is hard.

The unknown binary matrix can be seen as a preference matrix, which represents the preferences of n users on m items. In particular, a 1-entry at position (i, j) of the matrix indicates that user i likes item j, whereas a 0-entry indicates that user i does not like item j. The goal is to find a suitable item for each user. Instead of abstract users and items, we may think of the items as books and the users as bookstore customers. We allow the bookstore keeper to perform two different operations. One of them is suggesting a book to a customer and asking her to give a (binary) review after reading it. The other is to sell a customer

a book that she liked, that is, the book got a positive review from her. After selling a book to a customer, the customer does not return to the store. The customers that have bought a book are considered satisfied, and the goal is to satisfy all customers with as few queries as possible.

Naturally, satisfying a customer that does not like any books is impossible, and therefore we assume that the preference vector of each user contains at least one 1-entry. For simplicity, we assume that the feedback is instantaneous. We also assume that the customers come to the bookstore in a random fashion, and that the bookstore keeper is allowed to pick books for reviewing at random. The goal is to minimize the effort required from the customers, in particular the number of queries until all customers have been sold a book that they liked. The trivial upper and lower bounds for the cost are $n \cdot m$ and n respectively, as it takes $n \cdot m$ queries to learn the whole input matrix, and at least n queries to present a book to each customer.

In a usual competitive analysis, an offline algorithm that can see the whole input is compared against an online algorithm that has no previous knowledge of the input. We observe that an offline algorithm that sees the input matrix can simply sell each customer a book she likes, resulting in a cost of n for any input. Since any online algorithm would perform badly in comparison to this algorithm, we choose a weaker offline algorithm, referred to as *quasi-offline* algorithm and compare our algorithm against the weaker algorithm. We call the competitive analysis against the quasi-offline algorithm *quasi-competitive*.

Definition 1 (Quasi-Competitiveness). *An online algorithm A is α-quasi-competitive if for all inputs I*

$$c(A(I)) \leq \alpha \cdot c(OPT_q(I)) + \mathcal{O}(1) \ .$$

where OPT_q is the optimal quasi-offline algorithm and $c(\cdot)$ is the cost function of A and OPT_q, respectively.

Since seeing the whole input at once gives the offline algorithm too much power, we weaken the offline algorithm by not providing it with full information of the input. Instead, we give the quasi-offline algorithm a probability distribution D over possible preference vectors and the number of customers n. The preference vectors for these customers are chosen independently at random from D.

We show that from the perspective of the quasi-offline algorithm, solving our problem is equivalent to solving Min Sum Set Cover (**mssc**) problem, where customers correspond to the elements and books to the sets. The input for **mssc** is the same as for the well-known Set Cover problem, but the output is a linear order on the sets. This order induces a cost for each element, where the price is the ordinal of the first set that covers the element. The optimal solution to **mssc** minimizes the expected cost for a randomly chosen element.

If we allow the number of books to be large, then there might be lots of books without any useful information, for example books that no one likes. The quasi-offline algorithm can ignore these books, whereas an online algorithm

cannot. This results in an unavoidable increase in the costs for online algorithms while not affecting the cost of the quasi-offline algorithm. Therefore, we assume throughout the paper that the number of books is not much larger than the number of customers, i.e., $m \in \mathcal{O}(n)$.

The main result of the paper is an $\mathcal{O}(\sqrt{n}\log^2 n)$-quasi-competitive algorithm for our recommendation problem. We also show that the quasi-competitive ratio for any algorithm that does not know the input matrix is $\Omega(\sqrt{n})$. This indicates that our algorithm is within a polylogarithmic factor from the best possible online solution. We note that the definition of quasi-competitiveness extends to other problems by choosing a suitable quasi-offline algorithm for the considered problem. Naturally, the choice of the quasi-offline algorithm should be made carefully to preserve the difficulty of the problem.

2 Related Work

Previous work on **mssc** has concentrated on the case where the sets to which a given element belongs to are shown. In the offline case, Bar-Noy, Bellare, Halldórsson, Shachnai and Tamir showed that a greedy algorithm achieves a 4-approximation to the optimal algorithm [7]. Feige, Lovász and Tetali gave a simpler proof for this result and showed that it is NP-hard to get a $(4-\epsilon)$-approximation for any $\epsilon > 0$ [10]. On the online field, Munagala et al. gave an algorithm that provides an $\mathcal{O}(\log n)$-approximation for the optimal algorithm even if the contents of the sets are unknown [18]. The paper by Munagala et al. in addition to the work by Babu et al. [6] brought the problem closer to practical applications by modeling it as pipelined stream filters. In both papers, they study the problem of assigning different filters to data streams, where the overall processing costs depend on how the filters are ordered.

Learning the input of **mssc** can also be seen as learning binary relations. For example Goldman et al. studied the problem and showed that with arbitrary row types, the learner can be forced to make $\Omega(n^2)$ mistakes when predicting which row is in question [12]. They studied the learning task with four different kinds of online inputs: a helpful teacher, random, an adversary, and a case where the learner can choose which element to look at. Furthermore, Kaplan et al. [16] gave more general bounds for similar learning tasks by abstracting the input of an **mssc** instance into a set of DNF clauses, where an element belonging to a set corresponds to a term being true in the clause that corresponds to the set.

Using the solution for **mssc** for recommendations can also be abstracted into maintaining a ranking while learning in an active manner. Azar and Gamzu provided an $\mathcal{O}(\ln(1/\epsilon))$-approximation algorithm for ranking problems where the cost functions have submodular valuations [5]. Their algorithm iteratively selects sets that have maximal marginal contributions. The properties of problems with submodular cost functions were further studied in an adaptive environment by Golovin and Krause [13]. There also exists a lot of machine learning studies about adaptive and active versions of other classic optimization problems such as Set Cover [11, 17], Knapsack [8] and Traveling Salesman [15].

Our problem is a variant of an online recommendation problem. Learning recommendation systems, where the goal is to learn the whole preference matrix, has been studied by Alon et al. [2]. They showed that with high probability, one can learn the whole matrix with little error in logarithmic time in a distributed setting.

Awerbuch, Patt-Shamir, Peleg and Tuttle gave a centralized algorithm that outputs a set of recommendations that satisfy all customers with high probability [4]. The idea is to select a committee that learns its preference vectors completely. The favorite product of each committee member is then suggested to all remaining customers. They note that in the presence of malicious customers, the committee based approach has disadvantages. Thus, they also present a distributed algorithm for the recommendation problem that does not use a committee, and show that it is resilient to Byzantine behavior. The connection to our work is the model they use with the basic idea of suggesting the most preferred product to the rest of the users. The main contrast to their work is in the complexity measures. They use the similarities of preferences as a basis of the complexity of their algorithms, whereas we compare the cost of our algorithm to the cost of a greedy algorithm. We also show that the worst case performance of our algorithm is good against any online solution.

In addition, Awerbuch et al. studied reputation systems, which are closely related to recommendation systems [3]. They studied a model, where items are either liked by everyone or by no one. The goal is to find for all users an item they like by querying random objects or by asking other users for their preferences. They also considered Byzantine users and restricted access to items. The main similarity to our work is the cost of the recommendation algorithms in the model they use, where querying 0-entries has a unit cost and querying 1-entries is free but the execution goes on indefinitely. The main difference is in the worst case input. They assume that there is always a possibility of cooperation between users, whereas we analyze our solution for an arbitrary feasible input.

To the best of our knowledge, our work is the first to perform a competitive analysis on recommendation algorithms, where the power of the offline algorithm is reduced. In general, the offline algorithms being too powerful is not a new problem. However, the usual approach is to provide the online algorithm additional power. For example, online algorithms with some lookahead into the future have been studied for the list update [1] and bin packing [14] problems. In our case however, the cost of an offline solution is always n regardless of the input and therefore, a competitive analysis does not make sense even if the online algorithm was granted more power. The term competitive in our context was introduced earlier by Drineas et al. [9]. In contrast to us, they measure their competitiveness against the number of rows that the algorithm has to learn to be able to predict the rest.

3 Model

We begin defining our model by giving a formal description of our recommendation problem. We are given a set of customers U and a set of books B, where initially $|U| = n$ and $|B| = m \in \mathcal{O}(n)$. In addition, we are given a probability distribution D over all possible preference vectors, where the preference vectors correspond to binary vectors of length m with at least one 1-entry. Each customer is then assigned a hidden preference vector chosen independently at random from D. A recommendation algorithm works in rounds. At the beginning of each round, the algorithm is given a customer u uniformly at random from the set U. Then the algorithm has to recommend the customer u a book $b \in B$, which is equivalent to checking whether u likes b or not. We assume that this check is instantaneous, i.e., we immediately know whether customer u likes book b.

When the algorithm receives a positive review from a customer u it has the opportunity to label u as *satisfied*, after which u is removed from U. From here on, the set U is referred to as the set of *unsatisfied* customers. The goal of a recommendation algorithm is to satisfy all customers, i.e., the execution terminates when the set of unsatisfied customers U becomes empty.

An important concept throughout the paper is the *popularity* of a book. The popularity of a book corresponds to the number of unsatisfied customers liking it.

Definition 2. *Let b be a book. The popularity $|b|$ of the book b is the number of unsatisfied customers that like this book, i.e.,*

$$|b| = |\{u \in U \mid u \text{ likes } b\}| \ .$$

The execution of a round of a recommendation algorithm is divided into three steps:

1. Receive a customer $u \in U$ chosen uniformly at random.
2. Recommend a book b to the customer u.
3. If u likes b, choose whether or not to remove u from U.

The algorithm is allowed to make computations during all the steps, and all computations are considered free. The cost of a recommendation algorithm is the number of queries (rounds) the recommendation algorithm has to perform in expectation until all customers are satisfied.

We note that as the input can be interpreted as a $n \times m$ binary matrix, customer u can be identified with the index of the corresponding row and book b with the index of the corresponding column in the matrix. From here on $u \in U$ indicates both the element and the index and similarly for $b \in B$.

4 The Quasi-offline Algorithm

As we mentioned before, it is possible to build an example where it will take $\Omega(n \cdot m)$ queries in expectation to satisfy all customers for any online algorithm

(diagonal matrix for example). We tackle this issue by performing a quasi-competitive analysis on our algorithms, i.e., we compare our algorithms to a quasi-offline algorithm that is provided with the probability distribution D from where the preference vectors of the n customers were chosen from. Since any preference vector v of customer u in the input is picked at random and independently from previous picks, gaining information from other customers than u does not help the quasi-offline algorithm to identify u. Therefore, the quasi-offline algorithm does not gain anything from using different recommendation strategies on different customers or from recommending more books to u after finding a book u likes.

The recommendation strategy for any customer u is an ordered list of books that are successively recommended to u. Furthermore, the optimal strategy is a strategy that minimizes the total cost, i.e., the expected sum of recommendations made to all users. Let N be the smallest integer such that $P[X = v] \cdot N$ is an integer for every preference vector v, where X is a random variable that obeys the probability distribution D. Then our problem, from the perspective of the quasi-offline algorithm, is equivalent to solving an **mssc** instance that corresponds to the following input for our recommendation problem. For each preference vector v, there are exactly $P[X = v] \cdot N$ customers that have the corresponding preferences. And again, by considering the books as sets and the customers as elements, we get an **mssc** instance with N elements. A solution to **mssc** gives a straightforward solution to our problem, where each customer is recommended books according to the (same) ordering. Since the optimal algorithm for **mssc** also minimizes the total cost, our quasi-offline algorithm corresponds to the optimal algorithm for **mssc**.

It has been shown that a greedy algorithm, that successively selects sets that cover as many uncovered elements as possible, provides a 4-approximation to the optimal solution [7] for **mssc**. Therefore, we gain only an additional constant factor overhead by comparing our solution to the greedy one instead of the optimal. We refer to the greedy quasi-offline algorithm for **mssc** as the quasi-offline algorithm. We denote the ordered set of books chosen by the quasi-offline algorithm by $\mathcal{C} = b_1, b_2, \ldots, b_k$. Let S_i be the set of users satisfied by the quasi-offline algorithm with book b_i, i.e.,

$$S_i = \{u \in U | u \text{ likes } b_i \wedge u \text{ does not like } b_j \text{ for any } j < i\} \ .$$

We refer to the size of S_i as the *disjoint popularity* of b_i. The average time $\mathcal{E}$ taken on each customer by the quasi-offline algorithm is given by

$$\mathcal{E} = \frac{1}{n} \sum_{i=1}^{k} i \cdot |S_i| \ .$$

We note that $\mathcal{E}$ is also the expected time for the quasi-offline algorithm to satisfy a randomly picked customer.

We use the rest of the section to present general bounds on the introduced concepts, which we will later use in the analysis of our recommendation algo-

rithm. First we give an upper bound for the number of books of certain disjoint popularity in $\mathcal{C}$.

Lemma 1. *Let $r \in \mathbb{R}$. The maximum number of books with disjoint popularity of at least n^r in $\mathcal{C}$ is at most $n^{(1-r)/2}\sqrt{2\mathcal{E}}$.*

Proof. Let ℓ be the number of books with disjoint popularity n^r or larger. Now $\ell \leq k$ and then

$$\mathcal{E} \geq \frac{1}{n}\sum_{i=1}^{\ell} n^r \cdot i = \frac{1}{n} n^r \sum_{i=1}^{\ell} i = n^{r-1} \cdot \frac{\ell^2 + \ell}{2} \ ,$$

and thus $\ell \leq \sqrt{\ell^2 + \ell} \leq \sqrt{2\mathcal{E}} \cdot n^{1/2-r/2}$.

Then we give an upper bound for the size of U when given the popularity of the most popular book. The most popular book b^* in round i is the book with maximum popularity, i.e., $|b^*| \geq |b|$ for all $b \in B$. Note that since the popularities of the books can be reduced during the execution, another book might be the most popular in round $i + 1$.

Lemma 2. *Let n^r be the popularity of the most popular book for some $r \in \mathbb{R}$. Then the size of U is smaller than $3\sqrt{\mathcal{E}} \cdot n^{1/2+r/2}$.*

Proof. To count the total number of unsatisfied customers, we count the unsatisfied customers liking the books in $\mathcal{C}$. As the popularity of the most popular book is at most n^r, we know that there are at most n^r unsatisfied customers that like any single book in $\mathcal{C}$. By Lemma 1, initially there are at most $\sqrt{2\mathcal{E}}n^{1/2-r/2}$ books of disjoint popularity n^r or greater in $\mathcal{C}$. Therefore, the total number of unsatisfied customers liking these books is at most $\sqrt{2\mathcal{E}}n^{1/2+r/2}$.

To bound the number of users liking the rest of the books, we define a random variable X that denotes the number of books we have to suggest to a randomly chosen customer. We observe that $E[X] = \mathcal{E}$ and by Markov's inequality

$$p = \mathbb{P}(X > \sqrt{2\mathcal{E}}n^{1/2-r/2}) \leq \frac{\mathcal{E}}{\sqrt{2\mathcal{E}}n^{1/2-r/2}} \leq \sqrt{\mathcal{E}} \cdot n^{r/2-1/2} \ .$$

Therefore, we have $pn \leq \sqrt{\mathcal{E}} \cdot n^{r/2+1/2}$ customers that are not satisfied with the books of popularity n^r or larger. Finally, the total number of unsatisfied customers is at most

$$|U| \leq \sqrt{2\mathcal{E}} \cdot n^{1/2+r/2} + \sqrt{2\mathcal{E}} \cdot n^{1/2+r/2} < 3\sqrt{\mathcal{E}} \cdot n^{1/2+r/2} \ .$$

□

5 Online Algorithms

In this section, we introduce two online algorithms for the recommendation problem. First, we present an algorithm that achieves an optimal quasi-competitive ratio when restricting ourselves to a case where every customer likes exactly one book.

5.1 Customers Like Exactly One Book

Let us assume that each customer likes exactly one book. We observe that the probability of a randomly picked customer liking a random book is at least $1/m$. Therefore, by suggesting random books to random users, we get a positive feedback after $\mathcal{O}(m)$ queries in expectation regardless of the number of unsatisfied users.

Our algorithm for the easier environment with one book per customer is the following. Initially, we start with an empty set of *good* books G. After a positive feedback on book b, we add it to G. Each customer is recommended all the books in the set G (once) before they are recommended more random books. The cost of the algorithm with respect to $\mathcal{E}$ is summarized in the following theorem.

Theorem 1. *There exists a recommendation algorithm that satisfies all customers with $\mathcal{O}(n\sqrt{n\mathcal{E}})$ queries in expectation, when each customer likes exactly one book.*

Proof. By Lemma 1 the number of books of disjoint popularity of at least one is $\mathcal{O}(\sqrt{n\mathcal{E}})$. Since each customer likes exactly one book, the popularity of a book b is equal to the disjoint popularity of b. Therefore, the maximum number of books that need to be added to G is in $\mathcal{O}(\sqrt{n\mathcal{E}})$. Furthermore, attempting to satisfy all the future customers with the books from G takes $\mathcal{O}(n\sqrt{n\mathcal{E}})$ queries in total. As the expected number of queries to find a new book by randomly sampling customers and books is $\mathcal{O}(m)$, it takes $\mathcal{O}(m\sqrt{n\mathcal{E}})$ queries with random books to discover the books that satisfy all customers. Therefore, the cost of the algorithm to satisfy all customers is

$$\mathcal{O}(n\sqrt{n\mathcal{E}}) + \mathcal{O}(m\sqrt{n\mathcal{E}}) \in \mathcal{O}(n\sqrt{n\mathcal{E}}) \ .$$

□

A matching lower bound can be found by constructing a simple example, where there are lots of customers that like books of popularity one. These customers have to be satisfied by searching the preferred book in a brute force fashion.

Theorem 2. *Any recommendation algorithm without any initial knowledge of the input needs $\Omega(n\sqrt{n\mathcal{E}})$ queries in expectation to satisfy all customers.*

Proof. Let $H = (U, B)$ be an input with $|U| = n$ customers and $|B| = \Theta(n)$ books. In addition, let $1 \leq F \leq n$. The preferences on the books are distributed in the following manner: one book is liked by $n - \sqrt{nF}$ customers and the rest of the customers like books of popularity 1. The preferences of the customers in input H are illustrated in Figure 1. The average cost for the quasi-offline algorithm to satisfy a single customer is

$$\mathcal{E} = \frac{n - \sqrt{nF}}{n} + \frac{1}{n} \sum_{i=2}^{\sqrt{nF}+1} i \ ,$$

when $n \to \infty$. Thus, $\mathcal{E} \in \mathcal{O}(F)$.

Any algorithm that does not have any initial knowledge about the input has to learn all the books liked by single customers. As there are no mutual information within the books, the best way for any algorithm is to suggest randomly picked books to the customers until all of them give a positive feedback and label them as satisfied immediately. This takes $\Omega(|B|)$ queries in expectation for each customer not liking the popular book and therefore satisfying all $\sqrt{nF}$ of the customers with a single preference takes $\Omega(|B|\sqrt{nF}) \in \Omega(n\sqrt{n\mathcal{E}})$ queries in expectation. □

$$
\begin{array}{r}
n-\sqrt{nF} \\
\sqrt{nF}
\end{array}
\left\{\begin{array}{|cccccccccc}
1 & 0 & 0 & & & & & & & \\
1 & 0 & 0 & & & & & & & \\
1 & 0 & 0 & & & & & & & \\
1 & 0 & 0 & & & & & & & \\
1 & 0 & 0 & \ldots & & & & & & \\
1 & 0 & 0 & & & & & & & \\
1 & 0 & 0 & & & & & & & \\
 & \vdots & & & & & & & & \\
1 & 0 & 0 & 0 & 0 & & & & & \\
0 & 1 & 0 & 0 & 0 & & & & & \\
0 & 0 & 1 & 0 & 0 & & & & & \\
0 & 0 & 0 & 1 & 0 & & & & & \\
 & & & & & \ddots & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right.
$$

Fig. 1. An input matrix that is difficult for any algorithm that is initially oblivious to the input. The first $n - \sqrt{nF}$ rows have a single 1-entry in the first column. Let $r_1, \ldots, r_{\sqrt{nF}}$ denote the remaining rows. Then row r_i contains a single 1-entry at position $i + 1$. For any online algorithm, it will take $\Omega(\sqrt{nF})$ queries in expectation to find a 1-entry from rows $r_1, \ldots, r_{\sqrt{nF}}$ since they have no mutual information with any other rows.

5.2 Customers with Multiple Preferences

Now we lift the assumption that each customer likes exactly one book. The basic idea of our algorithm for the more general version of the problem is similar to the one preference case. However, we might now have a lot of less popular books than the greedy choice, where customers liking the less popular books are subsets of the ones liking the greedy choice. By suggesting the first book that is liked by any customer to all the following ones, we are likely to satisfy the customers with the less popular books which results in lots of additional queries on average.

Therefore, we search for books that are almost as popular as the most popular book within the unsatisfied customers. Specifically, we learn the preferences of the customers until we get at least $c \log n$ positive reviews on a single book for some constant c which decides the error rate of the choice. All the sampling information is stored in a matrix M, where an entry $M(u, b)$ corresponds to the number of positive feedbacks by customer u on book b. The total number of positive feedbacks on a certain book b corresponds to the sum of positive feedbacks on column b.

In addition, we might have lots of books with almost equal popularity that are not liked by the same customers and doing the sampling for all of them successively might be costly. Therefore, after $c \log n$ positive feedback on a single book, we use the gained sampling information to select a set of equally popular books to the set of good books instead of just one. To avoid choosing books with overlap, we re-estimate the popularities after each choice. A pseudo-code representation of the algorithm is given in Algorithm 1. As the sampling and the greedy choices are done successively, we present their pseudo-code as subroutines of Algorithm 1. The pseudo-codes for the sampling part and the greedy part are given in Algorithm 2 and Algorithm 3, respectively. Note that while sampling, users are not removed and thus the same user can give several positive feedbacks on the same book, i.e., even a single user will eventually give $c \log n$ positive feedbacks on some book.

We divide the execution of our algorithm into *phases*. One phase consists of two changes in the state, i.e., sampling until $c \log n$ positive feedbacks are given and the greedy choices have been made. We begin the analysis of the algorithm by showing that each greedy choice made by Algorithm 3 during a single phase is either within a constant factor from the best one or all the reasonable choices are already made. To do this, we categorize the books by their popularities. A book b belongs to category cat_i if $2^{i-1} \leq |b| < 2^i$. We refer to the upper bound of the popularities of the books in cat_i as the *size* of the corresponding category.

Lemma 3. *Let $b \in cat_i$ be the most popular book in the beginning of phase j. Each book chosen greedily by Algorithm 3 during phase j is liked by at least 2^{i-4} unsatisfied customers with high probability.*

Proof. Let X_b be a random variable that denotes the number of positive feedbacks given to book b during phase j. First, we want to show that the expected value $E(X_b)$ is close to the amount of sampling required on one book before the greedy part of phase j begins, i.e., $c \log n$. Let us assume for contradiction that $\mu = E(X_b) > (3/2)c \log n$. The Chernoff bound states that with high probability, the actual value of X_b is within a constant multiplicative factor from its expected value after enough sampling. More precisely the bound states that

$$P(X_b \leq c \log n) = P(X_b < (1 - 1/3)\mu) < \left(\frac{e^{-1/3}}{(2/3)^{2/3}} \right)^{\mu}$$
$$< \left(e^{-1/3} \right)^{c \log n} \in \mathcal{O}\left(n^{-c/3} \right) .$$

Algorithm 1 Phases

Require: A set of customers U and a set of books B.
  Initialize a zero matrix M of size $n \times m$.
  Initialize a state STATE $\leftarrow$ sample.
  Initialize a one vector v of length n.
  Book $b^* \leftarrow$ **null**
  while there are unsatisfied customers **do**
    Receive a random customer $u \in U$.
    if STATE = sample **then**
      Run Sampling(u).
    else
      Run Greedy(u).
    end if
  end while

Algorithm 2 Sampling(u)

  Choose a random book $b \in B$.
  if u likes b **then**
    $M(u, b) \leftarrow M(u, b) + 1$.
  end if
  if $\sum_{i=0}^{n-1} M(i, b) \geq \lfloor c \log n \rfloor$ **then**
    $b^* \leftarrow b$.
    Set $v(u') = 0$ for all $u' \in U$.
    STATE $\leftarrow$ greedy.
  end if

Algorithm 3 Greedy(u)

  if $v(i) = 1$ for all customers i **then**
    Set $v(u') = 0$ for all $u' \in U$.
    Set $b^* \leftarrow b$, where $\sum_{i=0}^{n-1} M(i, b)$ is largest, ties broken arbitrarily.
  end if
  $v(u) \leftarrow 1$.
  if u likes b^* **then**
    Remove u from U.
    for $0 \leq j < m$ **do**
      $M(u, j) \leftarrow 0$.
    end for
  end if
  if $\sum_{i=0}^{n-1} M(i, j) < (c \log n)/4$ for all j **then**
    STATE $\leftarrow$ sample.
    Reset M.
  end if

This indicates that $X_b > c \log n$ with high probability, which is a contradiction since the sampling stops when any book has more than $c \log n$ positive feedbacks. Thus, $E(X_b) \leq (3/2)c \log n$ with high probability.

The next step is to show that the number of positive feedbacks on any book $b' \in \text{cat}_{i-4}$ is smaller than $(c \log n)/4$ with high probability. As the popularity of b' is less than $|b|/8$ by the definition of the categories, we have

$$\mu' = E(X_{b'}) < \frac{E(X_b)}{8} \leq \frac{3c \log n}{16} \ .$$

Again using the Chernoff bound, we get that

$$P(X_{b'} > (c \log n)/4) < P(X_{b'} > (1 + 1/3)\mu') \in \mathcal{O}\left(n^{-3c/32}\right) \ .$$

□

The next step of the analysis is to show that the size of U has decreased by a significant amount after each phase. We do this by showing that after each phase, the most popular book belongs to a smaller category than before running the phase.

Lemma 4. *Let cat_i be the largest category. After running one phase of Algorithm 2, there are no books left in cat_i with high probability.*

Proof. Let b be the most popular book. Similarly as in Lemma 3 we can use the Chernoff bound to show that with enough sampling, $c \log n$ is at most within the factor $3/2$ from the number of elements discovered from b. The bound can also be used to show that with high probability, the number of positive feedbacks X'_b on any book $b' \in \text{cat}_i$ is more than $E(X_{b'}) \cdot (3/4)$, where $E(X_{b'}) > E(X_b)/2$ by the definition of categories.

Therefore, with high probability

$$X'_b > \frac{3E(X'_b)}{4} > \frac{3E(X_b)}{8} > \frac{c \log n}{4} \ .$$

As all the books with at least $(c \log n)/4$ positive feedbacks are chosen greedily, all books from cat_i will eventually be chosen or their popularity will reduce to a lower category during the execution of one phase with high probability. □

5.3 The Cost

After figuring out the number of phases needed to satisfy all customers, it remains to analyze the cost of a single phase. We begin by tackling the sampling part where the dominating factors for the cost are the number of unsatisfied customers and the popularity of the most popular book.

Lemma 5. *During each phase, Algorithm 2 is called $\mathcal{O}(n\sqrt{\mathcal{E}n} \log n)$ times in expectation to get $c \log n$ positive feedbacks on the most popular book by Algorithm 2.*

Proof. Let b be the most popular book. The probability of receiving a random customer that likes book b is

$$\frac{|b|}{|U|} \geq \frac{|b|}{3\sqrt{\mathcal{E}n|b|}} = \frac{\sqrt{|b|}}{3\sqrt{\mathcal{E}n}}$$

by Lemma 2. Furthermore the probability of choosing the correct book randomly is $1/m$. Therefore, the expected amount times book b is recommended to customers that like it after $3m\sqrt{\mathcal{E}n} \cdot c \log n$ queries is more than

$$\frac{m3\sqrt{\mathcal{E}n} \cdot c \log n}{\sqrt{|b|}} \cdot \frac{\sqrt{|b|}}{m3\sqrt{\mathcal{E}n}} = c \log n \ .$$

□

The last item needed for the analysis is an upper bound for the cost of the greedy part of the algorithm.

Lemma 6. *Running one phase of Algorithm 1 costs $\mathcal{O}(n\sqrt{\mathcal{E}n}\log n)$ in expectation.*

Proof. By Lemma 5 the cost of the sampling state of each phase is $\mathcal{O}(n\sqrt{\mathcal{E}n}\log n)$ in expectation after which the greedy state is assumed.

By Lemma 3 all the greedily chosen books are liked by at least 2^{i-4} customers with high probability, where i is the index of the largest category with non-empty set of books. By Lemma 2 there are at most $\mathcal{O}(\sqrt{\mathcal{E}n2^i})$ unsatisfied users left in the beginning of the phase. Therefore, after making $\mathcal{O}(\sqrt{\mathcal{E}n})$ greedy choices either all users have been satisfied or the algorithm has restarted the sampling part of the algorithm. Furthermore, it takes $\mathcal{O}(n \log n)$ rounds in expectation for the greedy part to suggest each book to all customers, therefore the expected number of calls to the greedy subroutine is

$$\mathcal{O}(n \log n) \cdot \mathcal{O}(\sqrt{\mathcal{E}n}) \in \mathcal{O}(n\sqrt{\mathcal{E}n}\log n) \ .$$

□

Theorem 3. *Algorithm 1 is $\mathcal{O}(\sqrt{n}\log^2 n)$-quasi competitive.*

Proof. By Lemma 4 each phase reduces the size of the largest category by at least a factor of 2 and therefore after $\mathcal{O}(\log n)$ phases the size of the largest category reduces to 1. Running one more phase of the algorithm results in the greedy algorithm including all books with high probability terminating the execution. As the cost of each phase of the algorithm is $\mathcal{O}(n\sqrt{\mathcal{E}n}\log n)$ by Lemma 6, in expectation the whole cost is

$$\mathcal{O}(\log n) \cdot \mathcal{O}(n\sqrt{n\mathcal{E}}\log n) \in \mathcal{O}(n\sqrt{n\mathcal{E}}\log^2 n) \ .$$

Since the cost of the quasi-offline algorithm is $n\mathcal{E}$, the quasi competitive ratio of Algorithm 1 is

$$\frac{n\mathcal{E}}{\mathcal{O}(n\sqrt{n\mathcal{E}}\log^2 n)} \in \mathcal{O}\left(\frac{\sqrt{n}\log^2 n}{\sqrt{\mathcal{E}}}\right) \ .$$

Specifically, when $\mathcal{E}$ is a constant, Algorithm 1 is $\mathcal{O}(\sqrt{n}\log^2 n)$-quasi competitive.

□

6 Conclusion

The main result of this paper is a centralized algorithm for a recommendation problem with a binary matrix as an input. As a general input results in the trivial lower bound of $\Omega(n^2)$ for any algorithm, we approached the problem from a competitive perspective. However, an offline algorithm that has access to the whole input matrix always has a cost of exactly n regardless of the input. Therefore, we introduced the concept of quasi-competitiveness, where the online algorithm is compared to an optimal quasi-offline algorithm that has a restricted access to the input matrix. In particular the quasi-offline algorithm is given the number of customers n and a probability distribution from which the preference vectors of the customers are chosen.

We observed that given the restriction, the best that the quasi-offline algorithm can do is to compute a list of books that minimizes the expected number of books the algorithm has to recommend to an unknown customer. Computing the static list is equivalent to solving the **mssc** problem. It is well known that a greedy algorithm is a 4-approximation and therefore we compared our solution to the greedy algorithm instead of the optimal.

We introduced an algorithm that achieves a cost of $\mathcal{O}(n\sqrt{n\mathcal{E}}\log^2 n)$, where $\mathcal{E}$ is the expected cost for the greedy **mssc** algorithm to cover a single element. Since the total cost of the greedy **mssc** algorithm is $n\mathcal{E}$, the quasi competitive ratio of our algorithm is $\mathcal{O}\left((\sqrt{n}\log^2 n)/\sqrt{\mathcal{E}}\right)$. Therefore, in the worst case, when $\mathcal{E}$ is a constant, our algorithm is $\mathcal{O}(\sqrt{n}\log^2 n)$-quasi competitive.

We also showed that any online algorithm has a cost of $\Omega(n\sqrt{n\mathcal{E}})$. Therefore, our quasi-competitive ratio is within $\mathcal{O}(\log^2 n)$ multiplicative factor from the best possible.

References

1. Susanne Albers. A Competitive Analysis of the List Update Problem with Lookahead. *Theoretical Computer Science*, 197:95–109, 1998.
2. Noga Alon, Baruch Awerbuch, Yossi Azar, and Boaz Patt-Shamir. Tell Me Who I Am: an Interactive Recommendation System. In *18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2006.
3. Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Mark Tuttle. Collaboration of Untrusting Peers with Changing Interests. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, 2004.
4. Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Mark R. Tuttle. Improved Recommendation Systems. In *16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.
5. Yossi Azar and Iftah Gamzu. Ranking with Submodular Valuations. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1070–1079, 2011.
6. Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive Ordering of Pipelined Stream Filters. In *ACM SIGMOD International Conference on Management of Data*, 2004.

7. Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On Chromatic Sums and Distributed Resource Allocation. *Information and Computation*, 140(2):183–202, 1998.
8. Brian Dean, Michel Goemans, and Jan Vondrák. Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity. *Mathematics of Operations Research*, 33:945–964, 2008.
9. Petros Drineas, Iordanis Kerenidis, and Prabhakar Raghavan. Competitive Recommendation Systems. In *34th ACM Symposium on Theory of Computing (STOC)*, 2002.
10. Uriel Feige, László Lovász, and Prasad Tetali. Approximating Min Sum Set Cover. *Algorithmica*, 40:219 – 234, 2004.
11. Michel Goemans and Jan Vondrák. Stochastic Covering and Adaptivity. In *Proceedings of the 7th Latin American Conference on Theoretical Informatics (LATIN)*, pages 532–543, 2006.
12. Sally A. Goldman, Robert E. Schapire, and Ronald L. Rivest. Learning Binary Relations and Total Orders. *SIAM Journal of Computing*, 20(3):245 – 271, 1993.
13. Daniel Golovin and Andreas Krause. Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization. *Journal of Artificial Intelligence Research (JAIR)*, 42:427–486, 2011.
14. Edward Grove. Online Bin Packing with Lookahead. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete algorithms*, 1995.
15. Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Approximation Algorithms for Optimal Decision Trees and Adaptive TSP Problems. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 690–701. Springer-Verlag, 2010.
16. Haim Kaplan, Eyal Kushilevitz, and Yishay Mansour. Learning with Attribute Costs. In *37th ACM Symposium on Theory of Computing (STOC)*, 2005.
17. Zhen Liu, Srinivasan Parthasarathy, Anand Ranganathan, and Hao Yang. Near-Optimal Algorithms for Shared Filter Evaluation in Data Stream Systems. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008.
18. Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The Pipelined Set Cover Problem. In *10th International Conference on Database Theory (ICDT)*, volume 3363. Springer Berlin / Heidelberg, 2005.