On Competitive Recommendations

Jara Uitto[a,1,*], Roger Wattenhofer[a]

[a] *ETH Zürich, Gloriastrasse 35, 8092 Zurich, Switzerland*

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 in this sense.

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 entries do we have to query (probe) 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. The customers come to the book store in a successive manner and once a customer enters the store, the task of the book store keeper is to suggest a book to her. The customer provides the book store keeper with a (binary) review of the suggested book. Once a customer finds a book that she likes, i.e., the book got a positive review, the customer is considered satisfied. The goal is to satisfy all customers with as few queries as possible.

Naturally, satisfying a customer who does not like any book is impossible and therefore, we assume that the *preference vector* of any customer u, i.e., the row in the preference matrix that corresponds to the preferences of u, contains at least one 1-entry. 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

*Corresponding author

Email address: juitto@tik.ee.ethz.ch (Jara Uitto)

[1] +41 44 63 20417

is to minimize the effort required from the customers, that is, the number of queries until all customers have found a book that they like. 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.

Our first observation is that there are instances of our recommendation problem, where any online algorithm performs badly. An example of such an input is a matrix with one 1-entry on each row at a random position. In this example, the rows share no mutual information and therefore, the best any online algorithm can do is to recommend random unrevealed books to the customers until they are satisfied. This indicates that the cost for any online algorithm is $\Omega(n \cdot m)$ in the worst case. Therefore, instead of looking only at the worst case input, we analyze our online algorithm in a competitive manner.

In a 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 recommend each customer a book she likes, resulting in a cost of n for any input. On the other hand, if each customer likes only one book chosen independently at random, any online algorithm will have to recommend $\Omega(m)$ books to each customer before finding a book she likes. Therefore, the competitive ratio of any online algorithm against the optimal offline algorithm is $\Omega(m)$.

Since any online algorithm would perform badly in comparison to the optimal offline algorithm that knows the input completely, we choose a weaker offline algorithm, referred to as *quasi-offline* algorithm and compare our algorithm against the weaker algorithm. 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. We hide the preference matrix and only show the quasi-offline algorithm a probability distribution D over possible preference vectors and the number of customers n. For every execution of the quasi-offline algorithm, the preference vectors for the n 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. The input of **mssc** is, similarly to the well-known Set Cover problem, a collection of sets, but the output is an ordered list of the sets. This order induces a cost for each element, where the cost 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. To connect our problem to **mssc**, we identify each customer with an element and each book with the set of customers that like it.

Consider a matrix where everyone likes book b and there are $m-1$ books that no one likes. The optimal quasi-offline algorithm simply proposes book b to everyone, resulting in a cost of n in total. On the other hand, an online algorithm that does not know which book is the popular one can be forced to do $\Omega(m)$ queries to find a single 1-entry in the matrix. This indicates that the number of books dominates the running time of any online algorithm while not affecting the cost of the quasi-offline algorithm. To make the cost of the online algorithm comparable to the quasi-offline algorithm, we assume throughout the paper that the number of books is not much larger than the number of customers, i.e., $m \in \mathcal{O}(n)$. We emphasize that besides this restriction and assuming that the input is feasible, i.e., every customer likes at least one book, we do not assume anything from the input.

We call the analysis against the quasi-offline algorithm *quasi-competitive*.

Definition 1 (Quasi-Competitiveness)**.** *Let A be an online algorithm. 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.

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 of 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.

2. Related Work

Our problem is a variant of an online recommendation problem. Learning recommendation systems, where the goal is to learn a good estimation of the whole preference matrix, has been studied by Alon et al. [2]. They showed that with high probability[2], one can learn the matrix with little error in polylogarithmic time in a distributed setting given that there are similarities between the preferences of different users. The main connection between their work and ours is the model, i.e., the task is to learn properties of the input matrix by probing the entries of the matrix and to minimize the number of probes. The main difference is in the objective and in the complexity measure. They are learning the matrix to distinguish between different users and to thereby provide good individual recommendations, whereas we are not interested in all the items a user likes and only aim to find some intersecting preferences to find a single item of interest for everyone. Furthermore, we benchmark our algorithm against **mssc**, which overcomes the trivial worst-case lower bound of $\Omega(n^2)$ in the case where there are no mutual information between the rows.

Our recommendation problem is essentially a special case of learning a binary relation. Goldman et al. studied a more general version of the problem, where the task of the learner is to predict given entries in the matrix. They showed that with an arbitrary input, the learner can be forced to make $\Omega(n^2)$ mistakes [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 entry to look at. Furthermore, Kaplan et al. [16] gave more general bounds for similar learning tasks by converting the input of **mssc** 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.

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 and learn the preference vectors of the committee members 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 an (quasi-)offline algorithm. We also show that the worst case performance of our online 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 measure the cost of their algorithm by the number of entries revealed during the execution. They also considered Byzantine users and restricted access to items. 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 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.

[2]We say that an event occurs *with high probability*, if the event occurs with probability at least $1 - n^{-c}$, where c is an arbitrarily large constant.

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.

Since **mssc** provides an ordering of items according to their value, it has applications to ranking problems. 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 greedily, i.e., the set with the maximal contribution is selected. 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].

3. Model

We begin defining our model by giving a formal description of our recommendation problem. The input for our recommendation problem consists of a set of customers $\{u_1, \ldots, u_n\} = U$, a set of books B, and a hidden probability distribution D, where initially $|U| = n$ and $|B| = m \in \mathcal{O}(n)$. The customers are *labeled* by their indices, i.e., a recommendation algorithm can distinguish users from each other. A preference vector of a user is a binary vector of length m with at least one 1-entry. Each customer is assigned a hidden preference vector chosen independently at random from D, where D is a probability distribution over all possible preference vectors. For simplicity, we assume that the probabilities of D are rational.

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 a book $b \in B$ to u, which is equivalent to checking whether u likes b or not. 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.

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

The probability distribution D is chosen by an adversary and the choice of the adversary is allowed to depend on the online algorithm. The choice of D directly induces a cost to the optimal quasi-offline algorithm. We emphasize that prior to the execution, an online recommendation algorithm A has no knowledge of neither D nor the random assignment of the preference vectors and A can only gain information about these parameters by querying the customers. We measure the quality of A with respect to its quasi-competitive ratio, i.e., maximum ratio between the cost of the online recommendation algorithm A and the cost of the optimal quasi-offline algorithm.

An important concept throughout the paper is the *popularity* of a book. The popularity of a book is the number of unsatisfied customers that like it.

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

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

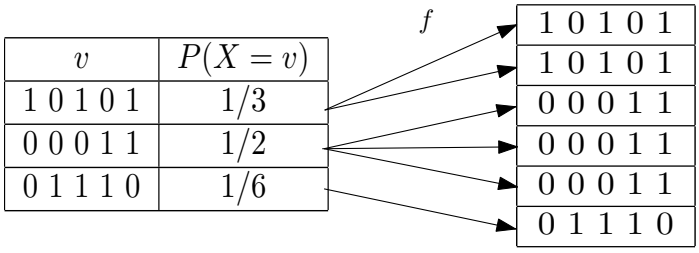

Figure 1: An input $I = (D, 6)$ for the quasi-offline algorithm (on the left) is transformed into an instance of **mssc**. The preference vectors that are assigned with non-zero probability are illustrated in the column is denoted by v. The probability of assigning the corresponding vector to a customer in the column denoted by $P(X = v)$, where X is a random variable that obeys distribution D.

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. We begin this section by showing a connection between the optimal quasi-offline algorithm and the optimal algorithm for **mssc** problem that allows us to benchmark our online algorithm directly against the optimal algorithm for **mssc**.

We observe that 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. In other words, the recommendation strategy for any customer u is an ordered set of books that are successively recommended to u. Therefore, to minimize the total expected cost, it is enough for the quasi-offline algorithm to find an ordered set of books that minimizes the expected cost for a single customer.

To identify an input $I = (D, n)$ of the quasi-offline algorithm with an input of **mssc**, let X be a random variable that obeys distribution D. The idea is to construct a function f that maps I to an input $f(I)$ for **mssc**. Let N be the smallest integer such that $P[X = v] \cdot N$ is an integer for every preference vector v and $N \geq n$. Recall that we assumed the probabilities in D to be rational and thus, N always exists. Now to construct an instance for **mssc**, let us again consider the books as sets and the customers as elements. For every preference vector v, let there be $P[X = v] \cdot N$ customers that have preference vector v. The construction is illustrated in Figure 1.

We denote the ordered set of books chosen by the quasi-offline algorithm by $\mathcal{C}(I) = b_1, b_2, \ldots, b_k$. Let $S_i(I)$ be the set of customers satisfied by the quasi-offline algorithm with book b_i, i.e.,

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

We refer to the size of $S_i(I)$ as the *disjoint popularity* of b_i. The average number of queries $\mathcal{E}(I)$ required to satisfy any customer by the quasi-offline algorithm is given by

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

We note that $\mathcal{E}(I)$ is also the expected cost for the quasi-offline algorithm to satisfy a randomly picked customer. To simplify notation, we write $\mathcal{E} = \mathcal{E}(I)$ whenever the exact input is either irrelevant or clear from the context.

Observation 3. *Let $I = (D, n)$ be an input of the quasi-offline algorithm. Then $\mathcal{E}(I) = c(I)/n = c(f(I))/N$, where $c(\cdot)$ is the cost function of the optimal quasi-offline algorithm and the optimal* **mssc** *algorithm, respectively.*

Proof. To prove the claim, let us first consider the solution $C(I) = b_1, \ldots, b_k$ given by the optimal quasi-offline algorithm. For every $b_i \in C(I)$, where $|S_i(I)|$ is the disjoint popularity of this book, there are exactly $(|S_i(I)| \cdot N)/n$ customers in $f(I)$ who like b_i and do not like b_j for any $j < i$. Therefore, the average cost per customer of the optimal **mssc** algorithm is given by

$$\frac{c(f(I))}{N} = \frac{1}{N}\sum_{i=1}^{k} i\frac{N}{n}|S_i(I)| = \frac{1}{n}\sum_{i=1}^{k} i \cdot |S_i(I)| = \mathcal{E}(I) \ .$$

□

It has been shown that a greedy algorithm, that successively selects sets that cover as many uncovered elements as possible, yields a 4-approximation to the optimal solution for **mssc** [7]. Therefore, we lose only an additional constant factor overhead by comparing our solution to the greedy one instead of the optimal. From here on, we refer to the greedy quasi-offline algorithm for **mssc** as the quasi-offline algorithm.

We use the rest of the section to present general bounds on the introduced concepts for any fixed input I, and thus omit the input from the notation, which we will later use in the analysis of our recommendation algorithm. First we give an upper bound for the number of books of certain disjoint popularity in $\mathcal{C}$.

Lemma 4. *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 therefore,

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

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

Next 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 5. *Let $r \in \mathbb{R}$ be such that n^r is the popularity of the most popular book. 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 that like 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 4, 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 customers 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

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

□

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 customers, we get a positive feedback after $\mathcal{O}(m)$ queries in expectation regardless of the number of unsatisfied customers.

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 6. *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 4 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 an 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 7. *Any online recommendation algorithm needs $\Omega(n\sqrt{n\mathcal{E}})$ queries in expectation to satisfy all customers in the worst case.*

Proof. Let $H = (U, B, D)$ be an input to the recommendation problem, where U is the set of customers, B the set of books, D the probability distribution, and assume that there are $|U| = n$ customers and $|B| = n$ books. In addition, let $1 \leq F \leq n$. The distribution D over the preference vectors is chosen in the following manner: There is one distinguished book $b_p \in B$ and $k = \sqrt{nF}$ books $b_i \neq b_p$, where $1 \leq i \leq k$. Book b_p is liked with probability $p = (n - \sqrt{nF})/n$ and book b_i is liked with probability $1/n$ for all i. Any other book is liked with probability 0. One possible outcome of the preferences of the customers in input H is illustrated in Figure 2. By Observation 3, when $n \to \infty$, we can write average cost to satisfy a single customer by the quasi-offline algorithm as

$$\mathcal{E} \leq \frac{1}{n}\left(n - \sqrt{nF} + \sum_{i=2}^{\sqrt{nF}+1} i\right) \in \mathcal{O}(F) \ .$$

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

Figure 2: 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 $\Omega(n)$ 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.

Let $U' \subseteq U$ be the set of the customers that do not like the popular book b_p. Consider now only the customers in U' and let A be a deterministic online algorithm that solves the recommendation problem. Let $u \in U'$ be some customer and let $\mathcal{H}$ be the set of all possible outcomes of the random choices of the preference vectors of customers in U'. When averaging over $\mathcal{H}$, the average number of queries needed to satisfy u is at least

$$\frac{1}{|B|-1} \cdot \left(\frac{1}{2} \cdot (|B|-1) \cdot |B| \right) = \frac{|B|}{2} = \frac{n}{2} \in \Omega(n) \ ,$$

for any deterministic online recommendation algorithm.

Since the preference vector for u is chosen independently from other customers, the expected number of queries needed to satisfy u is also independent from queries to other customers. By Yao's principle [19], any randomized online recommendation algorithm thus requires $\Omega(n\sqrt{n\mathcal{E}})$ queries in expectation to satisfy all customers in U'. □

5.2. Customers with Multiple Preferences

Now we lift the assumption that each customer likes exactly one book. The basic idea behind our algorithm for the more general version of the problem is similar to the one preference case. However, by randomly sampling and fixing the first book with a positive feedback, we might end up recommending a bad book to many customers that do not like it. As an example, consider an instance where everyone likes one book and there are $n/2$ unpopular books that are liked by $\log n$ customers. In this example, it is likely that we find many 1-entries from the unpopular books before recommending the most popular book to everyone.

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 determines the error rate of the choice. All the sampling information is stored in an integer 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 $c \log n$

Algorithm 1 Phases(U, B)

$M \leftarrow$ a zero matrix of size $|U| \times |B|$.
STATE $\leftarrow$ sample.
$V \leftarrow$ a vector of length n such that $V(u) = 1$ for all $u \in U$
$b^* \leftarrow$ an arbitrary book $b \in B$.
while there are unsatisfied customers **do**
  Receive a random customer $u \in U$.
  if STATE = sample **then**
    Run Sampling(u, M, V, b^*, U, B).
  else
    Run Greedy(u, M, V, b^*, U, B).
  end if
end while

Algorithm 2 Sampling(u, M, V, b^*, U, B)

Choose a random book $b \in B$.
if u likes b, i.e., $u(b) = 1$ **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, M, V, b^*, U, B)

if $V(u') = 1$ for all $u' \in U$ **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^*, i.e., $u(b^*) = 1$ **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 to a zero matrix.
end if

positive feedbacks 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, i.e., books that are liked by the same customers, 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-code for the sampling part is given in Algorithm 2 and for the greedy part in Algorithm 3. Note that while sampling, customers are not removed and thus, the same customer can give several positive feedbacks on the same book, i.e., even a single customer 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 on the popularities of the books in cat_i as the *size* of the corresponding category.

Lemma 8. *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 on 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, i.e., $c \log n$, on one book before the greedy part of phase j begins. 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

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

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 9. *Let i be the largest integer such that cat_i is non-empty. 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 to Lemma 8 we can use the Chernoff bound to show that with enough sampling, $c \log n$ is at most within factor 3/2 from $E[X_b]$, i.e., $(3/2)E[X_b] \geq c \log n$. By the definition of categories we get that $E[X_{b'}] > E[X_b]/2$. Therefore, the Chernoff 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 $E[X_{b'}] \cdot (3/4)$.

Therefore, with high probability

$$X'_b > \frac{3E[X'_b]}{4} > \frac{3E[X_b]}{8} \geq \frac{c \log n}{4} .$$

Since books are chosen greedily until there are no more books with at least $(c \log n)/4$ positive feedbacks, 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

It follows from Lemma 9 that after $\mathcal{O}(\log n)$ phases, the popularity of the most popular book reduces to zero, i.e., all customers are satisfied. Since we now have derived 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 10. *During each phase, Algorithm 2 is called $\mathcal{O}(n\sqrt{\mathcal{E}n}\log n)$ times in expectation and with high probability to get $c\log n$ positive feedbacks on some book by Algorithm 2.*

Proof. Consider phase i and let b be the most popular book in the beginning of this phase. The probability of receiving a random customer that likes book b is

$$\frac{|b|}{|U|} \geq \frac{|b|}{3\sqrt{\mathcal{E}n|b|}} = \frac{\sqrt{|b|}}{3\sqrt{\mathcal{E}n}} \geq \frac{1}{3\sqrt{\mathcal{E}n}}$$

by Lemma 5. Furthermore, the probability of choosing any specific book randomly is $1/m$. Therefore, the expected amount of times book b is recommended to customers that like it after $3m\sqrt{\mathcal{E}n}\cdot c\log n$ queries is at least

$$m3\sqrt{\mathcal{E}n}\cdot c\log n\cdot\frac{1}{m3\sqrt{\mathcal{E}n}} = c\log n\ .$$

Let X_k be the random variable that counts the number of times b is recommended to customers that like b after k queries during phase i. By applying a Chernoff bound, we get that

$$P\left(X_k < c\log n\right) < \mathcal{O}\left(n^{-c}\right)\ ,$$

for any $k > 2\cdot(3m\sqrt{\mathcal{E}n}\cdot c\log n)$. Since $2\cdot(m3\sqrt{\mathcal{E}n}\cdot c\log n)\in\mathcal{O}(n\sqrt{\mathcal{E}n}\log n)$, the claim follows. □

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

Lemma 11. *Running one phase of Algorithm 1 costs $\mathcal{O}(n\sqrt{\mathcal{E}n}\log n)$ in expectation and with high probability.*

Proof. By Lemma 10 the cost of the sampling part of any phase is $\mathcal{O}(n\sqrt{\mathcal{E}n}\log n)$ in expectation and with high probability after which the greedy state is assumed.

By Lemma 8 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 5 there are at most $\mathcal{O}(\sqrt{\mathcal{E}n2^i})$ unsatisfied customers left in the beginning of the phase. Therefore, after making $\mathcal{O}(\sqrt{\mathcal{E}n})$ greedy choices either all customers 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 and with high probability for the greedy part to recommend some book to every customer, therefore the number of calls to the greedy subroutine is

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

in expectation and with high probability. □

Theorem 12. *Algorithm 1 is $\mathcal{O}(\sqrt{n}\log^2 n)$-quasi competitive.*

Proof. In the beginning of the execution, the popularity of the most popular book is at most $n = 2^{\log n}$, i.e., the largest index of a non-empty category is at most $\log n$. By observing that the popularity of a book never increases and by applying Lemma 9, we get that the largest index of a non-empty category decreases by at least one in every phase with high probability. Therefore, after $\mathcal{O}(\log n)$ phases the largest index of a non-empty category is 0 with high probability and thus, there can not be any more unsatisfied customers.

As the cost of each phase is $\mathcal{O}(n\sqrt{\mathcal{E}n}\log n)$ by Lemma 11, 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

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

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 online 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 for **mssc** 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

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

Therefore, the worst case is obtained when $\mathcal{E}$ is a constant and it follows that 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 *Proceedings of the 18th Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 261–279, 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*, pages 112–119, 2004.

[4] Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Mark R. Tuttle. Improved Recommendation Systems. In *Proceedings of the 16th Symposium on Discrete Algorithms (SODA)*, pages 1174–1183, 2005.

[5] Yossi Azar and Iftah Gamzu. Ranking with Submodular Valuations. In *Proceedings of the 22nd Symposium on Discrete Algorithms (SODA)*, 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*, pages 407–418, 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 *Proceedings of the 34th Symposium on Theory of Computing (STOC)*, pages 82–90, 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 6th Symposium on Discrete Algorithms (SODA)*, pages 430–436, 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 *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, pages 356–365, 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*, pages 133–146, 2008.

[18] Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. The Pipelined Set Cover Problem. In *Proceedings of the 10th International Conference on Database Theory (ICDT)*, pages 83–98, 2005.

[19] Andrew Chi-Chin Yao. Probabilistic Computations: Toward a Unified Measure of Complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, 1977.