Theoretical Computer Science 620 (2016) 4–14

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On competitive recommendations

Jara Uitto *, Roger Wattenhofer

ETH Zürich, Gloriastrasse 35, 8092 Zurich, Switzerland

ARTICLE INFO

Article history:
Available online 30 October 2015

Keywords:
Learning
Online
Recommendation
Algorithms

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.

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 is to minimize the effort required from the customers, that is,

* Corresponding author. Tel.: +41 44 63 20417.
E-mail address: juitto@tik.ee.ethz.ch (J. Uitto).

http://dx.doi.org/10.1016/j.tcs.2015.10.034
0304-3975/

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 in 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

$$c(A(I)) \leq \alpha \cdot c(\mathrm{OPT}_q(I)) + \mathcal{O}(1) ,$$

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,[1] 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.

[1] 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.