Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Finding Similar Users in Social Networks

Aviv Nisgav · Boaz Patt-Shamir

Received: date / Accepted: date

Abstract We consider a system where users wish to find similar users. To model similarity, we assume the existence of a set of queries, and two users are deemed similar if their answers to these queries are (mostly) identical. Technically, each user has a vector of preferences (answers to queries), and two users are similar if their preference vectors differ in only a few coordinates. The preferences are unknown to the system initially, and the goal of the algorithm is to classify the users into classes of roughly the same preferences by asking each user to answer the least possible number of queries. We prove nearly matching lower and upper bounds on the maximal number of queries required to solve the problem. Specifically, we present an "anytime" algorithm that asks each user at most one query in each round, while maintaining a partition of the users. The quality of the partition improves over time: for n users and time T, groups of $\tilde{O}(n/T)$ users with the same preferences will be separated (with high probability) if they differ in sufficiently many queries. We present a lower bound that matches the upper bound, up to a constant factor, for nearly all possible distances between user groups.

Keywords recommendation systems · collaborative filtering · randomized algorithms · user classification · market segmentation

1 Introduction

Classical research in social networks tries to analyze their structure and evolution from the observer's viewpoint [16, 7]. Recently, with the emergence of Internet-based social networks such as Facebook [6], MySpace [12], and many others, social network systems include additional mechanisms to facilitate evolution. In particular, the system may try to help users to find other "compatible" users. Compatibility usually means having similar taste, where taste is broadly interpreted as preferences in a specific domain (such as musical taste, professional expertise, town of origin etc.), or a

Aviv Nisgav · Boaz Patt-Shamir
School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel.

combination of them. While some users are network-savvy and can find their similar peers on their own, there are other users who can't navigate the net well enough to do it. It is therefore quite useful for social networks to have an automated tool which helps the interested users to classify themselves into groups of similar characteristics. However, while users may be willing to cooperate, such cooperation is limited: users are typically reluctant to answer too many intimate questions, even if privacy is unequivocally promised.

In this paper we study this premise from the theoretical perspective. We propose a simple model, prove a lower bound on the least possible number of queries required by any classification algorithm, and present a randomized algorithm which is guaranteed (with high probability) to classify the users in groups of similar taste at a cost that nearly meets the lower bound.

Specifically, we model the user classification problem as follows. We assume that there are n users and m possible queries. Each user has an answer for each query, but these answers are not known when the algorithm starts. The answers of each user are modeled as a *preference vector*, where coordinates correspond to queries and entries correspond to answers. The queries are abstract in that the only thing assumed about them is that they have a finite set of answers (say, a query is a multiple-choice question). Algorithms are assumed to operate in rounds, where in each round some users are presented with queries determined by the algorithm based on past answers, with the possible help of coin tosses. The output of the algorithm is a label for each user, which serves as that user's "type identifier."

The quality of an algorithm is measured by two criteria. First, performance is judged by the quality of the output partition (in a way defined precisely in Section 2 below). And second, cost is measured by *query complexity*, namely the maximal number of queries the algorithm asks any particular user.

In this paper, we present the following results. Say that a user belongs to a type of popularity α if at least αn of the users share his preference vector. We give an algorithm that continuously refines the user partition by separating types that are either sufficiently popular or sufficiently different. For example, at time T, with high probability, users whose type popularity is at least $O(\frac{\log n}{Td})$ are separated if their opinions differ on at least d queries, for $1 \leq d \leq \sqrt{\log n}$. We also prove bounds for $d > \sqrt{\log n}$: A precise statement is provided in Theorem 1 in Section 2. We also present a lower bound that shows that our results are optimal (up to a constant factor) for nearly all values of d. A precise statement of the lower bound is provided in Theorem 2 in Section 2. A schematic summary of our results is presented in Figure 1.

Related Work. Our model is closely related to models of recommender systems [4, 5, 3]. Recommender systems are motivated by e-commerce websites such as Amazon and Netflix. It is assumed that there is a user-product matrix, where each entry is the grade a user gives to a product, and the goal is to find which product the user will like. Some variants of recommender systems assume that the matrix entries are partially known (say, they reflect past activity of the user), and the task of the algorithm is to predict unknown entries based on the given data. Other models assume that all entries are unknown, and the task of the algorithm is to direct the users which products to try

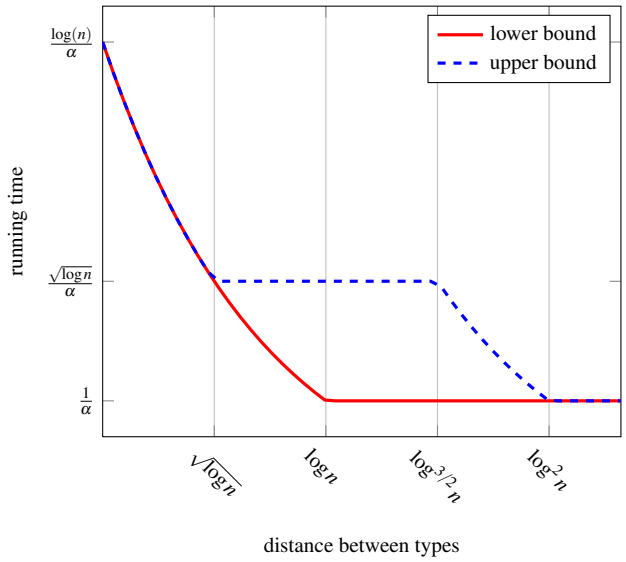

Fig. 1 *Schematic representation of the results, showing the running time as a function of the separation distance for fixed popularity α and $m = n$ (ignoring constant factors). The solid line shows the lower bound; the dashed line shows the upper bound (visible where it differs from the lower bound).*

(thereby revealing some entries of the matrix) so that the algorithm can recommend a good product more effectively [14].

In the former model, where a partial matrix is given, it is common to assume a linear model for user taste and product features, and apply algebraic techniques such as principal component analysis [8] or singular value decomposition (SVD) [15] to predict the missing entries. Papadimitriou et al. [13], and Azar et al. [4] rigorously prove conditions under which SVD is effective. Other generative user models that were considered include simple Markov chain models [10, 11], where users randomly select their "type," and each type is a probability distribution over the objects.

Drineas et al. [5] were the first to propose a competitive model, where the algorithm directs the users which products to try and the results of the tries are fed back to the algorithm. In [3] it was shown that in this model, a user can find a product he likes in $O(\log n/\alpha)$ tries, where α is the popularity of that user's taste.

The paper most closely related to our current work is the one by Awerbuch et al. [2]. In that paper, each user has an unknown preference vector, and the task is to reconstruct these vectors. Obviously, an algorithm solving this task also solves the user classification problem as a by-product: one can use the computed preference vectors as user labels. It is therefore interesting to compare the performance of the algorithms. Put in our terminology, the algorithm in [2] has query complexity $\Theta(\log n/\alpha)$, which is always worse than our algorithm, whose query complexity depends on the distance between the preference vectors to be separated.

Alon et al. [1] present an algorithm where user preference vectors are computed approximately using similar users in a competitive way. In that algorithm, the number of queries to a user with αn similar users is $O(\log^{3.5} n/\alpha^2)$, which is worse than ours.

Organization. The remainder of this paper is organized as follows. In Section 2 we present the model and state the main results. In Section 3 we present the algorithm. In Section 4 we prove the lower bound.

2 Formal Statements of the Problem and Results

The user classification problem is defined as follows. There are *n users* and *m queries*. Each user has a binary *answer* for each query. The vector of a given user's answers on all queries is called that user's *preference vector*. Users that have the same preference vector are said to belong to the same *type*. The *popularity* of a given type is said to be α for some $0 \leq \alpha \leq 1$ if αn of the users belong to that type.

The input to an algorithm is the number of users n and the number of queries m (the preference vectors are not part of the input); the output is an assignment of a *label* to each user. The algorithm proceeds in *rounds*, where in each round, the algorithm may ask some users to answer some queries: the algorithm may present at most one query to each user in a round. The maximal number of queries a user is requested to answer is the *query complexity* of the algorithm.

The goal of the algorithm is that the output labels will induce a good partition on the users, while keeping the query complexity as small as possible. We now define more precisely the notion of "good" partitions. To this end, we use the standard concept of Hamming distance: Let $\mathscr{D}$ be an arbitrary domain, and let $m > 0$ be an integer. The distance between two vectors $v_1, v_2 \in \mathscr{D}^m$, denoted $\mathrm{dist}(v_1, v_2)$, is the number of coordinates in which they differ.

Definition 1 Let $V \subseteq \mathscr{D}^m$ be a set of vectors where $\mathscr{D}$ is some domain, and let $L : V \to \mathscr{L}$ be a labeling function mapping each vector in V to a label from a set $\mathscr{L}$. Then L is an *ε-separating partition* for a given real number $0 \leq \varepsilon \leq 1$, if for all $v_1, v_2 \in V$ it holds that

- $\mathrm{dist}(v_1, v_2) = 0$ implies $L(v_1) = L(v_2)$, and
- $\mathrm{dist}(v_1, v_2) > \varepsilon m$ implies $L(v_1) \neq L(v_2)$.

Intuitively, we would like ε to be small: vectors that differ in d coordinates for some $d \leq \varepsilon m$ may or may not get the same label by an ε-separating partition.

A trivial 1-separating partition that labels all users by the same label always exists. Also, a perfect 0-separating partition can be attained by asking each user all m queries, allowing us to label each user by its complete users preference vector. Much better performance is guaranteed by Algorithm D in [2], which finds, using query complexity $O(\lceil \frac{m}{n} \rceil \log n/\alpha)$, a 0-separating partition of the users with type popularity at least α (that algorithm may fail with probability $n^{-\Omega(1)}$).

By contrast, the main result of this paper is the following.

Theorem 1 *Let n be the number of users, and let m be the number of queries. Denote $Z = m/n$. Then after $O(T)$ steps, with probability at least $1 - n^{-\Omega(1)}$, Algorithm*

Sep_Any of Section 3 finds a $\frac{d}{m}$-separating partition for all users whose type popularity is at least $\alpha(d)$, where

$$\alpha(d) = \begin{cases} \frac{\lceil Z \rceil}{T} \cdot \frac{\log n}{d} & 1 \leq d < \sqrt{\log n} \\ \frac{\lceil Z \rceil}{T} \cdot \sqrt{\log n} & \sqrt{\log n} \leq d < (\log n)^{3/2} \\ \frac{\lceil Z \rceil}{T} \cdot \frac{(\log n)^2}{d} & (\log n)^{3/2} \leq d < (\log n)^2 \\ \frac{\lceil Z \rceil}{T} & (\log n)^2 \leq d \end{cases}$$

Note that for a given running time, Algorithm Sep_Any provides better separation guarantees for larger types. Figure 1 shows the running time as a function of the separation distance for fixed popularity.

On the other hand, we have the following lower bound.

Theorem 2 *Let $\frac{\log n}{n} \leq \alpha \leq 1/2$ and $\delta \geq 3^{-\alpha n/2}$. Any algorithm which separates, with probability at least $1-\delta$, users whose preference vectors differ in d coordinates for $d < \alpha n$, and whose type popularity is α, has query complexity $\Omega\left(\frac{1}{\alpha}\left\lceil\frac{\log 1/\delta}{d}\right\rceil\right)$.*

The upper and lower bounds, for $\delta = n^{-\Omega(1)}$, coincide (to within a constant factor) for $d \leq \sqrt{\log n}$ and $d \geq \log^2 n$: see Figure 1 for illustration.

3 Algorithms

In this section we present our main result, namely a labeling algorithm which gives d/m-separation partition for users of large enough types. First, we present Algorithm Sep_Close which is most effective when the distance between users is $\Theta(\sqrt{\frac{m}{n}\log n})$. We use this "sweet spot" for larger distances in Algorithm Sep_Any, which uses Algorithm Sep_Close as a black box.

For notational convenience, our algorithms represent user partitions by the explicit user subsets rather than by labels. The labeling required by the problem specification can be done by labeling all users in a subset by the same unique label.

3.1 Algorithm Sep_Close

We now describe Algorithm Sep_Close, which is most effective for types that differ in $d = O(\sqrt{\frac{m}{n}\log n})$ queries, where m is the number of queries and n is the number of users. (We note that we later use Algorithm Sep_Close as a subroutine within Algorithm Sep_Any, and the number of queries and users may vary there.) Recall that we denote $Z = m/n$. The algorithm ensures (with high probability) that users of any two types of a sufficiently large popularity α which are distance d apart end up at different subsets, while users of the same type are never separated. The algorithm is an "anytime" algorithm in the sense that it continuously refines its output partition: at time T, it ensures separation of types with popularity $\alpha(d,T) = \Omega\left(\frac{1}{T}\left(\frac{Z\log n}{d} + d\right)\right)$. In particular, types at distance $d = \Theta\left(\sqrt{Z\log n}\right)$ are separated at time T if their popularity is at least $\alpha = \Omega\left(\frac{\sqrt{Z\log n}+1}{T}\right)$.

Algorithm 1 `Sep_Close`(U,I) *U is a set of users and I is a set of queries*

```
1: 𝒮 ← {U}
2: loop
3:     for all u ∈ U do
4:         Ask user u to answer a randomly chosen query in I
5:     end for
6:     for all S ∈ 𝒮 do
7:         𝒮 ← 𝒮 \ {S} ∪ Proc_Set(S,I)          // Proc_Set(S,I) is a partition of S
8:     end for
9:     The output (at all times) is the partition 𝒮.
10: end loop
```

Given two user types, define their *distinguishing queries* to be the queries on which their users disagree. The main idea is to try to find, for each yet-unseparated subset of users, a distinguishing query and ask each user in the subset to answer it (initially all users belong to the same subset).

Conceptually, the algorithm operates in two stages: first, each user answers a few random queries so as to ensure "coverage" of the queries by users. In the second stage, the algorithm asks users to answer "controversial" queries, namely queries many users disagree on. Based on their answers to a query, users are split into two subsets, then another controversial query is presented (a different one in each subset), and so on.

Intuitively, the idea in the algorithm is that queries distinguishing between "large" types can be detected as controversial after asking a random sample of the users to answer them. When a query is answered by sufficiently many users, the distribution of the answers in the sample is similar to the true distribution in the entire user population.

However, this idea alone is not sufficient: even if many users disagree about a certain query, this does not necessarily mean that this query distinguishes between two "large" types: it could be the case that users of a large type happen to agree that the answer to that query is '0', and all users of many small types think that the answer to that query is '1'. Nevertheless, we show that answering a controversial query partitions the subset into two relatively large subsets, and hence the algorithm tries such non-distinguishing queries only a bounded number of times.

In an "anytime" algorithm, we cannot run the two conceptual stages of the algorithm (namely of covering all queries by a sample of users, and of posing controversial queries to all users) in serial fashion: instead, the algorithm runs these two tasks in parallel. Consider running the algorithm for T query rounds: after the first $T/2$ rounds, $T/4$ random queries are presented to each user, which ensures a good sample for each query. Therefore, in the following $T/2$ steps, the algorithm can present the $T/4$ most controversial queries, which ensure, with high probability, that users from types which are large enough and far enough will be separated.

Pseudocode of the algorithm is given in Algorithm 1. It works as follows. Initially all users are in the same subset. At odd rounds each user answers one query at random, and at even rounds, the subsets are partitioned using the routine `Proc_Set`, whose pseudocode is presented in Algorithm 2. Specifically, partitioning a subset

Algorithm 2 `Proc_Set`(S,I) *S is a set of users, I is a set of queries*

1: Consider only users in S.
2: **for all** $i \in I$ **do**
3: Let Zeros(i) and Ones(i) be the number of times query i was answered '0' and '1', respectively.
4: Let $m_i^S = \min\{\text{Zeros}(i), \text{Ones}(i)\}$
5: **end for**
6: Let controversial$(S) = \text{argmax}_i(m_i^S)$
7: **for all** $u \in S$ **do**
8: Ask user u to answer query controversial(S).
9: **end for**
10: Let S_0 and S_1 be the sets of users who answer controversial(S) with '0' and '1', respectively.
11: **return** $\{S_0, S_1\}$.

of users is done according to their opinion on the query with the *maximal minority* (maxmin), defined as follows. Given a set of users and some of their answers to a set of queries, for each query define the majority and minority answers to be the most and, respectively, the least popular answer to the query among the given users. The maxmin query is the query with the largest number of users answering the minority answer. Note that the definition of the maximal minority query ignores the size of the majorities and the total number of users answering a query.

The output of the algorithm, for each user at any time, is the subset it belongs to. Note that even if two users answer differently on a query at Step 4 in `Sep_Close` they still might end up in the same subset if this query is not "controversial" enough and therefore never presented by routine `Proc_Set`. On the other hand, all users in a subset have answered exactly the same set of controversial queries, and they agreed on all of them.

We now analyze Algorithm 1. The following theorem shows the main property of the algorithm: At time T the algorithm labels differently users of types which are sufficiently big and distinct.

Theorem 3 *Let $D = O(\log n)$. At any time $T > 0$ users of the same type are in the same subset. Furthermore, with probability $1 - n^{-\Omega(1)}$ any two types at distance $d \geq D$ from each other and with popularity $\alpha = \Omega\left(\frac{Z \log n}{TD} + \frac{D}{T}\right)$ are in different subsets.*

Informally, Theorem 3 says that as time progresses, the algorithm distinguishes between distinct types of smaller and smaller popularity.

Before we prove Theorem 3, we state a useful corollary.

Corollary 1 *At any time $T > 0$ users of the same type are in the same subset. Furthermore, with probability $1 - n^{-\Omega(1)}$, for $Z = O(\log n)$ and $\alpha T = \Omega\left(1 + \sqrt{Z \log n}\right)$, the algorithm separates any two types of popularity α at distance $d \geq \sqrt{Z \log n}$.*

Proof The corollary follows directly from Theorem 3 with $D = \left\lceil \sqrt{Z \log n} \right\rceil$. □

We now commence with the proof of Theorem 3. Fix T and D, and let $\alpha = c\left(\frac{Z \log n}{DT} + \frac{D}{T}\right)$, for a constant c we define later. In our proofs, we focus on any two α-types, denoted A and B henceforth. We abuse notation slightly and use A (or B)

to refer both to the set of users and to their common preference vector. The theorem follows from Lemmas 1 and 3.

First we show the easy direction of Theorem 3.

Lemma 1 *Users of the same type are always in the same subset.*

Proof All users start at the same set and are separated only if they disagree on some query. As all users of a specific type agree on all queries, they are never separated by `Proc_Set`. □

We now turn to show separation. We will use the following notation. At any step of the algorithm, for each query i and subset S of users, m_i^S is the number of users in S who answered i with minority answer in Step 4 of Algorithm 1 (the random sampling step). Further, let $\text{opinion}_0^S(i)$ and $\text{opinion}_1^S(i)$ be the total number of users in S whose ith coordinate in the preference vector is 0 and 1, respectively (these numbers are the true distribution of opinions in the population of S). Let $M_i^S = \min\{\text{opinion}_0^S(i), \text{opinion}_1^S(i)\}$. (Note that $M_i^S \neq m_i^S$, because m_i^S is the result of a random sample of the users in S, while M_i^S is the true value over the complete user population in S.)

Our strategy to prove the theorem is as follows. Suppose users of A and B belong to the same subset S at time t. If enough users answer each query, then for at least one distinguishing query i^*, m_{i^*} is large. If a non-distinguishing query j is chosen at Step 6 of `Proc_Set`, then $m_j \geq m_{i^*}$ which should imply that M_j is large, which in turn ensures that many users are removed from S at Step 10 of `Proc_Set`. As this removal repeats over and over, after some number of rounds, users of types A and B cannot belong to the same subset.

The analysis is complicated by the fact that the samples are quite small, so we cannot ensure high probability that a particular distinguishing query will be selected; similarly, we cannot ensure that a query with a large minority in the sample indeed reflects a large minority in the actual population. However, we can prove that the desired events occur, if we try sufficiently many times.

The following lemma shows that at time $T/2$, m_i is large for at least one distinguishing query i.

Lemma 2 *Suppose two types with popularity α are in the same subset S at time $t \geq T/2$. Then with probability $1 - n^{-\Omega(1)}$, there exists a distinguishing query i^* such that $m_{i^*}^S \geq \frac{\alpha t}{2Z}(1-\varepsilon)$ for any constant $0 < \varepsilon < 1$.*

Proof Fix a subset S containing two α-types A and B at time t. Let a_i, for $1 \leq i \leq m$ be random variables representing the total number of users of type A that have answered query i by time t. Similarly, let b_i be the total number of users of type B that answered query i by time t. Let I denote the set of d queries distinguishing A from B. For $i \in I$, if both a_i and b_i are larger than $\frac{\alpha t}{2Z}(1-\varepsilon)$ then $m_i \geq \frac{\alpha t}{2Z}(1-\varepsilon)$. We show that with high probability there is at least one such query.

Consider a_i. There are at least αn users of type A in S, and each of these users has answered $t/2$ randomly chosen queries by time t. There are $m = Zn$ queries in all, and hence a_i is a binomial random variable with at least $\frac{\alpha n t}{2}$ trials and success

probability $\frac{1}{Zn}$. Using Chernoff bound and the assumption that $\alpha = c\left(\frac{Z\log n}{DT} + \frac{D}{T}\right)$, we obtain that for any i

$$\begin{aligned}\Pr\left(a_i < \frac{\alpha t}{2Z}(1-\varepsilon)\right) &< \exp\left(-\frac{\alpha t}{2Z}\cdot\frac{\varepsilon^2}{2}\right)\\ &\leq \exp\left(-\frac{\log n}{D}\cdot\frac{c\varepsilon^2}{8}\right).\end{aligned}$$

Similarly, for any i the probability that $b_i < \frac{\alpha t}{2Z}(1-\varepsilon)$ is less than $\exp\left(-\frac{\log n}{D}\cdot\frac{c\varepsilon^2}{8}\right)$.

The choices made by players in type A are independent from the choices made by players of type B hence

$$\begin{aligned}\Pr\left(m_i \geq \frac{\alpha t}{2Z}(1-\varepsilon)\right) &\geq \Pr\left(a_i \geq \frac{\alpha t}{2Z}(1-\varepsilon), b_i \geq \frac{\alpha t}{2Z}(1-\varepsilon)\right)\\ &> \left(1-\exp\left(-\frac{\log n}{D}\cdot\frac{c\varepsilon^2}{8}\right)\right)^2\\ &> 1-2\exp\left(-\frac{\log n}{D}\cdot\frac{c\varepsilon^2}{8}\right).\end{aligned}$$

For any $D < \frac{c\varepsilon^2}{8\ln 2}\log n$ and $d \geq D$ we obtain that the probability $m_i < \frac{\alpha t}{2Z}(1-\varepsilon)$ for all queries $i \in I$ is

$$\begin{aligned}\Pr\left(\bigcap_{i\in I}\left(m_i < \frac{\alpha t}{2Z}(1-\varepsilon)\right)\right) &= \prod_{i\in I}\left(1-\Pr\left(m_i \geq \frac{\alpha t}{2Z}(1-\varepsilon)\right)\right)\\ &< 2^d \exp\left(-\frac{c\varepsilon^2 d}{8D}\log n\right)\\ &= \exp\left(\frac{d}{D}\left(D\ln 2 - \frac{c\varepsilon^2}{8}\log n\right)\right).\end{aligned}$$

Hence for $D = O(\log n)$ the lemma holds for any constant $c > \frac{8\ln 2}{\varepsilon^2}\cdot\frac{D}{\log n}$. □

Lemma 2 gives a bound on m_j^S whenever the algorithm chooses query j at Step 6 in `Proc_Set`. The next lemma gives bound on M_j whenever j is a non-distinguishing query, and hence a lower bound on the number of users removed from the subset after each user in S answer this query.

Lemma 3 *Let $t \geq T/2$ and types A and B be α-types at distance D in the same subset S. If after D invocations of Step 10 in `Proc_Set`,A and B are in the same subset S', then $|S| - |S'| \geq \alpha n(1-\varepsilon)^2$ with probability at least $1 - n^{-\Omega(1)}$ for any positive constant $\varepsilon < 1/2$.*

Proof Let J be the D queries all users of type A answer at Step 10 of `Proc_Set` at the time interval $[t, t+2D]$ (J is well defined by Lemma 1). Denote S_j as the input to `Proc_Set` whenever query j is chosen for users of type A at Step 6. We show that if after D invocations of Step 10 of `Proc_Set`, subset S' contains both

type A and type B users, then with probability $1-2n^{-\Omega(1)}$, there exists $j^* \in J$ such that $M_{j^*}^{S_{j^*}} > \alpha n(1-\varepsilon)^2$ and since $|S'| \leq \max\{M_{j^*}^{S_{j^*}}, |S| - M_{j^*}^{S_{j^*}}\}$ and by definition $M_{j^*}^{S_{j^*}} < |S_{j^*}|/2 \leq |S|/2$, the lemma follows.

Let t_j be the time when query j is chosen for users of type A. By Lemma 2, with probability $1-n^{-\Omega(1)}$, at any time $t \geq T/2$ there exists a query i^* between A,B such that $m_{i^*} \geq \frac{\alpha t}{2Z}(1-\varepsilon)$, and hence whenever a non-distinguishing query j is chosen at Step 6 of `Proc_Set`, $m_j \geq \frac{\alpha t_j}{2Z}(1-\varepsilon)$.

Next we bound the probability that for all $j \in J$ we have $m_j^{S_j} \geq \frac{\alpha t_j}{2Z}(1-\varepsilon)$ when $M_j^{S_j} < \alpha n(1-\varepsilon)^2$.

For any $j \in J$, let X_j be a random variable defined as follows: Consider the execution of `Proc_Set` on S_j and let

$$X_j = \begin{cases} \text{Zeros}(j) & \text{If opinion}_0(j) < \text{opinion}_1(j) \\ \text{Ones}(j) & \text{Otherwise} \end{cases}.$$

Note that X_j is a binomial random variable with $\frac{t_j}{2}|S_j|$ trials while each trial is a success if a user chose query j and his opinion is $M_j^{S_j}$. Hence $E[X_j] = \frac{t_j M_j^{S_j}}{2Zn}$ and by linearity of expectation, we have

$$\begin{aligned} E\left[\sum_{j\in J} X_j \middle| M_j^{S_j} < \alpha n(1-\varepsilon)^2\right] &= \sum_{j\in J} E\left[X_j \middle| M_j^{S_j} < \alpha n(1-\varepsilon)^2\right] \\ &< \frac{\alpha(1-\varepsilon)^2}{2Z}\sum_{j\in J} t_j \qquad (1) \\ &= \frac{\alpha D(t+D/2)}{2Z}(1-\varepsilon)^2 . \end{aligned}$$

Now, if n_j denotes the total number of users in S_j who answer query $j \in J$ then $m_j = \min\{X_j, n_j - X_j\}$ and since $\Pr(m_j > x) = \Pr(x < X_j < n_j - x) \leq \Pr(X_j > x)$ for all $0 \leq x \leq n_j/2$ we have that the probability $m_j \geq \frac{\alpha t_j}{2Z}(1-\varepsilon)$ for all j whenever $M_j^{S_j} < \alpha n(1-\varepsilon)^2$ is

$$\begin{aligned} \Pr\left(\bigcap_{j\in J}\left(m_j \geq \frac{\alpha t_j}{2Z}(1-\varepsilon)\right)\right) &\leq \Pr\left(\sum_{j\in J} m_j \geq \frac{\alpha D(t+D/2)}{2Z}(1-\varepsilon)\right) \\ &\leq \Pr\left(\sum_{j\in J} X_j \geq \frac{\alpha D(t+D/2)}{2Z}(1-\varepsilon)\right) . \end{aligned}$$

To follow standard notation, let us define

$$z \stackrel{\text{def}}{=} \frac{\alpha D(t+D/2)}{2Z}(1-\varepsilon)$$
$$\mu \stackrel{\text{def}}{=} E\left[\sum_{j\in J} X_j \middle| M_j^{S_j} < \alpha n(1-\varepsilon)^2\right]$$
$$\delta \stackrel{\text{def}}{=} \frac{z}{\mu} - 1$$

Note that $\delta > \frac{\varepsilon}{1-\varepsilon} > \varepsilon$ by Eq. (1). Therefore we obtain that

$$\begin{aligned}
\Pr\left(\sum_{j\in J} X_j \geq z \middle| M_j^{S_j} < \alpha n(1-\varepsilon)^2\right) &\leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \\
&= \frac{e^{\delta\mu}}{(1+\delta)^z} \\
&= \exp\left(z\left(\frac{\delta}{1+\delta} - \ln(1+\delta)\right)\right) \\
&\leq \exp\left(z\left(\frac{\varepsilon}{1+\varepsilon} - \ln(1+\varepsilon)\right)\right) \\
&\leq \exp\left(z\left(\frac{\varepsilon}{1+\varepsilon} - \varepsilon + \frac{\varepsilon^2}{2}\right)\right) \\
&= \exp\left(-z\varepsilon^2\frac{1-\varepsilon}{2(1+\varepsilon)}\right) \\
&\leq \exp\left(-\frac{c\,\varepsilon^2(1-\varepsilon)^2}{4(1+\varepsilon)}\log n\right) .
\end{aligned}$$

The first inequality follows from the Chernoff bound; the second inequality holds since $z > 0$ and $0 < \varepsilon < \delta$. The third inequality is true since $\ln(1+x) > x - \frac{x^2}{2}$ for $0 < x < 1$, and the last one follows from the assumption that $\alpha = c\left(\frac{Z\log n}{DT} + \frac{D}{T}\right)$. In summary, the probability that the algorithm chooses D non-distinguishing queries $j \in J$ for which $M_j^{S_j} < \alpha n(1-\varepsilon)^2$ is at most $2n^{-\Omega(1)}$, and hence $|S| - |S'| \geq \alpha n(1-\varepsilon)^2$ with probability at least $1 - 2n^{-\Omega(1)}$. □

Using Lemmas 1 and 3 we now present the proof of Theorem 3.

Proof of Theorem 3 The claim that users of the same type are in the same subset follows directly from Lemma 1. To show separation, consider a type A, and let B be a type of popularity α, such that the vectors of A and B differ in at least D queries. If a query distinguishing A from B is selected at Step 6 of `Proc_Set` by time T, the types are separated and we are done. Otherwise, suppose for contradiction that users of type A and B are not separated by time T, and let $S(t)$ denote their common subset at any time $t \leq T$. By Lemma 3, with high probability we have for $t_1 = T/2 + 2D$, that $|S(t_1)| \leq |S(T/2)| - (1-\varepsilon^2)\alpha n$. Similarly, applying Lemma 3 inductively,

Algorithm 3 `Sep_Any`

1: For each $0 \le l \le \lceil \log\log n \rceil$, let I_l be a randomly selected subset of the queries of size $|I_l| = m2^{-l}$.
2: **for all** $0 \le l \le \lceil \log\log n \rceil$ **do**
3: Run an independent instance of `Sep_Close` on I_l. For $l > 0$ instance l is scheduled $\frac{2^{l/2}}{2\sum_{i=1}^{\log\log n} 2^{i/2}}$ fraction of the rounds; instance 0 is scheduled half of the rounds.
4: Let $S^{p,l} \in \mathscr{S}^l$ be the subset that includes user p by instance l.
5: **end for**
6: The global output by user p is $S^p = \bigcap_{l \in [0,\log\log n]} S^{p,l}$.

we have (w.h.p.) that for $k = 1,2,\ldots \lfloor \frac{T}{4D} \rfloor$, it holds that at time $t_k = T/2 + 2kD$, $|S(t_k)| \le |S(T/2)| - k(1-\varepsilon^2)\alpha n$. Therefore, at time $T \gg T/2 + 2D \cdot 1/\alpha$, we have that $|S(T)| < 0$, contradiction. □

3.2 Algorithm `Sep_Any`

The query complexity of Algorithm `Sep_Close` is linearly dependent on the separation distance d, and thus it is useful only for small values of d. In this section we extend Algorithm `Sep_Close` to an algorithm which is useful for larger separation distances as well.

For simplicity of presentation and to reduce the number of parameters, we present the results in this section for the case $m = \Theta(n)$, i.e., the number of queries is roughly equal to the number of users. Extension to the general case is straightforward: If $n = \Theta(m)$, the algorithm works essentially as presented. If $n \gg m$, add $(n-m)$ "dummy queries" on which all users answer '0', say; and if $n \ll m$, we can replicate each user m/n times, blowing up the number of queries each user has to answer by a factor of m/n.

The idea in Algorithm `Sep_Any` is as follows (see pseudocode in Algorithm 3). If the separation distance d was known to satisfy $\log n \le d \le \log^2 n$, then we could apply Algorithm `Sep_Close` on a random subset of $m\frac{\log n}{d}$ queries. It is not hard to see that after $\Omega\left(\frac{\log n}{\alpha\sqrt{d}}\right)$ rounds, we get a separation between α-types at distance d from each other, because with high probability, $\Theta(\log n)$ distinguishing queries are included in the randomly chosen queries, and separation follows from Corollary 1. If $d > \log^2 n$, then we can use the version that works for $d = \log^2 n$, because, as we show later, at $d = \log^2 n$ we already hit the lower bound. Since d is unknown, we apply algorithm `Sep_Close` $\log\log n$ times: At instance l for $l \in [1, \log\log n]$, a random subset of $|I_l| \stackrel{\text{def}}{=} m2^{-l}$ queries is used. We run these $\log\log n$ instances in parallel, with different speeds of execution: instance l runs at a speed faster by a factor of $\sqrt{2}$ form the speed of instance $l-1$. An additional instance, which looks at all queries, is used for separation distance smaller than $\log n$.

In Theorem 1 we summarize the performance of Algorithm `Sep_Any`. For convenience, we reproduce the theorem below. As explained above, we prove it for the case $m = \Theta(n)$.

Theorem 1 *Let n be the number of users, and let m be the number of queries. Denote $Z = m/n$. Then after $O(T)$ steps, with probability at least $1 - n^{-\Omega(1)}$, Algorithm*

Sep_Any of Section 3 finds a $\frac{d}{m}$-separating partition for all users whose type popularity is at least $\alpha(d)$, where

$$\alpha(d) = \begin{cases} \frac{[Z]}{T} \cdot \frac{\log n}{d} & 1 \leq d < \sqrt{\log n} \\ \frac{[Z]}{T} \cdot \sqrt{\log n} & \sqrt{\log n} \leq d < (\log n)^{3/2} \\ \frac{[Z]}{T} \cdot \frac{(\log n)^2}{d} & (\log n)^{3/2} \leq d < (\log n)^2 \\ \frac{[Z]}{T} & (\log n)^2 \leq d \end{cases}$$

Proof of Theorem 1 Consider Algorithm Sep_Any at some time T. Let p and q be any two users, and let

$$\alpha = \Omega\left(\frac{1}{T}\min\left\{\frac{\log n}{d} + \sqrt{\log n}\,,\ \frac{\log^2 n}{d} + 1\right\}\right).$$

For any $l \in [0, \log\log n]$, let T_l be the number of steps taken by instance l, and let $S^{p,l}$ and $S^{q,l}$ be the subsets to which users p and q, respectively, belong at instance l. Let the output of the algorithm for users p and q be S^p and S^q respectively.

If users p and q belong to the same type, then $S^{p,l} = S^{q,l}$ for every l by Theorem 3 and hence $S^p = S^q$ at all times. So suppose p and q belong to two distinct types of popularity α each. We proceed by case analysis, according to the distance d between the types of p and q.

We proceed by case analysis. Consider first the case where $\log n \leq d \leq \log^2 n$. Let $l^* = \left\lceil \log\left(\frac{d}{\log n}\right)\right\rceil$. Then $|I_{l^*}| = m2^{-l^*} = \frac{m\log n}{d}$. Let m_{l^*} be the number of queries in this subset on which p and q answer differently. By the random choice of I_{l^*}, m_{l^*} is a hypergeometric random variable with $m\frac{\log n}{d}$ selections from population of m queries, of which d distinguish p from q. Therefore (see [9]), we can bound the tail probability of m_{l^*} using bounds for a binomial random variable with $m\frac{\log n}{d}$ trials and success probability $\frac{d}{m}$. It follows that with probability $1 - n^{\Omega(1)}$, the subset I_{l^*} includes $\Theta(\log n)$ queries on which p and q answer differently. At time T, instance I_{l^*} has executed $T_{l^*} = T\frac{2^{l^*/2}}{2\sum_{i=1}^{i=\log\log n} 2^{i/2}} = \Theta\left(\frac{T\sqrt{d}}{\log n}\right)$ steps. As $\alpha T_{l^*} = \Omega\left(\frac{\log n}{\sqrt{d}}\right)$, it follows from Corollary 1 that $p \notin S^{q,l^*}$ with high probability, and hence $p \notin S^q$.

Consider now the case $d > \log^2 n$. Focus on iteration $\lceil \log\log n \rceil$. Similarly to the previous case, $I_{\lceil \log\log n \rceil}$ includes at least $\Theta(\log n)$ queries on which p and q answer differently, and $\alpha T_{\lceil \log\log n \rceil} \geq \Omega\left(1 + \frac{\log n}{\sqrt{d}}\right) = \Omega(1)$. By Corollary 1, $p \notin S^{q,\lceil \log\log n \rceil}$ with high probability, and hence $p \notin S^q$.

If $d < \log n$, then at time T, instance 0 has executed $T_0 = T/2$ steps and hence $\alpha T_0 = \Omega\left(\frac{\log n}{d} + \sqrt{\log n}\right)$. Therefore by Theorem 3, for $d' = \min\{d, \sqrt{\log n}\}$, we have that $p \notin S^{q,0}$ w.h.p., implying $p \notin S^q$. □

4 A Lower Bound

In this section we prove Theorem 2, namely a lower bound on the query complexity of the user classification problem. For convenience, we reproduce the statement below. The proof of Theorem 2 follows directly from Lemmas 4, 5 and Lemma 6.

Theorem 2 *Let* $\frac{\log n}{n} \leq \alpha \leq 1/2$ *and* $\delta \geq 3^{-\alpha n/2}$. *Any algorithm which separates, with probability at least* $1-\delta$, *users whose preference vectors differ in d coordinates for* $d < \alpha n$, *and whose type popularity is* α, *has query complexity* $\Omega\left(\frac{1}{\alpha}\left\lceil\frac{\log 1/\delta}{d}\right\rceil\right)$.

To prove the theorem, we use the following variant of Yao's minimax principle [17] which is useful for anytime algorithms. (Yao's principle is usually stated for running time: we apply it to success probability.)

Lemma 4 *Consider a problem with input domain* $\mathscr{D}$ *and let* $T > 0$. *If there exists a constant* δ, *and probability distribution p over* $\mathscr{D}$, *such that for any T-time deterministic algorithm A with input* $I \in \mathscr{D}$ *drawn randomly according to p it holds that* $\Pr_p[A \text{ solves } I] < 1-\delta$, *then for any T-time randomized algorithm R there exists an instance* $I_R \in \mathscr{D}$ *with* $\Pr[R \text{ solves } I_R] < 1-\delta$, *where the latter probability is over the coin tosses of R.*

Proof Define an indicator random variable χ_A which takes the value 1 if algorithm A succeeds to solve the problem and 0 otherwise. Clearly, $E_p[\chi_A] = \Pr_p[A \text{ solves } I]$. Given the results of coin tosses, a T-time randomized algorithm is a T-time deterministic algorithm, and therefore, denoting by q the distribution over coin tosses, we have that

$$\begin{aligned}
E[\chi_R] &= \sum_{I\in\mathscr{D}} \Pr_p[I] E_q[\chi_R \mid \text{instance } I] \\
&= \sum_{I\in\mathscr{D}} \Pr_p[I] \sum_{\text{tosses } \tau} \Pr_q[\tau] E[\chi_R \mid \text{instance } I, \text{tosses } \tau] \\
&= \sum_{\text{tosses } \tau} \Pr_q[\tau] \sum_{I\in\mathscr{D}} \Pr_p[I] E[\chi_R \mid \text{instance } I, \text{tosses } \tau] \\
&< \sum_{\text{tosses } \tau} \Pr_q[\tau](1-\delta) \\
&= 1-\delta\ ,
\end{aligned}$$

and hence for each T-time randomized algorithm R there exists an instance $I_R \in \mathscr{D}$ such that $\Pr_q[R \text{ solves } I_R] < 1-\delta$. □

As in Section 3.2, we assume for convenience that n is both the number of users and queries. First we show the bound holds for cases where d is small compared to $\log 1/\delta$.

Lemma 5 *Let* α, δ *be as in Theorem 2 and suppose that* $d < (\log_3 1/\delta)/2$. *Let A be a deterministic algorithm for the classification problem with query complexity* $T < \frac{\log_3 1/\delta}{8\alpha d}$. *There exists a distribution over the instances of A such that the probability that A separates two* α*-types at distance d is less than* $1-\delta$.

Proof Assume for simplicity that $1/\alpha$ is an integer. In the instances we construct there is a query set that consists of three disjoint subsets denoted X, Y and W, where $|X| = d$, $|Y| = 1/\alpha - 2$ and $|W| = n - |X| - |Y|$. The users are partitioned in $1/\alpha$ types with popularity α each. The types, denoted as $C_1, C_2, C_3, \ldots, C_{1/\alpha}$, are defined by the following preference vectors (see Figure 2).

$$
\begin{array}{lcc}
 & & \overbrace{}^{d}\ \overbrace{}^{\frac{1}{\alpha}-2}\ \overbrace{}^{n-d-\frac{1}{\alpha}+2} \\
C_1 & & 1\cdots1\ 00\cdots00\ 0\cdots0 \\
C_2 & & 0\cdots0\ 00\cdots00\ 0\cdots0 \\
C_3 & = & 0\cdots0\ 10\cdots00\ 0\cdots0 \\
 & & \qquad 1 \\
\vdots & & \qquad\quad \ddots \\
 & & \qquad\qquad 1 \\
C_{1/\alpha} & & \underbrace{0\cdots0}_{X}\ \underbrace{00\cdots01}_{Y}\ \underbrace{0\cdots0}_{W}
\end{array}
$$

Fig. 2 Typical example of preference vector for Lemma 5

- Users of type C_1 answer '1' to queries in X and '0' to all other queries.
- Users of type C_2 answer '0' to all queries.
- Let $Y = \{y_3, y_4, \ldots, y_{1/\alpha}\}$. For $i > 2$, users of type C_i answer '1' to y_i and '0' to all other queries.

Finally, the probability distribution over the instances is defined by randomly permuting the identities of users and of queries.

In order to bound the probability that an arbitrary deterministic algorithm A separates users of type C_1 from users of type C_2, we may assume that A knows in advance the answer of every user to all queries in Y. Clearly, the probability of success does not decrease due to this extra knowledge. As all answers of users in the types $\{C_3, \ldots, C_{1/\alpha}\}$ are defined by their answers to queries in Y, we henceforth focus on answers by users of types C_1 and C_2 only, and ignore all other users and their answers.

Consider the prefixes of executions of algorithm A in which users from types C_1 answer only '0'. Clearly, so long as these answers are only '0', the algorithm cannot separate users of type C_1 from users of type C_2 due to the random permutation of the users. Moreover, since all queries are answered in the same way (namely, '0'), any deterministic algorithm can be represented, during that prefix, by a fixed list of user-query index pairs: since the answers are fixed, they have no effect on the unfolding of the algorithm.

We now bound the probability that the algorithm gets only '0' answers, fix any users permutation and assume that the queries are ordered by a given permutation σ (the conditional probability is over the random permutation of the *queries*).

Let $X(\sigma)$ be the d queries distinguishing C_1 from C_2 (namely, $X(\sigma)$ is the set of queries in X under σ). Let w_q be the number of answers to query q, and define $w_{X(\sigma)} = \sum_{q \in X(\sigma)} w_q$. The probability that a fixed list of user-query pairs gets '0' answers by users of C_1 and C_2 for all queries is at least

$$\Pr[\text{all answers are } 0] \geq \prod_{k=0}^{w_{X(\sigma)}-1} \frac{\alpha n - k}{2\alpha n - k}\ , \tag{2}$$

because the kth user has probability at least $\frac{\alpha n-k}{2\alpha n-k}$ of being of type C_2 and answer '0' at the query, and there are $w_{X(\sigma)}$ responses to these queries.

Let $J = \left\{ j \in X \cup W \mid w_j \leq \frac{\alpha n}{2d} \right\}$. Intuitively, J is the set of queries (without Y) that received "not too many" answers. Note that J is large: at time T, the total number of answers is at most $2\alpha nT$, and the number of queries for which $w_i > \frac{\alpha n}{2d}$ is at most $4dT < \frac{\log_3 1/\delta}{2\alpha} \leq n/4$ by assumption on T. It therefore follows that

$$|J| \geq |X \cup W| - 4dT \geq n/2 \,. \tag{3}$$

Now, if $X(\sigma) \subseteq J$, then $w_{X(\sigma)} \leq \frac{\alpha n}{2}$, so in that case we have, by Eq. (2) and Eq. (3)

$$\begin{aligned} \Pr[\text{all answers are } 0 \mid X(\sigma) \subseteq J] &\geq \prod_{k=0}^{w_{X(\sigma)}-1} \frac{\alpha n - k}{2\alpha n - k} \\ &\geq \left(1 - \frac{\alpha n}{2\alpha n - w_{X(\sigma)}}\right)^{w_{X(\sigma)}} \\ &\geq 3^{-w_{X(\sigma)}} \,. \end{aligned} \tag{4}$$

Considering the case of $X(\sigma) \subseteq J$, since the total number of query permutations is $\binom{n-1/\alpha+2}{d} \leq \binom{n}{d}$, we may conclude from Eq. (4) that

$$\begin{aligned} \Pr[\text{all answers are } 0] &= \sum_{\sigma} \left(\Pr[\text{permutation } \sigma \text{ is chosen}] \sum_{\sigma} \cdot \Pr[\text{all answers are } 0 \mid \sigma] \right) \\ &\geq \frac{1}{\binom{n}{d}} \sum_{\sigma : X(\sigma) \subseteq J} \Pr[\text{all answers are } 0 \mid X(\sigma) \subseteq J] \\ &\geq \frac{1}{\binom{n}{d}} \sum_{\sigma : X(\sigma) \subseteq J} 3^{-w_{X(\sigma)}} \,. \end{aligned} \tag{5}$$

Eq. (5) means that in order to prove an upper bound on the probability of success of the algorithm, it is sufficient to prove a lower bound on $\sum_{\sigma : X(\sigma) \subseteq J} 3^{-w_{X(\sigma)}}$. This can be done using Lagrange multipliers as follows. Assume, without loss of generality, that $J = \{1, 2, \ldots, |J|\}$, and define $\varphi(w_1, \ldots, w_{|J|}) \stackrel{\text{def}}{=} \sum_{\sigma : X \subseteq J} 3^{-w_{X(\sigma)}}$. We seek a lower bound on φ subject to the constraint that $\sum_{j \in J} w_j \leq 2\alpha nT$. Since φ is monotone decreasing in each of its variables, the minimum is obtained on the boundary, where $\sum_{j \in J} w_j = 2\alpha nT$. In fact, the minimum is obtained when $w_j = 2\alpha nT/|J|$ for all $j \in J$, and therefore

$$\begin{aligned} \Pr[\text{all answers are } 0] &\geq \frac{1}{\binom{n}{d}} \sum_{\sigma : X(\sigma) \subseteq J} 3^{-w_{X(\sigma)}} \\ &\geq \frac{\binom{n/2}{d}}{\binom{n}{d}} 3^{-2d\frac{\alpha nT}{|J|}} \\ &> \left(\frac{n/2 - d}{n - d}\right)^d 3^{-\log_3(1/\delta)/2} \\ &\geq 3^{-\log_3(1/\delta)/2 - d} \\ &> \delta \,. \end{aligned}$$

The first inequality follows from Eq. (5); the second inequality follows from the Lagrange optimization described above and Eq. (3). The remaining inequalities follow from the assumptions on T, d and δ together with Eq. (3). □

Next we show that the bound holds when d is large compared to $\log 1/\delta$. The bound is proved by an instance where the Hamming distance between any two types is greater than d, and showing that a user answering less than $1/\alpha$ queries cannot be correctly associated with its type with probability greater than half.

Lemma 6 *Let $1/n \leq \alpha \leq 1/2$, let $d < \alpha n$, and let A be an algorithm which computes, with probability at least $1/2$, a d/n-separating partition for users whose type popularity is at least α. Then A has query complexity $\Omega(1/\alpha)$.*

Proof The proof uses a simple symmetry argument. Consider a case where users are partitioned to $1/\alpha$ symmetric types, where users answer '0' to all queries, except that each type has a distinct set of d queries to which it answers '1'. Since the Hamming distance between any two types is $2d$ any d/n-separating algorithm in fact identifies the user types precisely. Consider algorithms that actually know all $1/\alpha$ preference vectors of the types. Suppose that a user of a random type arrives. Clearly, the probability that any deterministic algorithm will identify that user's type correctly in less than $1/2\alpha$ queries is less than $1/2$, and therefore the result follows from Lemma 4. □

It is interesting to note that Lemma 5 argues about the total work done until the first user from the types in question hits a distinguishing query, while Lemma 6 argues about the number of queries the *last user* has to answer until his type is determined.

References

1. Alon N, Awerbuch B, Azar Y, Patt-Shamir B (2006) Tell me who I am: an interactive recommendation system. In: Proc. 18th Ann. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp 1–10
2. Awerbuch B, Azar Y, Lotker Z, Patt-Shamir B, Tuttle M (2005) Collaborate with strangers to find own preferences. In: Proc. 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp 263–269
3. Awerbuch B, Patt-Shamir B, Peleg D, Tuttle M (2005) Improved recommendation systems. In: Proc. 16th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA), pp 1174–1183
4. Azar Y, Fiat A, Karlin A, McSherry F, Saia J (2001) Spectral analysis of data. In: Proc. 33rd ACM Symp. on Theory of Computing (STOC), pp 619–626
5. Drineas P, Kerenidis I, Raghavan P (2002) Competitive recommendation systems. In: Proc. 34th ACM Symp. on Theory of Computing (STOC), pp 82–90
6. Facebook (2004) `www.facebook.com`
7. Freeman LC (2004) The Development of Social Network Analysis: A Study in the Sociology of Science. Empirical Press
8. Goldberg K, Roeder T, Gupta D, , Perkins C (2001) Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval Journal 4(2):133–151

9. Hoeffding W (1963) Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58(301):13–30
10. Kleinberg J, Sandler M (2003) Convergent algorithms for collaborative filtering. In: Proc. 4th ACM Conf. on Electronic Commerce (EC), pp 1–10
11. Kumar R, Raghavan P, Rajagopalan S, Tomkins A (1998) Recommendation systems: A probabilistic analysis. In: Proc. 39th IEEE Symp. on Foundations of Computer Science (FOCS), pp 664–673, URL citeseer.nj.nec.com/kumar98recommendation.html
12. MySpace (2003) `www.myspace.com`
13. Papadimitriou CH, Raghavan P, Tamaki H, Vempala S (1998) Latent semantic indexing: A probabilistic analysis. In: Proc. 17th ACM Symp. on Principles of Database Systems (PODS), ACM Press, pp 159–168
14. Rashid AM, Albert I, Cosley D, Lam SK, McNee SM, Konstan JA, Riedl J (2002) Getting to know you: learning new user preferences in recommender systems. In: IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces, ACM, New York, NY, USA, pp 127–134, DOI http://doi.acm.org/10.1145/502716.502737
15. Sarwar B, Karypis G, Konstan J, Riedl J (2000) Analysis of recommendation algorithms for e-commerce. In: Proc. 2nd ACM Conf. on Electronic Commerce (EC), ACM Press, pp 158–167, DOI http://doi.acm.org/10.1145/352871.352887
16. Wasserman S, Faust K (1994) Social Network Analysis: Methods and Applications. Cambridge University Press
17. Yao AC (1977) Probabilistic computations: toward a unified measure of complexity. In: Proc. 17th IEEE Symp. on Foundations of Computer Science (FOCS), pp 222–227