2011 IEEE Symposium on Security and Privacy

"You Might Also Like:" Privacy Risks of Collaborative Filtering

Joseph A. Calandrino[1], Ann Kilzer[2], Arvind Narayanan[3], Edward W. Felten[1], and Vitaly Shmatikov[2]

[1]Dept. of Computer Science, Princeton University {jcalandr,felten}@cs.princeton.edu
[2]Dept. of Computer Science, The University of Texas at Austin {akilzer,shmat}@cs.utexas.edu
[3]Dept. of Computer Science, Stanford University arvindn@cs.utexas.edu

***Abstract*—Many commercial websites use recommender systems to help customers locate products and content. Modern recommenders are based on collaborative filtering: they use patterns learned from users' behavior to make recommendations, usually in the form of related-items lists. The scale and complexity of these systems, along with the fact that their outputs reveal only relationships between items (as opposed to information about users), may suggest that they pose no meaningful privacy risk.**

In this paper, we develop algorithms which take a moderate amount of auxiliary information about a customer and infer this customer's transactions from temporal changes in the public outputs of a recommender system. Our inference attacks are passive and can be carried out by any Internet user. We evaluate their feasibility using public data from popular websites Hunch, Last.fm, LibraryThing, and Amazon.

I. Introduction

Recommender systems are ubiquitous on the Web. When you buy products from Amazon, rent movies on Netflix, listen to music on Last.fm, or perform myriad other tasks online, recommender systems make suggestions based on your behavior. They typically rely on *collaborative filtering*, or patterns learned from other users: for example, "customers who buy item X (as you just did) often buy item Y."

We investigate the privacy risks of recommender systems based on collaborative filtering. By design, such systems do not directly reveal behavior of individual users or any "personally identifiable information." Their recommendations are based on aggregated data involving thousands to millions of users, each with dozens to thousands of transactions. Moreover, modern collaborative filtering leverages relationships between items rather than relationships between users, creating an extra level of indirection between public recommendations and individual transactions. One might therefore assume that it is infeasible to draw meaningful inferences about transactions of specific users from the public outputs of recommender systems. We show that this assumption is wrong.

Our contributions. We develop a set of practical algorithms that allow accurate inference of (partial) individual behavior from the aggregate outputs of a typical recommender system. We focus on item-to-item collaborative filtering, in which the system recommends items similar to a given item. Our key insight is to exploit the *dynamics of public recommendations* in order to make the leap from aggregate to individual data. This paper is the first to make and quantitatively evaluate the observation that temporal changes in aggregate recommendations enable accurate inference of individual inputs.

Our algorithms require only passive, "black-box" access to the public outputs of a recommender system, as available to any Internet user. The attacker need not create fake customers or enter purchases or ratings into the system. We do not assume that customers' transactions are available in either identifiable or anonymized form. Our approach is thus fundamentally different from the techniques for re-identifying anonymized transactional records [26]. Re-identification assumes that the attacker has direct access to customers' records. By contrast, our attacks rely only on indirect access: the records are fed into a complex collaborative filtering algorithm and the attacker's view is limited to the resulting outputs.

Our algorithms monitor changes in the public outputs of recommender systems—item similarity lists or cross-item correlations—over a period of time. This dynamic information is then combined with a moderate amount of *auxiliary information* about some of the transactions of a particular "target" user. The combination is used to infer many of the target user's unknown transactions with high accuracy. Auxiliary information can be obtained by analyzing the user's publicly revealed behavior; we discuss this in more detail in Section III.

Overview of results. We evaluate our algorithms on real-world recommender systems which produce different types of recommendations. Our goal is not to claim privacy flaws in these specific sites—in fact, we often use data voluntarily disclosed by their users to verify our inferences—but to demonstrate the general feasibility of inferring individual transactions from the outputs of collaborative filtering systems.

Some recommender systems make item-to-item correlations available. An example is Hunch, a popular recommendation and personalization website. There is a tradeoff between the number of inferences and their accuracy. When optimized for accuracy, our algorithm infers a third of the test users' secret answers to Hunch questions with no error.

Other recommender systems make only item similarity or "related items" lists available, with or without numeric similarity scores. Examples include Last.fm, an online music service, and LibraryThing, an online book cataloging service and recommendation engine. The results from our LibraryThing experiment illustrate the yield-accuracy tradeoff, ranging from

1081-6011/11 $26.00 © 2011 IEEE
DOI 10.1109/SP.2011.40

58 inferences per user with 50% accuracy to 6 inferences per user with 90% accuracy. Another example of item similarity lists is the "Customers who bought this item also bought ..." feature on Amazon. Our ability to evaluate our algorithms on Amazon's recommender system is constrained by the lack of a "ground-truth oracle" for verifying our inferences, but we conducted a limited experiment to demonstrate the feasibility of adversarial inference against Amazon's recommendations.

By necessity, our experiments on real-world systems involve only a limited sample of users. To demonstrate that our inference algorithms also work at scale, we implemented an item-to-item collaborative filtering engine very similar to that used by Amazon, and ran it on the Netflix Prize dataset of movie-rating histories [28]. This allowed us to simulate a complete system, producing public recommendations as well as auxiliary information about users. The underlying dataset of individual ratings served as the "ground-truth oracle" for verifying inferences made by our algorithm. Our algorithm was able to infer 4.5% of transactions of sufficiently active users with an accuracy of 90%.

There is a passing similarity between our inference algorithms and actual collaborative filtering. Both use statistical methods to reach probabilistic conclusions about unknown aspects of users' behavior. Our algorithms, however, are technically different and pursue a fundamentally different goal: not to predict future events, but to infer past events. This translates into several concrete differences, discussed in Section V. For example, in contrast to prediction algorithms, ours perform best when a user *deviates* from normal patterns and if his transactions involve less popular items. We can also infer an approximate date when a transaction occurred.

For completeness with respect to different types of recommender systems, we present a simple active attack on user-based collaborative filtering. In broad terms, the attacker creates multiple sybil users whose transactional profile is similar to what he knows about the target user's profile and infers the target's non-public transactions from the recommendations made by the system to these sybils.

In summary, this work is the first to develop a generic method for inferring information about individual users' transactions from the aggregate outputs of collaborative filtering. We show that public outputs of common recommender algorithms may expose non-public details of individual users' behavior—products they purchase, news stories and books they read, music they listen to, videos they watch, and other choices they make—without their knowledge or consent.

II. Survey of recommender systems

Recommender systems have become a vital tool for attracting and keeping users on commercial websites. Their utility is supported by research [14] as well as common practice.

The task of a recommender system can be abstractly described as follows. Consider a matrix in which rows correspond to users and columns correspond to items. Each value in this matrix represents a user's revealed or stated preference (if any) for an item: for example, whether he purchased a book, how many times he listened to a song, or what rating he gave to a movie. Because the item set is typically far larger than a single user can consume and evaluate, this matrix is "sparse:" only a small fraction of entries are filled in. A recommender system takes this matrix as input, along with any available metadata about users (such as demographics) and items (such as item categories). The goal of the system is to extrapolate users' "true" preferences over the full item set.

Recommender systems can provide several types of recommendations. If the system suggests items to an individual user based on its knowledge of the user's behavior, it provides *user-to-item* recommendations. If the system helps users find similar users, it provides *user-to-user* recommendations. If, given an item, the system suggests similar items, it provides *item-to-item* recommendations. The system may even list users who are strongly associated with a given item, thus providing *item-to-user* recommendations. The same system may provide several types of recommendations: for example, Last.fm provides both item-to-item and user-to-user recommendations.

We focus on item-to-item recommendations, both because they are supported by essentially all popular online recommender systems and because their output is typically public and thus the most feasible avenue for an attack.

A thorough technical survey of the literature on recommender systems can be found in [1]. Recommender systems can be classified as content-based, collaborative, and hybrid. Content-based systems identify relationships between items based on metadata alone and recommend items which are similar to the user's past transactions. Purely content-based recommender systems pose no privacy risks under our attacks, since the system does not consider other users' transactions when making recommendations to a user.

Collaborative filtering is much more robust and domain-agnostic, and hence far more popular. Collaborative filtering identifies relationships between items based on the preferences of all users. Traditional collaborative filtering methods are *user-based*. For a given user, the system finds other users with a similar transaction history. In the user-to-user case, the system recommends these similar users; in the user-to-item case, it recommends items selected by the similar users.

The alternative is *item-based* collaborative filtering, which was first described by Sarwar et al. [31] and has become the dominant approach [2, 20, 21]. It generates recommendations using *item similarity scores* for pairs of items, which are based on the likelihood of the pair being purchased by the same customer. Although some systems make raw similarity scores public, their main uses are internal: for example, to find items which are similar to a user's previously purchased items in order to make user-to-item recommendations.

A. Item-to-item recommendations

It has become standard practice for online recommender systems to publish item-to-item recommendations, usually in the form of **item similarity lists** produced from item similarity scores. Given an item, these lists help find related items (see [6] for a survey of algorithms). On Amazon, this is seen as

the "Customers who bought this item also bought . . . " feature. Similar features are found on many commercial websites, including iTunes, Last.fm, Pandora, Netflix, YouTube, Hulu, and Google Reader. Item similarity lists even appear on many sites that do not have traditional user-to-item recommendations, such as IMDb, CNN, and the New York Times.[1] Item similarity lists may be limited to top N items or contain an ordered list of hundreds or thousands of items.

Many systems reveal additional information. Amazon reveals not only the **relative popularity** of items via bestseller lists and "sales rank," but also the percentage of users purchasing certain other items after viewing the given item.[2] For every song, Last.fm provides the number of listeners and how many times it was played by each listener. Given a book, LibraryThing provides several ordered lists of related books, including more common and more obscure recommendations; some lists also contain detailed transaction information, such as the precise number of users who have both books. Finally, Hunch gives all users access to the entire **item-to-item covariance matrix** via an API.

B. User-to-item recommendations

User-to-item recommendations may be user-based (finding similar users and recommending their items) or item-based (finding items related to ones that the user chose in the past). Amazon provides several personalized lists with up to 1,000 items to logged-in users. LibraryThing, Last.fm, and Netflix also provide recommendation lists to their users.

III. Attack model

We view the data that the system uses to make recommendations as a matrix where rows correspond to users and columns to items. Each cell represents a *transaction* (*e.g.*, the user's purchase or stated preference for an item). Entries may be dated; the date may or may not be sensitive from a privacy perspective. As users interact with the system, the matrix is continually updated and new recommendations are generated.

Our primary focus is on passive inference attacks. The attacker has access to the public outputs of the recommender system, which, depending on the system, may include item similarity lists, item-to-item covariances, and/or relative popularity of items (see Section II). The outputs available to the attacker are available to any user of the system. Crucially, the attacker observes the system over a certain period and can thus capture *changes* in its outputs: an increase in covariance between certain items, appearance of an item on the similarity list of another item, an increase in an item's sales rank, *etc.* Note, however, that each update incorporates the effects of hundreds or thousands of transactions. With the exception of auxiliary information (described below), inputs into our inference algorithms are based on aggregate statistics and contain neither personally identifiable information nor information about specific transactions.

For completeness, we also briefly consider active attacks, where the attacker creates fake, "sybil" users and manipulates their entries in the corresponding rows of the transaction matrix. Depending on the system, this includes adding new entries (easy in the case of ratings and stated preferences, more expensive for purchases), modifying existing entries, or deleting them (easy in the case of ratings and preferences and may also be possible for purchases; for example, one can instruct Amazon to ignore certain purchases when making recommendations). Observable outputs include items recommended by the system to the sybil users and, in the case of systems like Last.fm or LibraryThing, also user similarity lists which explicitly identify users with similar histories.

Auxiliary information. We assume that for some users, a subset of their transaction history is available to the attacker. We refer to this as the attacker's *auxiliary information*. An inference attack is successful if it enables the attacker to learn transactions which are *not* part of the auxiliary information. In other words, the attacker's objective is to "leverage" his prior knowledge of some of the target user's transactions to discover transactions that he did not know about.

There are many sources of auxiliary information. The first is the target system itself. On many websites, users publicly rate or comment on items, revealing a high likelihood of having purchased them. The system may even publicly confirm the purchase, *e.g.*, "verified purchase" on Amazon. Alternatively, on sites with granular privacy controls, some of the transactions may be publicly visible, while others remain private.

The second source is users revealing partial information about themselves via third-party sites. This is increasingly common: for example, music websites allow embedding of tracks or playlists on blogs or other sites, while Amazon Kindle allows "tweeting" a selected block of text; the identity of the book is automatically shared via the tweet.[3]

The third source is data from other sites which are not directly tied to the user's transactions on the target site, but leak partial information about them. For example, books listed in a Facebook user profile reveal partial information about purchases on Amazon. Linking users across different sites is a well-studied problem [17, 27]. On blippy.com, "a website where people obsessively review everything they buy," individual purchase histories can be looked up by username, making linkages to other sites trivial. Note that the third (and to a lesser extent, the second) source of auxiliary information is outside the control of the recommender system.

Furthermore, information about users' behavior is constantly leaked through public mentions on online fora, real-world interactions with friends, coworkers, and acquaintances, *etc.* Therefore, we do not consider the availability of auxiliary information to be a significant impediment to our attacks.

[1] Even offline retailers such as supermarkets frequently deploy item-to-item similarity analysis to optimize store layout [3].

[2] We do not exploit the latter information for the inference attacks in this paper. This is an interesting topic for future research.

[3] The stream of such tweets can be conveniently accessed in real time by searching Twitter for "amzn.com/k/".

Algorithm 1: RELATEDITEMSLISTINFERENCE

Input: Set of target items $\mathcal{T}$, set of auxiliary items $\mathcal{A}$, scoring function : $\mathbb{R}^{|\mathcal{A}|} \rightarrow \mathbb{R}$
Output: Subset of items from $\mathcal{T}$ which are believed by the attacker to have been added to the user's record

$inferredItems = \{\}$
foreach *observation time* τ **do**
  Δ = observation period beginning at τ
  N_Δ = delta matrix containing changes in positions of items from $\mathcal{T}$ in lists associated with items from $\mathcal{A}$
  foreach *target item* t *in* N_Δ **do**
    $scores_t = \text{SCOREFUNCTION}(N_\Delta[t])$
    if $scores_t \geq threshold$ *and* $t \notin \mathcal{A}$ **then**
      $inferredItems = inferredItems \cup \{t\}$
return $inferredItems$

IV. GENERIC INFERENCE ATTACKS

A. Inference attack on related-items lists

In this setting of the problem, the recommender system outputs, for each item, one or more lists of related items. For example, for each book, LibraryThing publishes a list of popular related books and a list of obscure related books.

The description of the inference algorithm in this section is deliberately simplified with many details omitted for clarity. Intuitively, the attacker monitors the similarity list(s) associated with each auxiliary item (*i.e.*, item that he knows to be associated with the target user). The attacker looks for items which either appear in the list or move up, indicating increased "similarity" with the auxiliary item. If the same target item t appears and/or moves up in the related-items lists of a sufficiently large subset of the auxiliary items, the attacker infers that t has been added to the target user's record.

Algorithm 1 shows the inference procedure. Intuitively, delta matrices N_Δ store information about the movement of each target item t in the related-items lists of auxiliary items $\mathcal{A}$ (we defer the discussion of matrix construction). The attacker computes a score for each t using a scoring function. The simplest scoring function counts the number of auxiliary items in whose related-items lists t has appeared or risen. If the final score exceeds a predefined threshold, the attacker concludes that t has been added to the user's record.

Scoring can be significantly more complex, taking into account full dynamics of item movement on related-items lists or giving greater weight to certain lists. To reduce false positives and improve inference accuracy, the scoring function must be fine-tuned for each system. For example, recommender systems tend to naturally cluster items. Netflix users who watched the DVD of the first season of "The Office" also tend to watch the second season. Suppose that some movie rises in the similarity lists of both seasons' DVDs. Because the overlap in viewers across seasons is so great, this does not reveal much more information than a movie rising in the list associated with a single DVD. In fact, it may reveal *less* if users who watch only one of the two seasons are very unusual.

Our scoring functions prefer sets of auxiliary items which span genres or contain obscure items. Consider LibraryThing, where users share the books they read. Classics such as *To Kill a Mockingbird* or *Catcher in the Rye* are so common that changes in their similarity lists tend to result from widespread trends, not actions of a single user. Movement of a book in a list associated with an obscure book reveals more than movement in a list associated with a bestseller.

B. Inference attack on the covariance matrix

In this setting of the problem, the item-to-item covariance matrix is visible to any user of the system. An example of an online recommender system that reveals the covariance matrix is Hunch (see Section VI-A). We also explain complications, such as asynchronous updates to the system's public outputs, which apply to the related-items scenario as well.

Let $\mathcal{I}$ be the set of items. The recommender system maintains an item-to-item matrix M. For any two distinct items $i, j \in \mathcal{I}$, the (i, j) cell of M contains a measure of the similarity between i and j. In the setting of this section, (i, j) and (j, i) contain the covariance between i and j. In the setting of Section IV-A, the (i, j) cell contains the position, if any, of item i in j's related-items list, along with additional information such as numeric similarity strength. As users interact with the recommender system by making purchases, entering their preferences, *etc.*, the system continually accumulates more data and updates M at discrete intervals.

For each user u, the recommender system maintains a "record" $\mathcal{S}_u \subset \mathcal{I}$. As the user interacts with the system, some item t may be added to $\mathcal{S}_u$, reflecting that t is now related to the user. In some systems, the same item may be added to $\mathcal{S}_u$ multiple times: for example, the user may listen to a particular song, watch a movie, or purchase a product more than once. The system may also remove items from $\mathcal{S}_u$, but this is less common and not used for our attack.

Consider a toy case when a single user u interacts with the system between time τ_1 and $\tau_2 = \tau_1 + \Delta$, and t is added to the user's item list $\mathcal{S}_u$. Covariance between t and all other items in $\mathcal{S}_u$ must increase. Let M_1 be the matrix at time τ_1, M_2 the matrix at time τ_2, and $M_\Delta = M_2 - M_1$. Then, for all items $s_i \in \mathcal{S}_u$, the (s_i, t) entry of M_Δ will be positive. Of course, real-world recommender systems interact concurrently with multiple users whose item sets may overlap.

Intuitively, the attack works as follows. The attacker has auxiliary information about some items in the target user's record (Section III). By observing simultaneous increases in covariances between auxiliary items and some other item t, the attacker can infer that t has been added to the user's record.

Formally, the attacker's auxiliary information is a subset $\mathcal{A} \subseteq \mathcal{S}_u$. It helps—but is not necessary—if $\mathcal{A}$ is uniquely identifying, *i.e.*, for any other user u_j of the recommender system, $\mathcal{A} \nsubseteq \mathcal{S}_{u_j}$. This is possible if items in $\mathcal{A}$ are less popular or if $\mathcal{A}$ is large enough [26].

The attacker monitors the recommender system and obtains the covariance matrix M at each update. Let $\mathcal{T} \subseteq \mathcal{I} \setminus \mathcal{A}$ be the set of items the user may have selected. The attacker observes the submatrix of M formed by rows corresponding to the items in $\mathcal{T} \cup \mathcal{A}$ and columns corresponding to the items in $\mathcal{A}$. Call

this submatrix N. Since $\mathcal{A} \subseteq \mathcal{S}_u$, when an item $t \in T$ is added to $\mathcal{S}_u$, covariances between t and many $a_i \in \mathcal{A}$ will increase. If the attacker can accurately recognize this event, he can infer that t has been added to $\mathcal{S}_u$.

The inference procedure is significantly complicated by the fact that when an item is added to $\mathcal{S}_u$, not all of its covariances are updated at the same time due to processing delays. In particular, (t, a_i) covariances for $a_i \in \mathcal{A}$ may update at different times for different auxiliary items a_i. Furthermore, auxiliary items may enter the system at or around the same time as t. We cannot use the (t, a_i) covariance unless we are certain that the addition of item a_i to u's record has been reflected in the system. Before attempting an inference, we compute the subset of auxiliary items which "propagated" into the covariance matrix. The algorithm works by measuring increases in pairwise covariances between auxiliary items; we omit the details due to space limitations. In the following, we refer to this algorithm as PROPAGATEDAUX.

Constructing delta matrices. Suppose the attacker observes the covariance submatrices $N_{\tau_1}, N_{\tau_2}, \ldots$ at times $\tau_1, \tau_2, \ldots$. For each observation, the attacker creates a delta matrix N_Δ which captures the relevant *changes* in covariances. There are several ways to build this matrix. In the following, $\tau_{\max}$ is a parameter of the algorithm, representing the upper bound on the length of inference windows.

Strict time interval. For each τ_i, set $N_\Delta = N_{\tau_{i+1}} - N_{\tau_i}$. Since not all covariances may update between τ_i and τ_{i+1}, some entries in N_Δ may be equal to 0.

First change. For each τ_i, N_Δ consists of the first changes in covariance after τ_i. Formally, for each entry (x, y) of N, let $\tau_k > \tau_i$ be the first time after τ_i such that $\tau_k \leq \tau_{\max}$ and $N_{\tau_k}[x][y] \neq N_{\tau_i}[x][y]$. Set $N_\Delta[x][y] = N_{\tau_k}[x][y] - N_{\tau_i}[x][y]$.

Largest change. Similar to first change.

Making an inference. The attacker monitors changes in the submatrix N. For each relevant interval Δ, the attacker computes the delta matrix N_Δ as described above and uses PROPAGATEDAUX to compute which auxiliary items have propagated into N. Then he applies Algorithm 2. In this algorithm, $scoreSet_t$ is the set of all auxiliary items whose pairwise covariances with t increased, $support_t$ is the subset of $scoreSet_t$ consisting of auxiliary items which have propagated, $score_t$ is the fraction of propagated items whose covariances increased. If $score_t$ and $support_t$ exceed certain thresholds (provided as parameters of the algorithm), the attacker concludes that t has been added to the user's record.

Inference algorithms against real-world recommender systems require fine-tuning and adjustment. Algorithm 2 is only a high-level blueprint; there are many system-specific variations. For example, the algorithm may look only for increases in covariance that exceed a certain threshold.

C. Inference attack on kNN recommender systems

Our primary focus is on passive attacks, but for completeness we also describe a simple, yet effective active attack on the *k-nearest neighbor* (kNN) recommendation algorithm [1]. Consider the following user-to-item recommender system. For each user U, it finds the k most similar users according to some similarity metric (*e.g.*, the Pearson correlation coefficient or cosine similarity). Next, it ranks all items purchased or rated by one or more of these k users according to the number of times they have been purchased and recommends them to U in this order. We assume that the recommendation algorithm and its parameters are known to the attacker.

Now consider an attacker whose auxiliary information consists of the user U's partial transaction history, *i.e.*, he already knows m items that U has purchased or rated. His goal is to learn U's transactions that he does not yet know about.

The attacker creates k sybil users and populates each sybil's history with the m items which he knows to be present in the target user U's history. Due to the sparsity of a typical transaction dataset [26], $m \approx O(\log N)$ is sufficient for the attack on an average user, where N is the number of users. (In practice, $m \approx 8$ is sufficient for datasets with hundreds of thousands of users.) With high probability, the k nearest neighbors of each sybil will consist of the other $k-1$ sybils and the target user U. The attacker inspects the list of items recommended by the system to any of the sybils. Any item which appears on the list and is *not* one of the m items from the sybils' artificial history must be an item that U has purchased. Any such item was not previously known to the attacker and learning about it constitutes a privacy breach.

This attack is even more powerful if the attacker can adaptively change the fake history of his sybils after observing the output of the recommender system. This capability is supported by popular systems—for example, Netflix users can change previously entered ratings, while Amazon users can tell the site to ignore certain transactions when making recommendations—and allows the attacker to target multiple users without having to create new sybils for each one.

Algorithm 2: MATRIXINFERENCE

Input: Set of target items $\mathcal{T}$, set of auxiliary items $\mathcal{A}$, PROPAGATEDAUX returns a subset of $\mathcal{A}$, implementation-specific parameters $threshold_{support,score}$
Output: Subset of items from $\mathcal{T}$ which are believed by the attacker to have been added to $\mathcal{S}_u$

```
inferredItems = {}
foreach observation time τ do
    propagated_τ = PROPAGATEDAUX(A, τ)
    Δ = observation period beginning at τ
    N_Δ = delta matrix containing changes in covariances
    between items in T ∪ A
    foreach item t in T do
        scoreSet_t = subset of a ∈ A such that N_Δ[t][a] > 0
        support_t = |scoreSet_t ∩ propagated_τ|
        score_t = |support_t| / |propagated_τ|
        if score_t ≥ threshold_score and
        support_t ≥ threshold_support then
            inferredItems = inferredItems ∪ {t}
return inferredItems
```

D. Attack metrics

Our attacks produce *inferences* of this form: "Item Y was added to the record of user X during time period T." The main metrics are yield and accuracy. **Yield** is the number of inferences per user per each observation period, regardless of whether those inferences are correct. **Accuracy** is the percentage of inferences which are correct. We use yield rather than alternative metrics that focus on the number of correct inferences because the attacker can adjust the parameters to control the number of inferences made by our algorithm but cannot directly control the number or proportion that are correct. Where it makes sense, we also express yield as the percentage of the user's transactions inferred by our algorithm, but in general, we focus on the absolute number of inferences.

High yield and high accuracy are not simultaneously necessary for an attack to be dangerous. A single accurate inference could be damaging, revealing anything from a medical condition to political affiliation. Similarly, a large number of less accurate inferences could be problematic if their implications are uniformly negative. While the victim may retain plausible deniability for each individual inference, this provides little or no protection against many privacy violations. For example, plausible deniability does not help in situations where judgments are based on risk (*e.g.*, insurance) or prejudice (*e.g.*, workplace discrimination), or where the inferred information further contributes to a negative narrative (*e.g.*, confirms existing concerns that a spouse is cheating).

There is an inherent tradeoff between yield and accuracy. The higher the yield, the higher the number of incorrect inferences ("false positives"). Different combinations of parameters for our algorithms produce either more inferences at the cost of accuracy, or fewer, but more accurate inferences. Therefore, we evaluate our algorithms using the **yield-accuracy curve**.

V. Inference vs. prediction

At first glance, our inference algorithms may look similar to standard collaborative filtering algorithms which attempt to *predict* the items that a user may like or purchase in the future based on his and other users' past transactions.

The two types of algorithms are completely different, both technically and conceptually. We infer the user's *actual transactions*—as opposed to using the known behavior of similar users to guess what he may do or have done. Prediction algorithms discover common patterns and thus have low sensitivity to the presence or absence of a single user. Our algorithms are highly sensitive. They (1) work *better* if there are no similar users in the database, but (2) do not work if the target user is not the database, even if there are many similar users.

Collaborative filtering often exploits covariances between items; our algorithms exploit *changes* in covariance over time. The accuracy of predictions produced by collaborative filtering does not change dramatically from period to observation period; by contrast, we infer the approximate *date* when the transaction occurred, which is very hard to discover using collaborative filtering. Finally, our algorithms can infer even transactions involving very obscure items. Such items tend to populate lower ranges of auxiliary items' similarity lists, where a single transaction has the biggest impact. Section VII shows that transactions involving obscure items are more likely to be inferred by our algorithms.

Prediction quality can be seen as a baseline for feasible inference quality. A prediction is effectively an expected probability that a user with item a will select some target item t at any time. If a user with item a selects item t during a given time period, he exceeds this expected probability, causing a temporary rise (until other users balance the impact). By looking at changes in predictions over short periods of time, we can reconstruct how user behavior deviated from the predictions to produce the observed changes. This yields more accurate information than predictions alone. As Sections VI-A and VII show, our algorithms not only outperform a Bayesian predictor operating on the same data, but also infer items ranked poorly by a typical prediction algorithm.

Finally, it is worth mentioning that we use some machine-learning techniques for tuning inference algorithms that operate on related-items lists (see Section VI-C). These techniques are very different from collaborative filtering. Whereas collaborative filtering attempts to predict future behavior based on past behavior of other users, our models are backward-facing. We know that an item has risen in a similarity list, but we don't know why. To produce accurate inferences, we must learn which observations are sufficient to conclude that this rise signals addition of the item to the target user's record. In summary, we use machine learning to *learn the behavior of the recommender system itself*, not the behavior of its users.

VI. Evaluation on real-world systems

We evaluated our inference algorithms on several real-world recommender systems. Our goal was not to carry out an actual attack, but to demonstrate the feasibility and measure the accuracy of our algorithms. Therefore, all experiments were set up so that we knew each user's record in advance because the user either revealed it voluntarily through the system's public interface or cooperated with us. This provided the "ground-truth oracle," enabling us to measure the accuracy of our inferences without violating anyone's privacy.

A. Hunch

Hunch.com provides personalized recommendations on a wide range of topics. For each topic, Hunch poses a series of multiple-choice questions to the user and uses the responses to predict the user's preferences. Hunch also has a large set of generic personal questions in the category "Teach Hunch About You" (THAY), intended to improve topic recommendations. Hunch aggregates collected data and publishes statistics which characterize popular opinions in various demographics. For example, according to responses given to Hunch, "birthers" are 94% more likely to say that cultural activities are not important to them and 50% more likely to believe in alien abductions [16].

Statistics collected by Hunch are accessible via an API. They include the number of users responding to each THAY

question, the percentage selecting each possible answer, the number of users who responded to each pair of questions, and covariances between each pair of possible answers.

We show that aggregate statistics available via the Hunch API can be used to infer an individual user's responses to THAY questions, even though these responses are not made public by Hunch. Suppose the attacker knows some auxiliary information about a Hunch user (*e.g.*, height, hair color, age, hometown, political views) which allows the attacker to reliably predict how the user will respond to the corresponding THAY questions. We refer to the latter as AUX questions. See Section III for possible sources of auxiliary information.

Setup. The attacker forms a list of questions consisting of both AUX questions and questions for which he does *not* know the user's responses. We refer to the latter as TARGET questions; the objective of the experiment is to infer the user's responses to them. For our experiment, we chose questions with at least 4 possible answers. There were 375 such questions in the THAY set at the time of our experiment with simulated users (see below), but new THAY questions are continually added and users may even suggest new questions.

Immediately prior to the attack, the attacker uses the API function `responsePairStats` to collect all pairwise covariances between possible answers to questions on his list. Next, he directs the target user to specific questions from his list via links of the form http://hunch.com/people/⟨username⟩/edit-answer/?qid=⟨qid⟩ where ⟨username⟩ is replaced by the target user's username and ⟨qid⟩ is replaced with the question id. The attacker must know the username, but the site provides a social networking feature with profile pages where usernames can be linked to real names, interests, and other personal information. We assume that the user responds to all questions at the same time and that his responses to most AUX questions match the attacker's auxiliary information (our inference algorithms are robust to some mistakes in AUX).

Our goal is to show that individual responses can be inferred from the public outputs of recommender systems, not to conduct an actual attack. Therefore, we omit discussion of mechanisms for convincing a Hunch user to respond to a set of THAY questions. Similarly, it is a matter of opinion which questions and answers constitute sensitive information about an individual. For our purposes, it is sufficient to show that the attacker can infer the values of the user's secret responses to questions chosen by the attacker.

Data collection. Hunch does not update the covariance matrix immediately after the user responds to the attacker-supplied questions. At the time of our experiment, Hunch had approximately 5,000 possible answers to THAY questions and thus had to keep statistics on 12.5 million answer pairs. The update cycle of pairwise statistics varies, but seems to be on the order of 2-3 weeks. Each day during this period, for each known AUX response a_i, the attacker uses `responsePairStats` to collect the covariances between (1) a_i and all possible answers to TARGET questions, and (2) a_i and a_j, where $i \neq j$ (*i.e.*, cross-covariances between all AUX responses).

Algorithm 3: HUNCHINFERENCE

Input: Set $\mathcal{Q}$ of non-overlapping sets $\mathcal{R}_q$ containing all possible answers to each TARGET question q, set of known responses to AUX questions $\mathcal{A}$, PROPAGATEDAUX returns a subset of $\mathcal{A}$, implementation-specific parameters $threshold_{support,score}$
Output: Inferred responses to TARGET questions q

```
inferredResponses = {}
foreach answer set R_q in Q do
    maxScore = threshold_score
    maxSupport = threshold_support
    foreach observation time τ do
        propagated_τ = PROPAGATEDAUX(A, τ)
        Δ = observation period beginning at τ
        N_Δ = delta matrix containing changes in covariances
        between items in R_q ∪ A
        foreach TARGET answer r in R_q do
            scoreSet_r = subset of a ∈ A such that
            N_Δ[r][a] > 0
            support_r = |scoreSet_r ∩ propagated_τ|
            score_r = |support_r| / |propagated_τ|
            if score_r ≥ threshold_score then
                if support_r > maxSupport then
                    inferredResponses[q] = {r}
                    maxSupport = support_r
                    maxScore = score_r
                else if support_r = maxSupport then
                    if score_r > maxScore then
                        maxScore = score_r
                        inferredResponses[q] = {r}
                    else if score_r == maxScore then
                        inferredResponses[q] =
                        inferredResponses[q] ∪ {r}
return inferredResponses
```

The above covariances are not updated simultaneously, which greatly complicates the attacker's task. Hunch appears to split THAY questions into chunks and update pairwise answer statistics one chunk at a time. For instance, covariances between possible answers to question 1 and question 2 may update on Tuesday and Friday, while covariances between answers to question 1 and question 3 update on Thursday. The attacker must be able to detect when the covariances he is interested in have "propagated" (see Section IV-B).

Inferring secret responses. Algorithm 3 shows the inference procedure. Intuitively, the algorithm looks for a subset of AUX answers whose cross-covariances have increased (indicating that they propagated into the covariance matrix), and then for a single answer to each of the TARGET questions whose covariances with most of the AUX responses in the propagated subset have increased simultaneously.

For the algorithm to work, it is essential that large chunks of AUX responses propagate into the covariance matrix at the same time (as is the case for Hunch). The attacker can expect to see large positive shifts in covariance between the user's (known) responses to AUX questions and (unknown) responses to TARGET questions soon after both AUX and TARGET have propagated. The larger the number of AUX

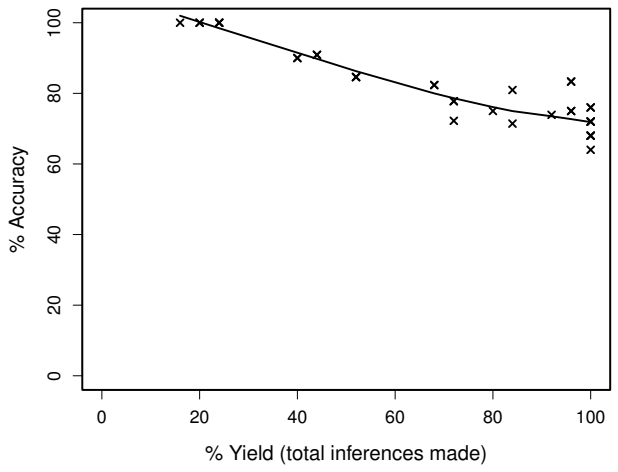

Fig. 1. Hunch: Accuracy vs. yield for real users. Each point represents a particular tuning of the algorithm, $threshold_{score}$ ranges from 45% to 78%, $threshold_{support}$ ranges between 32% and 57% of AUX size.

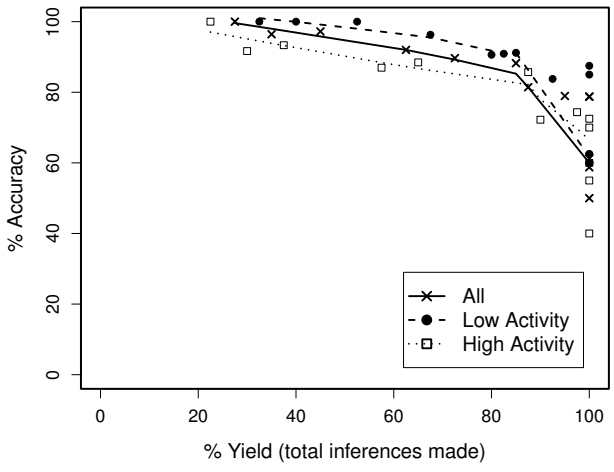

Fig. 2. Hunch: Accuracy vs. yield for simulated users: average of 8 users, 4 users assigned low-activity questions, 4 users assigned high-activity questions, $threshold_{score}$ ranges from 40% to 75%, $threshold_{support}$ ranges between 28% and 55% of AUX size.

questions for which this pattern is observed, the higher the attacker's confidence that the TARGET answer for which covariances have increased is the user's true response.

Results. For the experiment with real users, we used 5 volunteers and chose THAY questions with at least 4 possible answers. Questions were ordered by sample size, and each user was assigned 20 questions in a round-robin fashion; 15 were randomly designated as AUX and 5 as TARGET. We requested that users respond honestly to all questions and collected their responses to serve as the "ground-truth oracle." After all responses were entered into Hunch, we collected pairwise answer statistics via the API as described above and applied Algorithm 3 to infer the responses to TARGET questions.

Results are shown in Fig. 1 in the form of a yield-accuracy curve, with each point corresponding to a particular setting of the algorithm's parameters. We constructed a linear relation between $threshold_{score}$ and $threshold_{support}$ parameters which produced good results across all experiments. We use this relation for all Hunch graphs. Parameter ranges are listed in captions. Here yield is the fraction of unknown responses for which the algorithm produces candidate inferences and accuracy is the fraction of candidate inferences that are correct.

For the experiment on simulated users, we used all 375 Hunch THAY questions with at least 4 possible answers. We monitored the number of users responding to each question (calculated as change in sample size) for 1 week prior to the experiment and ranked questions by activity level. The 40 questions with the lowest activity were assigned to user A, the next 40 to user B, *etc.*, for a total of 9 users. Due to a data collection error, the data for one user had to be discarded.

For each user, 30 questions were randomly assigned as AUX and 10 as TARGET. The simulated users "selected" answers

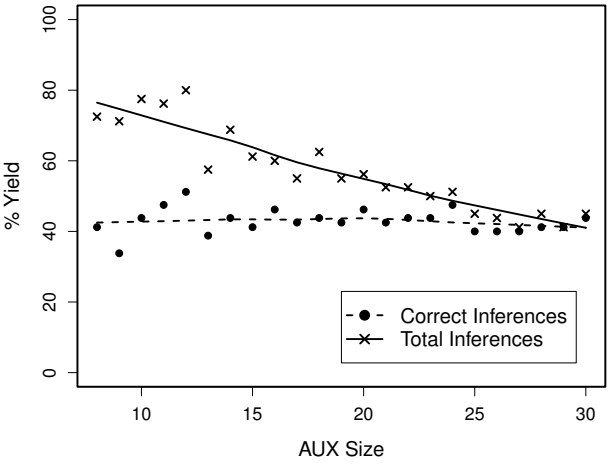

Fig. 3. Hunch: Yield vs. size of AUX for simulated users. $threshold_{score}$ is 70%, $threshold_{support}$ is 51.25% of AUX size.

following the actual distribution obtained from Hunch, *e.g.*, if 42% of real users respond "North America" to some question, then the simulated user selects this answer with 0.42 probability. Results are in Fig. 2. As expected, the inference algorithm performs better on less active questions. Overall, our algorithm achieves 78% accuracy with 100% yield.

Fig. 3 shows, for a particular setting of parameters, how yield and accuracy vary with the size of auxiliary information. As expected, larger AUX reduces the number of incorrect

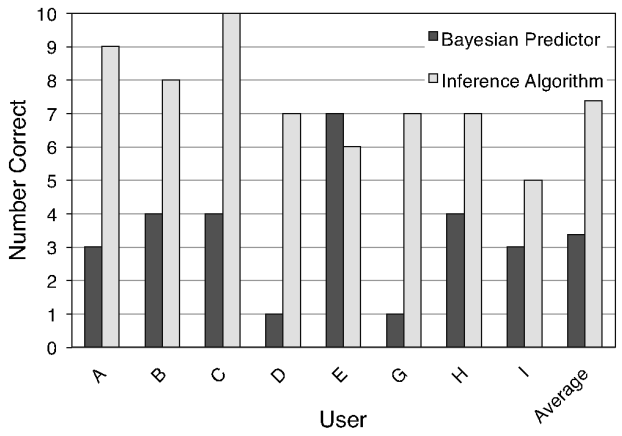

Fig. 4. Inference vs. Bayesian prediction on simulated Hunch users. $threshold_{score}$ is 55%, $threshold_{support}$ is 40% of AUX size. We used low thresholds to ensure that our algorithm reached 100% yield.

inferences, resulting in higher accuracy.

Inference vs. prediction. To illustrate the difference between inference and prediction, we compared the performance of our algorithm to a Bayesian predictor. Let X_i be a possible answer to a TARGET question and Y_j be the known response to the jth AUX question. The predictor selects X_i to maximize $P(X_i|Y_1)P(X_i|Y_2)\cdots P(X_i|Y_n)$, where $P(X_i|Y_j) = \frac{P(X_iY_j)}{P(Y_j)} = \frac{covar(X_i,Y_j)+P(X_i)P(Y_j)}{P(Y_j)}$. Individual probabilities and covariances are available from the Hunch API. As Fig. 4 shows, our algorithm greatly outperforms the Bayesian predictor, averaging 73.75% accuracy vs. 33.75% accuracy for the predictor. This is not surprising, because our algorithm is not making educated guesses based on what similar users have done, it is observing the effects of actual transactions!

B. LibraryThing

LibraryThing is an online recommender system for books with over 1 million members and 55 million books in their online "libraries" [19]. For each book, LibraryThing provides a "People with this book also have... (more common)" list and a "People with this book also have... (more obscure)" list. These lists have 10 to 30 items and are updated daily.

Algorithm 4 shows our inference procedure. Here delta matrices contain changes in "related books (common)" and "related books (obscure)" lists. Because these lists are updated frequently, the algorithm takes a $window$ parameter. When a book is added to the user's library, we expect that, within $window$, it will rise in the lists associated with the "auxiliary" books. Algorithm 5 counts the number of such lists, provided they are associated with less popular auxiliary books.

We used the site's public statistics to select 3 users who had consistently high activity, with 30-40 books added to their collections per month. 200 random books from each user's collection were selected as auxiliary information. The experiment ran for 4 months. Results are shown in Fig. 5.

Algorithm 4: LIBRARYTHINGINFERENCE

Input: set of "common" delta matrices $\mathcal{N}_C$, set of "obscure" delta matrices $\mathcal{N}_O$, scoring function: $\mathbb{R}^{|\mathcal{A}|} \to \mathcal{P}(\mathcal{A})$, set of AUX items $\mathcal{A}$, lookup table of popularities P, $threshold_{pop}$ for popularity of AUX books, $threshold_{score}$ for score, $window$ time interval in which we expect changes to propagate
Output: Inferred books from the target user's library

```
inferences = {}
scoreSets = dict{}
foreach observation time τ do
    foreach delta matrix N_Δ in N_C, N_O within window of τ do
        foreach target item t in N_Δ do
            if t ∉ scoreSets then
                scoreSets[t] = {}
            scoreSets[t] = scoreSets[t] ∪
            SCOREFUNCTION(N_Δ[t], A, P, threshold_pop)
foreach target item t in keys of scoreSets do
    if |scoreSets[t]| ≥ threshold_score and t ∉ A then
        inferences = inferences ∪ {t}
return inferences
```

Algorithm 5: LTSCOREFUNCTION

Input: Delta-matrix row $\mathbb{T}_t$ corresponding to book t, implemented as a dictionary keyed on AUX books and containing the relative change in list position, set of AUX books $\mathcal{A}$, lookup table of popularities P, $threshold$ for popularity of AUX book
Output: Subset of AUX books with which t increased in correlation

```
scoreSet = {}
foreach a ∈ A do
    if popularity P[a] ≤ threshold then
        if T_t[a] > 0 then
            scoreSet = scoreSet ∪ {a}
return scoreSet
```

When parameters are tuned for higher yield, related-items inference produces many false positives for LibraryThing. The reason is that many public libraries have accounts on LibraryThing with thousands of books. Even if some book rises in K lists related to the target user's library, K is rarely more than 20, and 20 books may not be enough to identify a single user. False positives often mean that another LibraryThing user, whose library is a superset of the target user's library, has added the inferred book to *their* collection.

C. Last.fm

Last.fm is an online music service with over 30 million monthly users, as of March 2009 [18]. Last.fm offers customized radio stations and also records every track that a user listens to via the site or a local media player with a Last.fm plugin. A user's listening history is used to recommend music and is public by default. Users may choose not to reveal real-time listening data, but Last.fm automatically reveals the number of times they listened to individual artists and tracks. Public individual listening histories provide a "ground-truth oracle" for this experiment, but the implications extend to

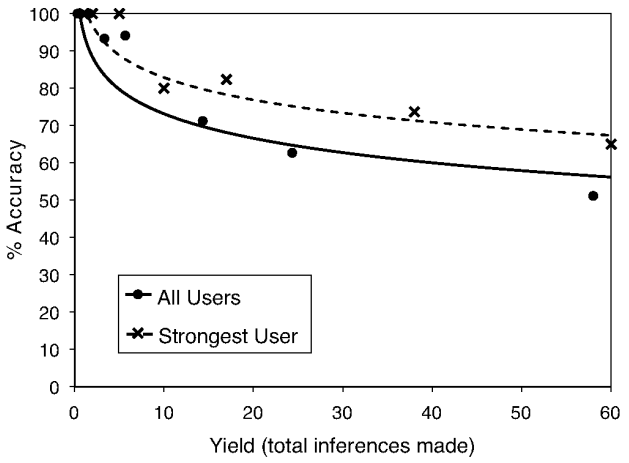

Fig. 5. LibraryThing: Accuracy vs. yield: for all users (averaged) and for the strongest user.

TABLE I
ATTRIBUTES OF CANDIDATE INFERENCES. COMPUTED SEPARATELY FOR AUX ITEMS FOR WHICH TARGET ROSE AND THOSE FOR WHICH IT FELL.

Attribute	Definition
$numSupports$	Number of AUX similarity lists in which TARGET rose/fell
$avgInitPos$	Average initial position of TARGET item in supporting AUX item similarity list
$avgChange$	Average change of TARGET item in supporting AUX item similarity lists
$avgListeners$	Average number of listeners for AUX items supporting TARGET
$avgPlays$	Average number of plays for AUX items supporting TARGET
$avgArtistScore$	Average number of other AUX supports by same artist as any given AUX support
$avgSimScore$	Average sum of match scores for all other AUX supports appearing in any given AUX support's updated similarity list
$avgTagScore$	Average sum of cosine similarity between normalized tag weight vectors of any given AUX support and every other AUX support

cases where individual records are private.

Aggregated data available through the Last.fm site and API include user-to-user recommendations ("neighbors") and item-to-item recommendations for both artists and tracks. We focus on track-to-track recommendations. Given a track, the API provides an ordered list of up to 250 most similar tracks along with "match scores," a measure of correlation which is normalized to yield a maximum of one in each list.

Our Last.fm experiment is similar to our LibraryThing experiment. Using changes in the similarity lists for a set of known auxiliary (AUX) tracks, we infer some of the TARGET tracks listened to by the user during a given period of time.

Unlike other tested systems, Last.fm updates all similarity lists simultaneously on a relatively long cycle (typically, every month). Unfortunately, our experiment coincided with changes to Last.fm's recommender system.[4] Thus during our six-month experiment, we observed a single update after approximately four months. Batched updates over long periods make inference more difficult, since changes caused by a single user are dominated by those made by millions of other users.

Several features of Last.fm further complicate adversarial inference. Last.fm tends not to publish similarity lists for unpopular tracks, which are exactly the lists on which a single user's behavior would have the most impact. The number of tracks by any given artist in a similarity list is limited to two. Finally, similarity scores are normalized, hampering the algorithm's ability to compare them directly.

Setup. We chose nine Last.fm users with long histories and public real-time listening information (to help us verify our inferences). As auxiliary information, we used the 500 tracks that each user had listened to most often, since this or similar information is most likely to be available to an attacker.

Inferring tracks. After each update, we make inferences using the generic algorithm for related-items lists (Algorithm 1) with one minor change: we assume that each value in N_Δ also stores the initial position of the item prior to the change.

Our scoring function could simply count the number of AUX similarity lists in which a TARGET track has risen. To strengthen our inferences, we use the scoring function shown in Algorithm 6. It takes into account information summarized in Table I (these values are computed separately for the lists in which the TARGET rose and those in which it fell, for a total of 16 values). To separate the consequences of individual actions from larger trends and spurious changes, we consider not only the number of similarity lists in which the TARGET rose and fell, but also the TARGET's average starting position and the average magnitude of the change. Because the expected impact of an individual user on an AUX item's similarity list depends on the popularity of the AUX item, we consider the average number of listeners and plays for AUX tracks as well.

We also consider information related to the clustering of AUX tracks. Users have unique listening patterns. The more unique the combination of AUX tracks supporting an inference, the smaller the number of users who could have caused the observed changes. We consider three properties of AUX supports to characterize their uniqueness. First, we look at whether supports tend to be by the same artist. Second, we examine whether supports tend to appear in each other's similarity lists, factoring in the match scores in those lists. Third, the API provides weighted tag vectors for tracks, so we evaluate the similarity of tag vectors between supports.

To fine-tune the parameters of our inference algorithm, we use machine learning. Its purpose here is to understand the Last.fm recommender system itself and is thus very different from collaborative filtering—see Section V. For each target user, we train the PART learning algorithm [12] on all other target users, using the 16 features and the known correct or incorrect status of the resulting inference. The ability to

[4]Private communication with Last.fm engineer, July 2010.

Algorithm 6: LASTFMSCOREFUNCTION

Input: Delta-matrix row $\mathbb{T}_t$ corresponding to item t. Each vector entry contains $initPos$ - the initial position of t in the list, and $change$ - the number of places t has risen or fallen in the list. We also assume that each AUX item is associated with $(listeners, plays, simList, tags)$ which are, respectively, the number of listeners, the number of plays, the updated similarity list with match scores, and the vector of weights for the item's tags.

Output: A $score$ for the row

$score = 0$
$\mathcal{A}_{rise} = \{a \in \mathbb{T}_t$ such that $\mathbb{T}_t[a][change] > 0\}$
$\mathcal{A}_{fall} = \{a \in \mathbb{T}_t$ such that $\mathbb{T}_t[a][change] < 0\}$
$data_{rise} = [0, 0, 0, 0, 0, 0, 0, 0]$
$data_{fall} = [0, 0, 0, 0, 0, 0, 0, 0]$
foreach $relevant \in \{rise, fall\}$ **do**
  $numSupports = |\mathcal{A}_{relevant}|$
  $avgInitPos = avg(\mathbb{T}_t[a][initPos]$ for $a \in \mathcal{A}_{relevant})$
  $avgChange = avg(\mathbb{T}_t[a][change]$ for $a \in \mathcal{A}_{relevant})$
  $avgListeners = avg(a[listeners]$ for $a \in \mathcal{A}_{relevant})$
  $avgPlays = avg(a[plays]$ for $a \in \mathcal{A}_{relevant})$
  $avgArtistScore = 0$
  $avgSimScore = 0$
  $avgTagScore = 0$
  foreach *aux item* $a \in \mathcal{A}_{relevant}$ **do**
    foreach *aux item* $a' \neq a$ *in* $\mathcal{A}_{relevant}$ **do**
      if $a[artist] == a'[artist]$ **then**
        $avgArtistScore+ = 1$
      if a' *in* $a[simList]$ **then**
        $simList = a[simList]$
        $avgSimScore+ = simList[a'][matchScore]$
      $tags = norm(a[tags])$
      $tags' = norm(a'[tags])$
      $avgTagScore+ = cosineSim(tags, tags')$
  if $numSupports > 0$ **then**
    $avgArtistScore = avgArtistScore/numSupports$
    $avgSimScore = avgSimScore/numSupports$
    $avgTagScore = avgTagScore/numSupports$
    $data_{relevant} = [numSupports, avgInitPos,$
    $avgChange, avgListeners, avgPlays,$
    $avgArtistScore, avgSimScore, avgTagScore]$
if $|\mathcal{A}_{rise}| > 0$ **then**
  $score =$ MLPROBCORRECT$(data_{rise}, data_{fall})$
return $score$

analyze data from multiple users is not necessary: a real attacker can train on auxiliary data for the target user over multiple update cycles or use his knowledge of the recommender system to construct a model without training. The model obtained by training on other, possibly unrepresentative users underestimates the power of the attacker.

After constructing the model, we feed the 16 features of each TARGET into the PART algorithm using the Weka machine-learning software [35]. Given the large number of incorrect inferences in the test data, Weka conservatively classifies most attempted inferences as incorrect. Weka also provides a probability that an inference is correct (represented by MLPROBCORRECT in Algorithm 6). We take these probabilities into account when scoring inferences.

We also account for the fact that similarity lists change in length and cannot contain more than two tracks by the same artist. For example, if a track was "crowded out" in a similarity list by two other tracks from the same artist, we know that its position was below the lowest position of these two tracks. Therefore, we judge the magnitude of any rise relative to this position rather than the bottom of the list. Similarly, if a list grows, we can confirm that a track has risen only if it rises above the bottom position in the previous list.

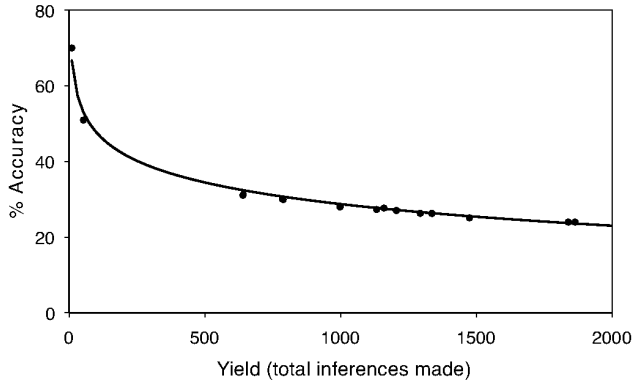

Fig. 6. Accuracy vs. yield for an example Last.fm user.

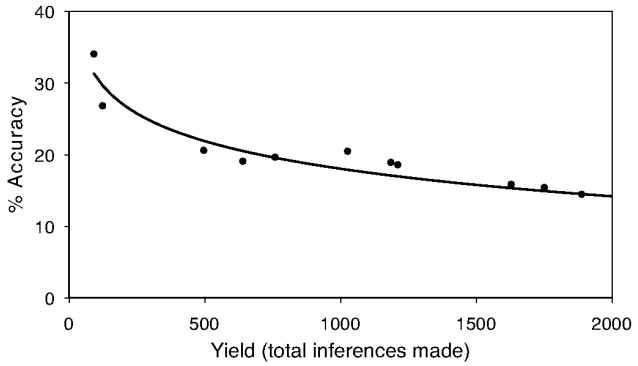

Fig. 7. Accuracy vs. yield for another Last.fm user.

Results. The performance of our inference algorithm was negatively impacted by the fact that instead of a typical monthly update, it had to make inferences from a huge, 4-month update associated with a change in the recommender algorithm.[5] Figs. 6 and 7 shows the results for two sample users; different points correspond to different settings of the threshold parameter of the algorithm. For the user in Fig. 6, we make 557 correct inferences at 21.3% accuracy (out of 2,612 total), 27 correct inferences at 50.9% accuracy, and 7 correct inferences at 70% accuracy. For the user in Fig. 7, we make 210 correct inferences at 20.5% accuracy and 31 correct inferences at 34.1% accuracy.

[5]Last.fm's changes to the underlying recommender algorithm during our experiment may also have produced spurious changes in similarity lists, which could have had an adverse impact.

For a setting at which 5 users had a minimum of 100 correct inferences, accuracy was over 31% for 1 user, over 19% for 3 users, and over 9% for all 5 users. These results suggest that there exist classes of users for whom high-yield and moderate-accuracy inferences are simultaneously attainable.

D. Amazon

We conducted a limited experiment on Amazon's recommender system. Without access to users' records, we do not have a "ground-truth oracle" to verify inferences (except when users publicly review an inferred item, thus supporting the inference). Creating users with artificial purchase histories would have been cost-prohibitive and the user set would not have been representative of Amazon users.

The primary public output of Amazon's recommender system is "Customers who bought this item also bought ..." item similarity lists, typically displayed when a customer views an item. Amazon reveals each item's sales rank, which is a measure of the item's popularity within a given category.

Amazon customers may review items, and there is a public list of tens of thousands of "top reviewers," along with links to their reviews. Each reviewer has a unique reviewer identifier. Reviews include an item identifier, date, and customer opinions expressed in various forms. Customers are not required to review items that they purchase and may review items which they did not purchase from Amazon.

Setup. Amazon allows retrieval of its recommendations and sales-rank data via an API. The data available via the API are only a subset of that available via the website: only the 100 oldest reviews of each customer (vs. all on the website) and only the top 10 similar items (vs. 100 or more on the website).

We chose 999 customers who initially formed a contiguous block of top reviewers outside the top 1,000. We used the entire set of items previously reviewed by each customer as auxiliary information. The average number of auxiliary items per customer varied between 120 and 126 during our experiment. Note that this auxiliary information is imperfect: it lacks items which the customer purchased without reviewing and may contain items the customer reviewed without purchasing.

Data collection ran for a month. We created a subset of our list containing *active* customers, defined as those who had written a public review within 6 months immediately prior to the start of our experiment (518 total). If a previously passive reviewer became active during the experiment, we added him to this subset, so the experiment ended with 539 active customers. For each auxiliary item of each active customer, we retrieved the top 10 most related items (the maximum permitted by the API)[6] daily. We also retrieved sales-rank data for all items on the related-item lists.[7]

Making inferences. Our algorithm infers that a customer has purchased some target item t during the observation period if t appears or rises in the related-items lists associated with at least K auxiliary items for the customer. We call the corresponding auxiliary items the *supporting items* for each inference. The algorithm made a total of 290,182 unique (user, item) inferences based on a month's worth of data; of these, 787 had at least five supporting items.

One interesting aspect of Amazon's massive catalog and customer base is that they make items' sales ranks useful for improving the accuracy of inferences. Suppose (case 1) that you had previously purchased item A, and today you purchased item B. This has the same effect on their related-items lists as if (case 2) you had previously purchased B and today purchased A. Sales rank can help distinguish between these two cases, as well as more complicated varieties. We expect the sales rank for most items to stay fairly consistent from day to day given a large number of items and customers. Whichever item was purchased today, however, will likely see a slight boost in its sales rank relative to the other. The relative boost will be influenced by each item's popularity, *e.g.*, it may be more dramatic if one of the items is very rare.

Case studies. Amazon does not release individual purchase records, thus we have no means of verifying our inferences. The best we can do is see whether the customer reviewed the inferred item later (within 2 months after the end of our data collection). Unfortunately, this is insufficient to measure accuracy. Observing a public review gives us a high confidence that an inference is correct, but the lack of a review does not invalidate an inference. Furthermore, the most interesting cases from a privacy perspective are the purchases of items for which the customer would *not* post a public review.

Therefore, our evidence is limited to a small number of verifiable inferences. We present three sample cases. Names and some details have been changed or removed to protect the privacy of customers in question. To avoid confusion, the inferred item is labeled t in all cases, and the supporting auxiliary items are labeled a_1, a_2, and so on.

Mr. Smith is a regular reviewer who had written over 100 reviews by Day 1 of our experiment, many of them on gay-themed books and movies. Item t is a gay-themed movie. On Day 20, its sales rank was just under 50,000, but jumped to under 20,000 by Day 21. Mr. Smith's previous reviews included items a_1, a_2, a_3, a_4, and a_5. Item t was not in the similarity lists for any of them on Day 19 but had moved into the lists for all five by Day 20. Based on this information, our algorithm inferred that Mr. Smith had purchased item t. Within a month, Mr. Smith reviewed item t.

Ms. Brown is a regular reviewer who had commented on several R&B albums in the past. Item t is an older R&B album. On Day 1, its rank was over 70,000, but decreased to under 15,000 by Day 2. Ms. Brown had previously reviewed items a_1, a_2, and a_3, among others. Item A moved into item a_1 and item a_2's similarity lists on Day 2, and also rose higher in item a_3's list that same day. Based on this information, our algorithm inferred that Ms. Brown had purchased item t. Within two months, Ms. Brown reviewed item t.

[6]The set of inferences would be larger (and, likely, more accurate) for an attacker willing to scrape complete lists, with up to 100 items, from the site.

[7]Because any item can move into and off a related-items list, we could not monitor the sales ranks of all possible target items for the full month. Fortunately, related-items lists include sales ranks for all listed items.

Mr. Grant is a regular reviewer who appears to be interested in action and fantasy stories. Item t is a fairly recent fantasy-themed movie. On Day 18, its sales rank jumped from slightly under 35,000 to under 15,000. It experienced another minor jump the following day, indicating another potential purchase. Mr. Grant's previous reviews included items a_1, a_2, and a_3. On Day 19, item t rose in the similarity lists of a_1 and a_2, and entered a_3's list. None of the supporting items had sales rank changes that indicate purchases on that date. Based on this information, our algorithm inferred that Mr. Grant had bought item t. Within a month, Mr. Grant reviewed item t.

In all cases, the reviewers are clearly comfortable with public disclosure of their purchases since they ultimately reviewed the items. Nevertheless, our success in these cases suggests a realistic possibility that sensitive purchases can be inferred. While these examples include inferences supported by items in a similar genre, we have also observed cross-domain recommendations on Amazon, and much anecdotal evidence suggests that revealing cross-domain inferences are possible. For example, Fortnow points to a case in which an Amazon customer's past opera purchases resulted in a recommendation for a book of suggestive male photography [11].

VII. Evaluation on a simulated system

To test our techniques on a larger scale than is readily feasible with the real-world systems that we studied, we performed a simulation experiment. We used the Netflix Prize dataset [28], which consists of 100 million dated ratings of 17,770 movies from 460,000 users entered between 1999 and 2005. For simplicity, we ignored the actual ratings and only considered whether a user rated a movie or not, treating the transaction matrix as binary. We built a straightforward item-to-item recommender system in which item similarity scores are computed according to cosine similarity. This is a very popular method and was used in the original published description of Amazon's recommender system [21].

We restricted our simulation to a subset of 10,000 users who have collectively made around 2 million transactions.[8] Producing accurate inferences for the entire set of 460,000 users would have been difficult. Relative to the total number of users, the number of items in the Netflix Prize dataset is small: 17,770 movies vs. millions of items in (say) Amazon's catalog. At this scale, most pairs of items, weighted by popularity, have dozens of "simultaneous" ratings on any single day, making inference very challenging.

Methodology. We ran our inference algorithm on one month's worth of data, specifically July 2005. We assumed that each user makes a random 50% of their transactions (over their entire history) public and restricted our attention to users with at least 100 public transactions. There are around 1,570 such users, or 16%. These users together made around 1,510 transactions during the one-month period in question.

[8]Reported numbers are averages over 10 trials; in each trial, we restricted the dataset to a random sample of 10,000 users out of the 460,000.

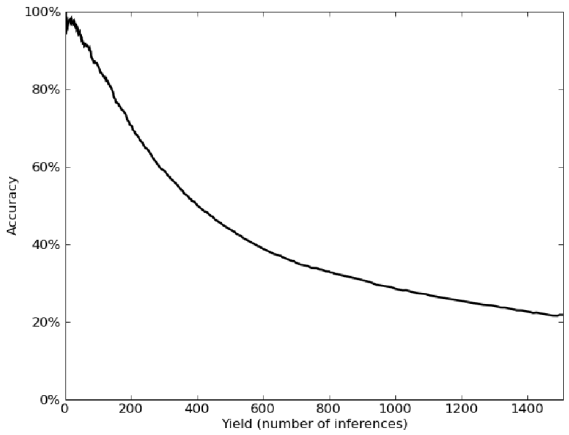

Fig. 8. Inference against simulated recommender: yield vs. accuracy.

We assume that each item has a public similarity list of 50 items along with raw similarity scores which are updated daily. We also assume that the user's total number of transactions is always public. This allows us to restrict the attack to (customer, date) pairs in which the customer made 5 or fewer transactions. The reason for this is that some users' activity is spiky and they rate a hundred or more movies in a single day. This makes inference too easy because guessing popular movies at random would have a non-negligible probability of success. We focus on the hard case instead.

Results. The results of our experiment are summarized in Fig. 8. The algorithm is configured to yield at most as many inferences as there are transactions: $1510/1570 \approx 0.96$ per vulnerable user for our one-month period. Of these, 22% are correct. We can trade off yield against accuracy by only outputting higher-scoring inferences. An accuracy of 80% is reached when yield is around 141, which translates to 7.5% of transactions correctly inferred. At an accuracy level of 90%, we can infer 4.5% of all transactions correctly. As mentioned earlier, these numbers are averages over 10 trials.

Inference vs. prediction. Fig. 9 shows that transactions involving obscure items are likely to be inferred in our simulated experiment. Fig. 10 shows that our inference algorithm accurately infers even items that a predictor would rank poorly for a given user. For this graph, we used a predictor that maximizes the sum-of-cosines score, which is the natural choice given that we are using cosine similarity for the item-to-item recommendations. The fact that the median prediction rank of inferred items lies outside the top 1,000 means, intuitively, that we are *not* simply inferring the items that the user is likely to select anyway.

VIII. Mitigation

Differential privacy. Differential privacy is a set of algorithms and analytical techniques to quantify and reduce privacy risk

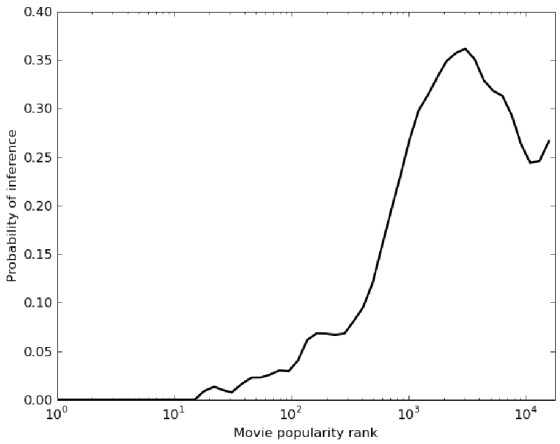

Fig. 9. Likelihood of inference as a function of movie popularity.

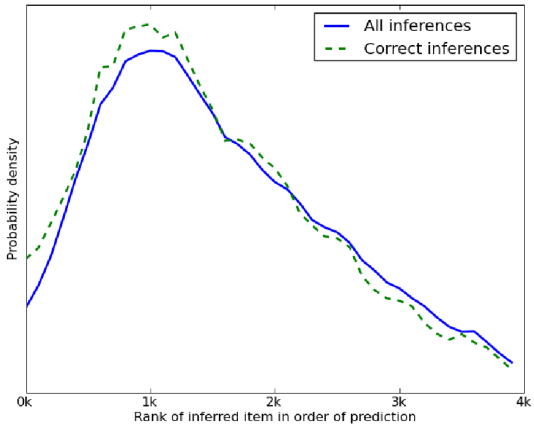

Fig. 10. Simulated recommender: Inferences vs. predictions.

to individual participants when answering statistical database queries [9]. Two threads of differential privacy research are potentially applicable to recommender systems. First, McSherry and Mironov showed how to construct differentially private covariance matrices, with some loss of accuracy for the collaborative filtering algorithms that use them [23]. While this is promising, they do not consider updates to the matrix, and it is not known how to build a dynamic system using this approach without destroying the privacy guarantee. The second line of research is on differential privacy under "continual observation," which directly addresses the needs of a dynamic system [5, 10]. These techniques break down, however, for systems with a large "surface area" such as similarity lists for every item. Designing a differentially private online recommender system remains an open problem.

Restrict information revealed by the system. Our inference algorithms exploit the fact that modern recommender systems make a large amount of information available for automated collection through their API. Restricting publicly available recommendation information may significantly complicate (but may not completely foil) the attack, while preserving legitimate functionality.

Limit the length of related-items lists. Amazon's "Customers who bought this item also bought ..." or Last.fm's "Similar music" lists can contain more than 100 items (although on Amazon, only the top 10 are available for automated collection via the API). Recommendations near the top of an item's related-items list have a strong relationship with that item, which is unlikely to be impacted significantly by a single purchase. The ordering of recommendations near the bottom of the list is more volatile and reveals more information.

Factor item popularity into update frequency. Less popular items tend to be more identifying, so limiting the frequency of updates involving these items may decrease the efficacy of our inference attacks. For example, purchases of less popular items may be batched and reflected in the item-to-item matrix only several times a year. The same applies to their "sales rank," which for rare items can be sensitive even to a single purchase. Unfortunately, this mitigation technique may dull the impact of sudden, important shifts in the data, such as a surge in popularity of a previously obscure book.

Avoid cross-genre recommendations. In general, "straddlers," *i.e.*, customers with interests in multiple genres, tend to be at higher risk for privacy breaches. This risk could be mitigated by avoiding cross-genre recommendations except when items have strong relationships. This has the shortcoming of obstructing potentially surprising, but useful recommendations.

Limit the speed and/or rate of data access. The large-scale, passive attacks described in this paper require that the attacker extract a somewhat large quantity of data from the recommender system over a long period of time. Limiting the speed of access (by rate-limiting the API, using crawler-detection heuristics, *etc.*) may reduce the amount of available data and consequently the scale of the attack. Unfortunately, this may also prevent some legitimate uses of the data. Furthermore, these limits may not stop smaller, more focused attacks or a capable attacker (*e.g.*, one with access to a botnet).

It is hard to set a limit which would completely foil the attack. For example, Hunch's API limits each "developer key" to 5,000 queries per day (an attacker can easily acquire multiple keys). Our experiments in Section VI-A required between 500 and 2,500 queries per day per target user.

User opt-out. Many sites allow users to opt out of their recommender systems, but the opt-out is often incomplete. Amazon allows customers to request that certain purchases not be used for their own personalized recommendations. This option is primarily used to prevent gift purchases from influencing one's recommendations. For customers with privacy concerns, a more useful option would be to prevent a purchase from influencing the recommender system in any manner at all.

If users habitually chose to opt out, however, recommendation quality could suffer significantly.

While each mitigation strategy has limitations, a careful combination of several techniques may provide substantial practical benefits with only modest drawbacks.

IX. Related work

Privacy and collaborative filtering. To our knowledge, this is the first paper to show how to infer individual behavior from the public outputs of recommender systems. Previous work on privacy risks of recommenders focused on "straddlers" whose tastes span unrelated genres and assumed that the attacker is given the entire (anonymized) database of user transactions [30]. This model may be applicable in scenarios where collaborative filtering is outsourced, but is unrealistic for real-world recommender systems. Similarly, de-anonymization attacks require access to static datasets [13, 26].

Shilling attacks on collaborative filtering systems [24, 25] aim to influence the system by causing certain items to be recommended more often. We briefly mention an active attack on user-to-item collaborative filtering which is somewhat similar, but pursues a completely different goal.

Research on "social recommendations"—made solely based on a social graph—has shown that accurate recommendations necessarily leak information about the existence of edges between specific nodes in the graph [22]. This work differs from ours in that it (i) does not model user transactions, only edges in the social graph, (ii) does not consider temporal dynamics, and (iii) analyzes recommendations made to a user rather than public recommendations.

Previous work on protecting privacy in collaborative recommender systems aimed to hide individual user records from the system itself [4, 29, 32, 36]. These papers do not address the risk that individual actions can be inferred from temporal changes in the system's public recommendations and do not appear to provide much protection against this threat.

Privacy of aggregated data. Our attacks belong to a broad class of attacks that infer individual inputs from aggregate statistics. Disclosure of sensitive data from statistical summaries has long been studied in the context of census data [33]. Dinur and Nissim showed theoretically that an attacker who can query for arbitrary subsets of rows of a private database can learn the entire database even if noise has been added to aggregated answers [7]. Differential privacy was developed in part to provide a rigorous methodology for protecting privacy in statistical databases [8, 9]. Attacks on statistical databases often exploit the aggregates that happen to involve too few individuals. By contrast, we show that even with large aggregates, temporal changes can reveal underlying inputs.

Homer et al. showed that given a statistical summary of allele frequencies of a DNA pool—such as might be published in a genome-wide association study (GWAS)—it is possible to detect whether or not a target individual is represented in the pool, provided that the attacker has access to the individual's DNA [15]. The attack exploits the fact that DNA is very high-dimensional, thus the number of attributes is much greater than the number of records under consideration. Wang et al. strengthened the attack of Homer et al. and also developed a second type of attack, which uses a table of pairwise correlations between allele frequencies (also frequently published in GWA studies) to *disaggregate* the table into individual input sequences [34]. By contrast, the inference attacks described in this paper are not based on disaggregation.

X. Conclusions

Recommender systems based on collaborative filtering have become an essential component of many websites. In this paper, we showed that their public recommendations may leak information about the behavior of individual users to an attacker with limited auxiliary information. Auxiliary information is routinely revealed by users, but these public disclosures are under an individual's control: she decides which items to review or discuss with others. By contrast, item similarity lists and item-to-item covariances revealed by a recommender system are based on *all* transactions, including ones that users would not disclose voluntarily. Our algorithms leverage this to infer users' non-public transactions, posing a threat to privacy. We utilize aggregate statistics which contain no "personally identifiable information" and are widely available from popular sites such as Hunch, Last.fm, LibraryThing, and Amazon. Our attacks are passive and can be staged by any user of the system. An active attacker can do even more.

We study larger, established sites as well as smaller and/or newer sites. Our results in the latter category are stronger, supporting the intuitive notion that **customers of larger sites are generally safer** from a privacy perspective and corroborating the findings in [23]. Smaller datasets increase the likelihood that individual transactions have a perceptible impact on the system's outputs.

Our work concretely demonstrates the risk posed by data aggregated from private records and **undermines the widely accepted dichotomy between "personally identifiable" individual records and "safe," large-scale, aggregate statistics**. Furthermore, it demonstrates that the *dynamics* of aggregate outputs constitute a new vector for privacy breaches. Dynamic behavior of high-dimensional aggregates like item similarity lists falls beyond the protections offered by any existing privacy technology, including differential privacy.

Modern systems have **vast surfaces for attacks on privacy**, making it difficult to protect fine-grained information about their users. Unintentional leaks of private information are akin to side-channel attacks: it is very hard to enumerate all aspects of the system's publicly observable behavior which may reveal information about individual users. Increasingly, websites learn from—and indirectly expose—aggregated user activity in order to improve user experience, provide recommendations, and support many other features. Our work demonstrates the inadequacy of current theory and practice in understanding the privacy implications of aggregated data.

ACKNOWLEDGEMENTS

We thank Ilya Mironov for useful discussions and Ian Davey, Benjamin Delaware, Ari Feldman, Josh Kroll, Joshua Leners, and Bill Zeller for helpful comments on earlier drafts of this paper. The research described in this paper was partially supported by the NSF grants CNS-0331640, CNS-0716158, and CNS-0746888, Google research awards, the MURI program under AFOSR grant no. FA9550-08-1-0352, and the DHS Scholarship and Fellowship Program under DOE contract no. DE-AC05-06OR23100.

REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 17(6), 2005.

[2] R. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix Prize. http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf.

[3] A. Borges. Toward a new supermarket layout: From industrial categories to one stop shopping organization through a data mining approach. In *SMA Retail Symposium*, 2003.

[4] J. Canny. Collaborative filtering with privacy. In *S & P*, 2002.

[5] H. Chan, E. Shi, and D. Song. Private and continual release of statistics. In *ICALP*, 2010.

[6] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *TISSEC*, 22(1), 2004.

[7] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, 2003.

[8] C. Dwork. Differential privacy. In *ICALP*, 2006.

[9] C. Dwork. Differential privacy: a survey of results. In *TAMC*, 2008.

[10] C. Dwork, M. Naor, T. Pitassi, and G. Rothblum. Differential privacy under continual observation. In *STOC*, 2010.

[11] L. Fortnow. Outed by Amazon. http://weblog.fortnow.com/2008/02/outed-by-amazon.html (Accessed Nov 17, 2010).

[12] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *ICML*, 1998.

[13] D. Frankowski, D. Cosley, S. Sen, L. Terveen, and J. Riedl. You are what you say: privacy risks of public mentions. In *SIGIR*, 2006.

[14] R. Garfinkel, R. Gopal, B. Pathak, R. Venkatesan, and F. Yin. Empirical analysis of the business value of recommender systems. http://ssrn.com/abstract=958770, 2006.

[15] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. Pearson, D. Stephan, S. Nelson, and D. Craig. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genet*, 4, 2008.

[16] http://blog.hunch.com/?p=8264 (Accessed Nov 19, 2010).

[17] D. Irani, S. Webb, K. Li, and C. Pu. Large online social footprints–an emerging threat. In *CSE*, 2009.

[18] http://blog.last.fm/2009/03/24/lastfm-radio-announcement (Accessed Nov 2, 2010).

[19] http://www.librarything.com/press/ (Accessed Nov 10, 2010).

[20] G. Linden, J. Jacobi, and E. Benson. Collaborative recommendations using item-to-item similarity mappings. U.S. Patent 6266649. http://www.patentstorm.us/patents/6266649/fulltext.html, 2008.

[21] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. In *IEEE Internet Computing*, January-February 2003.

[22] A. Machanavajjhala, A. Korolova, and A. Sarma. Personalized social recommendations - accurate or private? Manuscript, 2010.

[23] F. McSherry and I. Mironov. Differentially private recommender systems. In *KDD*, 2009.

[24] B. Mehta and W. Nejdl. Unsupervised strategies for shilling detection and robust collaborative filtering. *UMUAI*, 19(1–2), 2009.

[25] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams. Effective attack models for shilling item-based collaborative filtering systems. In *WebKDD*, 2005.

[26] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *S & P*, 2008.

[27] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *S & P*, 2009.

[28] http://www.netflixprize.com/rules (Accessed Nov 19, 2010).

[29] H. Polat and W. Du. Privacy-preserving top-n recommendation on horizontally partitioned data. In *Web Intelligence*, 2005.

[30] N. Ramakrishnan, B. Keller, B. Mirza, A. Grama, and G. Karypis. Privacy risks in recommender systems. In *IEEE Internet Computing*, November-December 2001.

[31] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

[32] R. Shokri, P. Pedarsani, G. Theodorakopoulous, and J-P. Hubaux. Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In *RecSys*, 2009.

[33] C. Sullivan. An overview of disclosure principles. U.S. Census Bureau Research Report, 1992.

[34] R. Wang, Y Li, X. Wang, H. Tang, and X. Zhou. Learning your identity and disease from research papers: information leaks in genome wide association study. In *CCS*, 2009.

[35] Weka 3 - data mining software in Java. http://www.cs.waikato.ac.nz/ml/weka/ (Accessed Nov 3, 2010).

[36] J. Zhan, C. Hsieh, I. Wang, T. Hsu, C. Liau, and D. Wang. Privacy-preserving collaborative recommender systems. In *SMC*, 2010.