Effective Anonymization of Query Logs*

Yuan Hong, Xiaoyun He, Jaideep Vaidya, Nabil Adam, and Vijayalakshmi Atluri
MSIS Department and CIMIC, Rutgers University
1 Washington Park, Newark, NJ, USA, 07102
{yhong,xiaoyun,jsvaidya,adam,atluri}@cimic.rutgers.edu

ABSTRACT

User search query logs have proven to be very useful, but have vast potential for misuse. Several incidents have shown that simple removal of identifiers is insufficient to protect the identity of users. Publishing such inadequately anonymized data can cause severe breach of privacy. While significant effort has been expended on coming up with anonymity models and techniques for microdata, there is little corresponding work for query log data. Query logs are different in several important aspects, such as the diversity of queries and the causes of privacy breach. This necessitates the need to design privacy models and techniques specific to this environment. This paper takes a first cut at tackling this challenge. Our main contribution is to define effective anonymization models for query log data along with proposing techniques to achieve such anonymization. We analyze the inherent utility and privacy tradeoff, and experimentally validate the performance of our techniques.

Categories and Subject Descriptors: H.2.0 [Database Management]: Security, integrity, and protection; H.3.m [Information Storage and Retrieval]

General Terms: Security

Keywords: Privacy

1. INTRODUCTION

Of all the data collected by search engines, the queries submitted by users are among the most valuable. Indeed, query logs can give great insight into human intent and have numerous diverse uses. However, they also have immense potential for misuse. Publishing of user query logs has become a sensitive issue. Similar to other data releases involving individual records such as microdata, the potentially personal content of query logs raises genuine privacy concerns [3, 8]. In particular, the recent incident involving AOL [3] has increased the public awareness of how the information in the query log file can be used to profile a single user without their knowledge. On the other hand, these search data, if published, are invaluable for researchers and law enforcement [8]. Thus, the challenge is to develop anonymization methods to publish query log data without breaching privacy or diminishing utility.

Indeed, query logs differ in several important characteristics, and their anonymization problem involves new challenges. First, the causes of privacy breach are not the same. Privacy breaches occur in released tabular data mainly due to the identification of an individual by combining quasi-identifiers in the data with other external knowledge or the value inferences of sensitive attributes. In the query logs, however, one of the big causes of privacy leaks is that the queries issued by a particular user unintentionally and inherently connote much individual identifying and sensitive information. In addition, even a single user usually conducts hundreds of searches over a short period of time. Putting these searches together may easily reveal the identity of the user. Moreover, unlike relational data, user queries are typically quite unstructured As such, it is clear that we cannot directly apply the techniques developed for anonymizing typical tabular data to the anonymization of query logs. Another problem is due to the diversity of queries – two users with the same information needs may easily formulate queries quite differently. Most queries are imprecise and short with average length of 2-3 words. This implies that it might not be practical to try to find exactly matching queries issued by a number of different users. However, it is possible to cluster similar queries [4, 2]. We can use this to measure similarity, and then define anonymization models using it.

In the most general sense, the query log anonymization problem is simply to transform a given query log QL into another query log QL' satisfying some privacy requirements. The framework of k-anonymity[11, 12] guarantees that every individual is hidden in a group of size k with respect to the quasi-identifiers. While it has some limitations, its applicability is widely accepted, and it does provide a measure of privacy. In this paper, we first follow the spirit of k-anonymity and define the anonymization problem of query logs. By following this formulation, we can ensure that a person cannot be uniquely identified from the searches.

However, k-anonymity cannot be directly applied to the query log anonymization problem, due to the fundamental difference in the domain. The key problem is that, for query logs, the quasi-identifiers which uniquely identify an individual are unstructured. At best, the similarity of users can be defined based on the similarity of the set of query records issued by each user. As a result, we introduce a similarity parameter δ (formally defined later), which indicates the degree of query similarity between users.

DEFINITION 1. *(k^δ-ANONYMITY) A query log QL satisfies k^δ-anonymity if for every user u who has a set of query records in QL, there exist at least $k-1$ other users who are δ-similar to user u based on their query records in QL.*

*This work is supported in part by the National Science Foundation under Grant No. CNS-0746943.

CIKM'09, November 2–6, 2009, Hong Kong, China.

2. SIMILARITY METRICS

To reduce loss of utility, we must first define similarity metrics between queries and users. Similar queries may be identified through clustering. However, users are related to both queries and clicked URLs through a bipartite graph. We then define a measure for user similarity based on both of these.

2.1 Query Similarity

In the graph based iterative clustering approach of [4], the query log click-through data is used to construct a bipartite graph, with unique queries on one side, and unique URLs on the other side, with links indicating a co-occurrence of a query and a URL in the same query log record. Let $\mathcal{N}(x)$ denote the set of vertices neighboring vertex x in a graph. Intuitively, vertex y is "similar" to vertex x if $\mathcal{N}(x)$ and $\mathcal{N}(y)$ have a large overlap. Formally, the similarity $sim(x, y)$ between vertices x and y is defined by $sim(x, y) := \frac{|\mathcal{N}(x) \cap \mathcal{N}(y)|}{|\mathcal{N}(x) \cup \mathcal{N}(y)|}$. Then, the distance $d(x, y)$ between vertices x and y is defined as follows: $d(x, y) := 1 - sim(x, y)$. It is easy to prove that $d(x, y)$ is metric. Based on this metric, we are able to cluster queries. Assume that $c_1, c_2, \ldots, c_m$ denote the resulting clusters after clustering all queries $(q_1, q_2, \ldots)$, irrespective of users, and m is the total number of clusters. To bound the diversity of queries belonging to a single cluster, we now define the diameter of a cluster:

DEFINITION 2. *Let $d(q_i, q_j)(q_i, q_j \in c_l)$ be the distance between queries q_i and q_j. The diameter of cluster c_l is*

$$diam(c_l) := \max_{q_i, q_j \in c_l} d(q_i, q_j).$$

The diameter parameter δ is used to decide whether to merge queries into a cluster (note that all pairs of queries in a cluster have to be within δ of each other). Thus, the distance between any two arbitrary queries in the same cluster is no greater than δ. The algorithm iteratively creates clusters from the unclustered queries, growing each cluster to include as many queries as possible. While we use this method, the clustering approach used is orthogonal to our work, and any alternative algorithm could also be followed. Finally, if there are any clusters whose sizes are quite small, say smaller than a specified threshold, we can eliminate such clusters with rare queries that may potentially be quite identifying.

2.2 User Similarity

Due to the ambiguity in query terms (e.g. does "Sun" stand for Sun, the company or Sun, the star?), the clicked links can represent the query objects with more precision. We thus prefer to quantify the similarity of query objects using the vector of numeric clicked URLs at the first step. Since the total number of query objects issued by each user varies, the count of clicked URLs is normalized, as follows: For each user u_x, the normalized clicked URL vector $v'_x = \frac{v_x(i)}{N_x}$, where $v_x(i)$ is the count of the ith unique URL for u_x and N_x represents the number of *user* x's total query objects. The dimension of each user's vector depends on how many unique URLs that user has clicked in a particular query log. Furthermore, we should try to find out the combination of these two elements (queries and clicked URLs). Since each query object includes exactly one clicked URL and one query that belongs to a unique cluster, we can decompose each user x's vector v'_x into a fraction matrix that consists of several vectors. The number of columns is given by the number of query clusters while the number of rows is the same as the number of unique URLs. Each element (in ith row and jth column) in the matrix represents the *normalized number* of entered queries in the ith cluster which link to the jth document. We can formally define this new matrix as follows:

DEFINITION 3. *(USER'S QUERY CLUSTER-URL MATRIX) Given a user's normalized URL vector v' of size n and query clusters $C = \{c_1, \ldots, c_m\}$, the user's Query-URL Matrix M is an $m \times n$ matrix with the element $m_{ij} = v'(j) \cdot fr_{ij}$ where fr_{ij} represents the fraction of queries with URL j belonging to cluster c_i. There exist: $\sum_{i=1}^{m} \sum_{j=1}^{n} m_{ij} = 1$ and $\forall j, \sum_{i=1}^{m} m_{ij} = v'(j)$.*

Since two users generally have different set of unique clicked URLs, the number of dimensions and corresponding URLs in the two vectors v'_x and v'_y are probably distinct. Thus, to measure the difference, we need to transform the two users "Query-URL" matrices into a consistent format. Essentially, we can transform each user's normalized Query-URL matrix into a consistent Query-URL matrix whose columns consist of the union of the columns of the original two matrices. This way, corresponding cells of both matrices represent the same query-url combination.

Now, the matrix-based distance metric for user similarity is defined as follows. For users u_x and u_y, their distance can be defined in terms of their transformed matrices M'_x and M'_y as follows:

$$D(u_x, u_y) := \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} |m'_{ij}(x) - m'_{ij}(y)| \quad (1)$$

where $m'_{ij}(x)$ and $m'_{ij}(y)$ represent the value of element in the ith row and jth column of M'_x and M'_y, and the distance between u_x and u_y is calculated by summing over all the absolute differences of two values in the same position of M'_x and M'_y. Note that $D(u_x, u_y)$ is a metric. The smaller the value of $D(u_x, u_y)$, the more similar the users are. Since $D(u_x, u_y)$ is the sum of all absolute difference values of every element in M'_x and M'_y, if there is no common clicked URL in those two query vectors or the common clicked URLs are associated with queries in totally different clusters, then $\sum_{i=1}^{m} \sum_{j=1}^{n} |m'_{ij}(x) - m'_{ij}(y)| = 2$, and therefore $D(u_x, u_y)$ has the range $[0, 1]$. The assumption here is that if users submit more similar set of queries and click more similar set of URLs, we consider them to be more similar. Recall that we have defined k^δ-Anonymity based on the δ-similarity of users, which is now defined below.

DEFINITION 4. *(δ-SIMILARITY OF USERS) Two users u_x and u_y are said to be δ-similar if the metric $D(u_x, u_y) = 0$ and the diameter of their query clusters is less than or equal to δ.*

3. ANONYMIZATION ALGORITHM

Given an existing set of queries with specified k and δ, anonymization is carried out to satisfy k^δ-anonymity. The anonymization process consists of the following two steps. First, we form groups of similar users with size no less than k. The second step involves making all of the users in a group indistinguishable from each other. Specifically, we try to add similar query objects and suppress dissimilar query objects so that each user in a group has the same normalized Query Cluster-URL matrix. That is, the normalized portion of queries falling in each clusters along with the associations with their clicked URLs is the same.

3.1 Constrained User Clustering

The first procedure is to group users such that each group includes at least k users. Obviously, we would like to group similar users together as much as possible. We first select one user as the *centroid user* of the group, calculate the similarity between this *centroid user* and all the other un-clustered users, and select users with their raw query objects from the top most similar users until the number of clustered users exceed or equal to k in one group. In

this process, the diameter among all users' query clusters should be less than δ, so the similarity between users is indirectly constrained by δ. However, the user-clustering process turns out to be primarily determined by the similarities of users, so we can consider the diameter of query clusters as the first constraint for grouping users. Meanwhile, we must ensure that every group has at least k users. Thus, we can define our user clustering approach as centroid-based user clustering method with constraints.

3.2 Anonymization

For any arbitrary pair of users, u_x and u_y, there may be some unique URLs (denoted as the set URL_1) that are clicked by u_x but not by u_y; alternatively, there may exist some unique URLs, denoted URL_2 which are clicked by u_y but not u_x; and there exist some common unique URLs, URL_3 for both of them as well. Similarly, there may exist three different kinds of unique query sets Q_1, Q_2 and Q_3 for u_x and u_y. If u_x is the *centroid user*, we should definitely add URL_1 and Q_1, suppress URL_2 and Q_2 and adjust URL_3 and Q_3 (add or suppress) for u_y to make the final normalized matrices similar. Therefore, we need both addition and suppression on original query logs during the anonymizing process. Let $U = \{u_1, \ldots, u_k\}$ be a group of k closest users. We first explain how we anonymize a pair of users, say $u_x, u_y \in U$. As discussed earlier, our aim is to make $D(u_x, u_y) = 0$, which is essentially equivalent to make all $m'_{ij}(x) = m'_{ij}(y)$ ($i \in [1, m]$ and $j \in [1, n]$). For instance, we let u_x's query objects Q_x be fixed and modify u_y's query objects Q_y according to Q_x. Without loss of generality, suppose that, after clustering and decomposing normalized clicked URLs, we have $m'_{ij}(x) > m'_{ij}(y)$ (Alternatively, if $m'_{ij}(x) < m'_{ij}(y)$, we can compute the suppression in a similar way). Let $n_{ij}(y)$ be the number of query objects in cluster c_i with a fixed URL url_j to be added to Q_y. We can then derive the following equation:

$$m'_{ij}(x) = \frac{N_y(ij) + n_{ij}(y)}{N_y} \tag{2}$$

$N_y(ij)$ is the number of query objects in which all queries are distributed into cluster c_i and all the clicked URLs are url_j. Thus, we have $n_{ij}(y) = \lceil m'_{ij}(x) \cdot N_y - N_y(ij) \rceil$ (we add more similar query objects if possible), and the number of addition on URLs is the same as $n_{ij}(y)$ as well. Alternatively, if $m_{ij}(x) < m_{ij}(y)$ and $n_{ij}(y)$ represents the number of suppression for a specified c_i and url_j, we have the following equation:

$$m'_{ij}(x) = \frac{N_y(ij) - n_{ij}(y)}{N_y} \tag{3}$$

Then, we have $n_{ij}(y) = \lfloor N_y(ij) - m'_{ij}(x) \cdot N_y \rfloor$ (we suppress less similar query objects if possible) for c_i and url_j.

When modifying the query log, remember that a URL is generally associated with queries from more than one cluster. It is possible that one of clusters may require addition of it while another cluster requires suppression. If we add and suppress URLs and queries, respectively, the relationship between them would be disorganized that leads to increased inaccuracy and reduces the utility of the anonymized query log. In our method, we initialize a query object $(q \in c_i, url_j)$ for anonymizing query objects in a specific query cluster c_i and a fixed URL url_j. If this query object is created as addition, the query q can be an arbitrary query in cluster c_i; Otherwise, the query q should be found in the non-centroid user's set of query objects. Also, how to add and suppress should also be considered. If we first add URL_1 for u_y until all the normalized vectors of clicked URLs in URL_1 are equal to the *centroid user*'s, then suppress all URL_2, and eventually adjust URL_3, the u_y's total number of URLs may be greatly changed, and the final result of the similarity/distance may be not accurate (in many cases, we cannot obtain $D(u_x, u_y) \approx 0$). In order to enhance the accuracy of similarity between anonymized query log and *centroid user*'s query log along with the similarity between anonymized query log and original query log for the same user (which also expresses good utility of anonymized query log), we can keep the total number of query objects from every user invariant and calculate the required addition and suppression. The actual queries to add or suppress are chosen equiprobably from among the candidates available.

4. EXPERIMENTAL EVALUATION

We extracted data from a large-scale query log to evaluate our approach and examine both privacy and utility behavior.

4.1 Clustering and Anonymization

Figure 2(a) shows that it is easier to find more similar users as the total number of users increases (the number of groups with at least k users continually increases, while the number of groups with less than k users shows a much smaller increase).

After clustering the users, the next stage in anonymization is adding and suppressing queries. FIgure 1(a) shows that the total number of added or suppressed queries for all users clearly increases with the number of users, though the number of queries per user decreases. This decrease is due to the increased similarity between users. Figure 1(b) plots the average number of queries that have to be added and suppressed for varying values of k. Similarly, Figure 1(c) plots the decreasing average number of total query objects for addition and suppression for varying values of δ due to the increasing similarity.

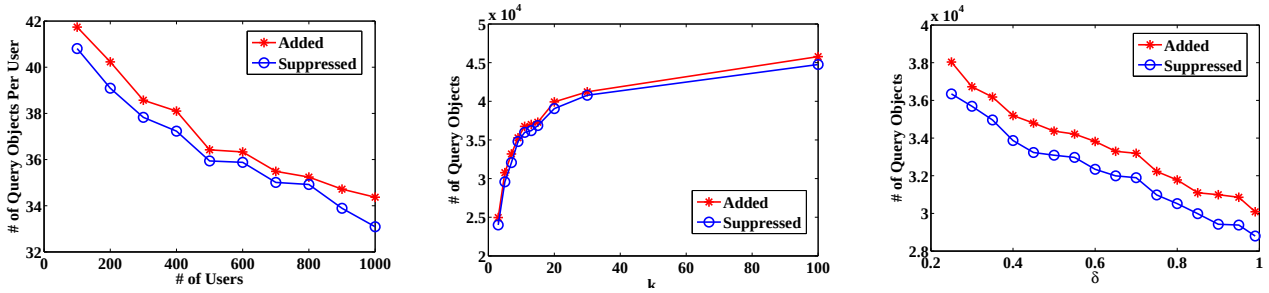
4.2 Utility Evaluation

We also tested the utility of our anonymized query objects in terms of the query diversity. Figure 2(b) shows the percentage of suppression against the number of users. The percentage of suppression is defined as $p = \frac{N - N'}{N}$ where N represents the number of unique queries or URLs while N' represents the number of anonymized ones. We can see that as the number of users increases, p decreases as well. We can conclude that the diversity would not be significantly impacted if the query log is large enough.

One of the most important applications of query log mining is query suggestion[5]. This requires accurately knowing the most frequent queries and clicked URLs, and in fact their co-occurrences. In this sense, the top frequent queries and clicked URLs reveal more utility than unusual queries and clicked URLs. We extracted the top 50 queries and clicked URLs that are frequently submitted from the tested query objects. Then, we count how many of them remain the top 50 of the anonymized query log. Figure 2(c) shows an increasing trend on these 50 queries and URLs in our experiments. With 1000 users, there are 24 common queries in the top 50 frequent queries in the original query log and anonymized query log. As for top 50 frequent URLs, the common URLs are 28 out of 50. Again, the figure also indicates that, if the size of the query log keeps increasing, these two top frequent query (URL) sets will become extremely similar. In general, we also observed that in many cases, the approximate ordering of the popular URLs/links is maintained. This indicates that we can achieve good utility using our proposed query log anonymization model.

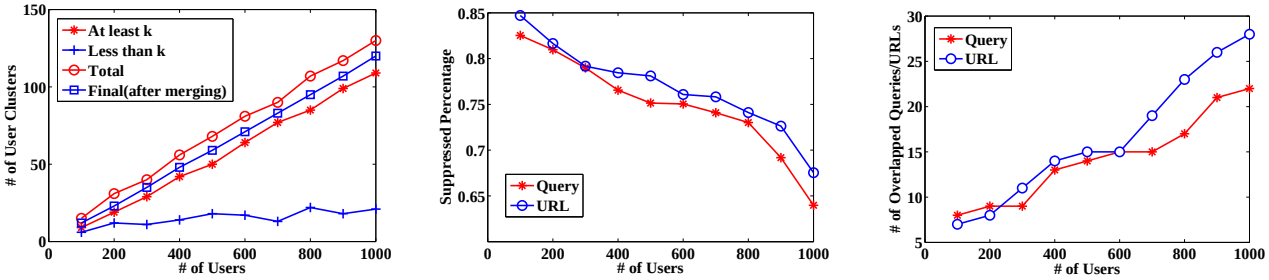
5. RELATED WORK

Following the AOL query log incident, recently there has been work on anonymizing query logs. Kumar et al.[10] propose a token-



(a) Varying $\#ofusers$ (b) Varying k (c) Varying δ

Figure 1: Anonymization on Varying Parameters



(a) User clustering on varying number of users (b) Percentage of suppression (c) Top 50 queries/URLs

Figure 2: Utility experiments

based hashing approach to anonymize query logs. In particular, each query in the log is tokenized, and then a secure hash function applied to produce hashes that are inverted. However, inversion cannot be done using just the token frequencies. It turns out that their token-based approach does not work since serious leaks are possible even when the order of tokens is hidden. Adar [1] proposes a secret sharing scheme, where a query must appear at least t (number of shares) times before it can be decoded. While this is empirically effective to a certain extent, it may potentially remove too many harmless queries, thus reducing data utility. Cooper [6] provides a survey on the query log privacy-enhancing techniques from a policy perspective, including log deletion, hashing queries, identifier deletion, hashing identifiers, etc. However, no methods are provided for anonymizing query logs. Recently, Korolova et al. [9] propose an algorithm to select and publish a subset of search query logs providing sound privacy guarantees (based on Differential Privacy[7]). However, we resolve this problem in a different way, and our work is complementary to that work. Samarati and Sweeney [11] first introduced the concept of k-anonymity to protect privacy. k-anonymity is the standard privacy model used, for example, by HIPAA and the European Union Data Directive.

6. CONCLUSIONS AND FUTURE WORK

Web search query logs of users have a number of uses including user behavior analysis, query characterization, among others. While such statistical analyses prove useful, releasing them without proper anonymization may compromise privacy of persons as query logs can potentially contain users' personal information. Since traditional anonymization techniques data are not directly applicable to query logs, in this paper, we present an anonymization model for query logs by extending the notion of k-anonymity and providing metrics for user similarity and query similarity. Our experimental results show that it is possible to achieve high user similarity in the anonymized logs quite efficiently.

In the future, we intend to test the suitability of our methods to different query log mining applications, and to develop methods specifically adapted for important applications. Since k-anonymity is also known to have several limitations, we will explore the applicability of other privacy models to this domain.

7. REFERENCES

[1] E. Adar. User 4xxxxx9: Anonymizing query logs. In *Workshop at the WWW '07*, Banff, Alberta, Canada, May 2007. WWW.

[2] R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza. Improving search engines by query clustering. *JASIST*, 58(12):1793–1804, 2007.

[3] M. Barbaro and T. Z. Jr. A face is exposed for aol searcher no. 4417749, Augest 9, 2006. (New York Times).

[4] D. Beeferman and A. L. Berger. Agglomerative clustering of a search engine query log. In *KDD*, pages 407–416, 2000.

[5] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD*, pages 875–883. ACM, 2008.

[6] A. Cooper. A survey of query log privacy-enhancing techniques from a policy perspective. *TWEB*, 2(4), 2008.

[7] C. Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, pages 1–12, Venice, Italy, July 9-16 2006.

[8] K. Hafner. Researchers yearn to use aol logs, but they hesitate, Augest 23, 2006. (New York Times).

[9] A. Korolova, K. Kenthapadi, N. Mishra, and A. Ntoulas. Releasing search queries and clicks privately. In *WWW*, pages 171–180, 2009.

[10] R. Kumar, J. Novak, B. Pang, and A. Tomkins. On anonymizing query logs via token-based hashing. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *WWW*, pages 629–638. ACM, 2007.

[11] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In *PODS '98*, page 188, New York, NY, USA, 1998. ACM.

[12] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.