Privacy-Preserving Top-k Queries

Jaideep Vaidya
Rutgers University and CIMIC
Newark, NJ 07102-1803
jsvaidya@rbs.rutgers.edu

Chris Clifton
Purdue University
W. Lafayette, IN 47907-2066
clifton@cs.purdue.edu

There is an increasing need for ranked and "best match" queries, in particular the top-k query: Finding the k closest matches to a query "point". Top-k queries are particularly relevant when dealing with privacy-sensitive information. For example, profiling has potential anti-terrorism applications (e.g., finding visa applicants who are likely terrorists). Profiles rarely match individuals exactly - a prime example of the need for similarity or best match queries. However, profiles often match innocent people (or transactions), subjecting them to unwarranted scrutiny.

By stripping a top-k query to its minimum – disclosing only an unranked set of k matches – we meet the goals of the top-k query while maintaining the privacy concept of *k-anonymity*[5]. In the context of profiling, k-anonymity requires that any individual that "matches" be indistinguishable from at least $k-1$ other individuals – in other words, all k would match. If k is large enough, then any use of the results will have to assume that most of the k are innocent (or good credit risks, or ...), and individuals will not be mistreated simply because they match the profile. We present an algorithm that meets the above privacy constraints in a distributed setting: Different sites have different attributes for each individual, and must not disclose individual information other than that inherently revealed by the top-k result.

Distributed processing of top-k queries *in the absence of privacy concerns* has been addressed. Fagin observed that if each site produces a list of its closest objects, and the lists have k items in common, then the *union* of the lists is guaranteed to contain the top-k. He used this observation to develop an algorithm that can work with $O(k)$ communication cost [3], although worst case could be $O(n)$.[1]

Our basic idea is to replicate Fagin's algorithm, while limiting disclosure to that inherently required to meet the efficiency goals of the algorithm. For example, if a site is required to test an object that is not close locally, it learns that this object must be somewhat close at another site – preventing this disclosure would require testing items that Fagin's algorithm would not, hurting efficiency. This disclosure is unlikely to raise privacy issues, and enables an algorithm whose typical cost is in terms of k rather than n. Although omitted due to space constraints, we have proven the disclosure properties of our algorithm under the semi-honest model, and shown somewhat stronger results assuming non-collusion of dishonest parties.

[1]To see the inherent difficulty of this problem, assume an object is close at most sites, but the farthest object at one site. The total distance could place it in the top-k, but the one site would not expect this.

1. Algorithm

We follow the basic structure of Fagin's $A0$ algorithm[3]. The A0 algorithm consists of 3 basic phases:

Sorted access phase: Each site i is accessed under sorted access, such that it starts outputting the graded tuples ordered on the local grade. This phase continues until there are at least k common objects in the output of all of the subsystems/sites. (i.e., the idea is to locally query and obtain sorted objects from each site until there are at least k common objects from all sites). Formally, if we represent the local set at site i as O_i, we continue until $|\bigcap_i O_i| \geq k$. Under a monotone grading function, the top-k objects are guaranteed to be within the union of all the objects [3].

The key to privacy is that everything in this phase is done locally except for testing if $|\bigcap_i O_i| \geq k$. The cardinality of set intersection protocol of [6], this check can be performed without any site disclosing local IDs.

Random access phase: For each object seen in the sorted access phase (i.e., $\forall o \in \bigcup_i O_i$), each subsystem randomly accesses its database to find the local score of that object.

Using the protocol of [4] we can compute the union $\bigcup_i O_i$ without revealing the sets O_i. While this does reveal objects not in the top-k, by increasing the candidate set of the previous phase in blocks of size k, we ensure that the disclosed objects are protected under k-anonymity privacy standards. Computing the local scores involve no disclosure, and thus no privacy loss.

Computation phase: Compute the global score for each object in $\bigcup_i O_i$. The top-k objects from this list are the required top-k objects.

The key to performing this phase while preserving privacy is to first sum the local distances so that two sites hold (random) shares of the global scores: For each object, each site j chooses a random r_j, and sends $score(j) + r_j$ to one site and $-r_j$ to the other. These sites find a distance threshold that includes k elements (Section 1.1), then use Yao's circuit evaluation / secure comparison approach [7] to determine if each object's global score is above or below that threshold without either site learning the global score.

The algorithm does place restrictions on the scoring functions; the score function must be expressible as a sum of locally computed functions. Fortunately, most common distance functions meet these restrictions, in particular the Manhattan and Euclidean metrics used in [2]. (We need not compute constant factors, square roots, etc.; since we only need a rank and not the actual distance score.)

Correctness follows from the correctness of Fagin's $A0$ algorithm, assuming that the component protocols are correct. Note that any two parties can collect the random shares of scores; the only criteria is that all parties must trust that these two will not collude to violate privacy.

1.1 Finding the kth element

Formally, the remaining issue is as follows: There is a list of n elements, $e_1, \ldots, e_n$. We want to find the score separating the kth largest element in the list from the $k+$ 1st. Specifically, there are two parties A and B having n elements ($a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ respectively) such that $\forall i, a_i + b_i = d_i \pmod F$.

Aggarwal et. al solved the problem for horizontally partitioned data[1], i.e., two parties A and B have n_a and n_b items ($a_1, \ldots, a_{n_a}$ and $b_1, \ldots, b_{n_b}$ respectively). Our method for vertically partitioned data shares the same basic idea: a binary search for an appropriate threshold score. Use of secure comparisons enables the search to proceed while revealing only if the threshold has been found.

The key idea is that as long as we know the range of the elements (the field F), binary search can be used to identify the value of kth element. The algorithm begins with an initial guess (say $F/2$). For every element, the two sites run a secure comparison protocol that compares the guess to the sum of the random shares with the sites (as in the final step of the main algorithm described above.) The output for each site are also random shares of 1 (if the element is greater) or 0 (if the element is equal or smaller), e.g., for a 1, the first site gets a random r ($0 \leq r < F$) and the second $r + 1$ (or, by random choice, vice-versa).

Once all the elements are compared, the two sites can independently sum up their shares $\pmod F$ and run another secure comparison to find out if the total of these sums is greater than k; if so, the estimate is too high. (At the same time, a comparison is done returning shares of 1 if the score is less than the guess; if the sum of these exceeds k, the guess is too low.) Based on the results, the guess is raised or lowered. Within $\log |F|$ rounds a threshold is found.

2. Summary

The primary contribution of this paper is a secure method for doing top-k selection from vertically partitioned data. This has particular relevance to privacy-sensitive searches, and meshes well with privacy policies such as k-anonymity. We have demonstrated how secure primitives from the literature can be composed with efficient query processing algorithms, with the result having provable security properties. There remain many open problems in developing secure solutions based on efficient non-secure query processing algorithms. The paper also shows a trade-off between efficiency and disclosure (i.e., the candidate object set.) It is worth exploring whether one could have a suite of algorithms (or a configurable algorithm) to optimize these tradeoffs, e.g., algorithms that guarantee k-anonymity with efficiency based on the choice of k rather than the (often expensive) guarantees of Secure Multiparty Computation.

References

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the k^{th}-ranked element. In *Proceedings of IACR Eurocrypt (EUROCRYPT04)*, Interlaken, Switzerland, May 2-6 2004.

[2] S. Chaudhuri and L. Gravano. Evaluating top-*k* selection queries. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *Proceedings of 25th International Conference on Very Large Data Bases*, pages 397–410, Edinburgh, Scotland, Sept. 7-10 1999. VLDB, Morgan Kaufmann.

[3] R. Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, pages 83–99, 1999. (Special issue for selected papers from the 1996 ACM Symposium on Principles of Database Systems).

[4] M. Kantarcıoğlu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1026–1037, Sept. 2004.

[5] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: *k*-anonymity and its enforcement through generalization and suppression. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1998.

[6] J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, to appear.

[7] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.