A New Privacy-Preserving Distributed k-Clustering Algorithm

Geetha Jagannathan* Krishnan Pillaipakkamnatt† Rebecca N. Wright*

Abstract

We present a simple I/O-efficient k-clustering algorithm that was designed with the goal of enabling a privacy-preserving version of the algorithm. Our experiments show that this algorithm produces cluster centers that are, on average, more accurate than the ones produced by the well known iterative k-means algorithm. We use our new algorithm as the basis for a communication-efficient privacy-preserving k-clustering protocol for databases that are horizontally partitioned between two parties. Unlike existing privacy-preserving protocols based on the k-means algorithm, this protocol does not reveal intermediate candidate cluster centers.

1 Introduction

Privacy-preserving distributed data mining allows the cooperative computation of data mining algorithms without requiring the participating organizations to reveal their individual data items to each other. Most of the privacy-preserving protocols available in the literature convert existing (distributed) data mining algorithms into privacy-preserving protocols. The resulting protocols can sometimes leak additional information [12, 5, 7].

Privacy-preserving data mining algorithms, the first of which were introduced by Agarwal and Srikant [1] and Lindell and Pinkas [8], allow parties to cooperate in the extraction of knowledge, without any party having to reveal individual data items. Since Yao's general-purpose secure circuit-evaluation protocol [14] is impractical, many secure special-purpose protocols have been developed for specific data mining problems.

Clustering is a well-studied combinatorial problem [6]. The task is to group similar items in a given data set into *clusters* with the goal of minimizing an *objective function*. The error-sum-of-squares (ESS) objective function is defined as the sum of the squares of the distances between points in the database to their nearest cluster centers. The k-clustering problem requires the partitioning of the data into k clusters with the objective of minimizing the ESS. Lloyd's (k-means) algorithm [9] for k-clustering and Ward's algorithm for hierarchical agglomerative clustering make use of the notion of ESS. Although Ward's algorithm has been observed to work well in practice, it is rather slow ($O(n^3)$) and does not scale well to large databases. Recently there have been a number of data mining algorithms (e.g., BIRCH [15] and STREAMLS [4]) designed for input that is too large to fit entirely in main memory.

In this paper, we present a simple deterministic algorithm, Recluster, for I/O-efficient k-clustering. This algorithm, which was explicitly designed with conversion to a privacy-preserving version in mind, examines each data item only once and uses only sequential access to the data. For fixed k, Recluster runs in $O(n)$ time and uses $O(\log n)$ space. Our experimental results show that Recluster is, on average, more accurate in identifying cluster centers than the k-means clustering algorithm. Although there are other clustering algorithms that improve on the k-means algorithm, this is the first for which an efficient cryptographic privacy-preserving version has been demonstrated.

We also present a privacy-preserving version of the Recluster algorithm, for two-party horizontally-partitioned databases. This protocol is communication efficient and it reveals the cluster centers (or the cluster assignments to data, if both parties desire) to both parties only at the end of the protocol. Unlike existing privacy-preserving protocols based on the k-means algorithm, this protocol does not reveal intermediate candidate cluster centers. These existing solutions can be made more secure but only at the cost of a high communication complexity. An alternate solution would be to develop privacy-preserving versions of other k-clustering algorithms [10, 15, 4]. However, these algorithms do not scale well to large databases [10], involve complicated data structures [15], or can be complicated to transform into a privacy-preserving protocol [4]. In comparison, our privacy-preserving version of Recluster is simple and communication efficient, and produces good clusters.

2 Preliminaries

Two parties, Alice and Bob, own databases $D_1 = \{d_1, \ldots, d_m\}$ and $D_2 = \{d_{m+1}, \ldots, d_n\}$, respectively. They wish to jointly compute a k-clustering of $D_1 \cup D_2$

*CS Dept, Stevens Institute of Technology, Hoboken, NJ. Supported by the NSF under Grant No. 0331584.

†CS Dept, Hofstra University, Hemstead, NY

(that is, the data is *horizontally partitioned*). Both parties learn the final k cluster centers, and nothing else. Alternatively, with additional computation and communication, each party could learn the cluster to which each of their data objects belongs.

If there were a trusted third party to whom Alice and Bob were both willing to send their data, this party could then compute the clustering and send the cluster centers to Alice and Bob. However, in many settings, there is no such party. *Secure multiparty computation* seeks protocols that can carry out the required computation without requiring a trusted third party. In this paper, we assume that Alice and Bob are *semi-honest*, meaning that they follow their protocol as specified, but may try to use the information they have learned (such as messages they receive) in order to infer information about the other party's data.

Our solution makes use of several cryptographic concepts. We say that *Alice and Bob have random shares of a value x drawn from a field F of size N* (or simply *Alice and Bob have random shares of x*) to mean that Alice knows a value $a \in F$ and Bob knows a value $b \in F$ such that $(a + b) \bmod N = x$, where a and b are uniformly random in field F. Throughout the paper, we assume that a finite field F of a sufficiently large size N is chosen such that all computations can be done in that field, and all computations throughout the remainder of the paper take place in F.

An encryption scheme is *additively homomorphic* if there is some operation $\otimes$ on encryptions such that for all cleartext values a and b, $E(a) \otimes E(b) = E(a+b)$. Our solutions make use of a semantically secure additively homomorphic encryption scheme (such as [11]). We also make use of a secure scalar product protocol [3] and Yao's circuit evaluation protocol [14] for small circuits.

3 The k-Clustering Algorithm

Our algorithm runs in the typical "divide, conquer and combine" fashion. This strategy would require us to divide the database into two equal halves, recursively produce k cluster centers from each of the halves, and then merge these $2k$ centers into the k final centers. However, we take a slightly different tack—we produce $2k$ cluster centers from each recursive call, and then merge the total of $4k$ centers thus received (by the two recursive calls at each level) into $2k$ centers. Finally, we use the same merge technique to produce the k final centers from the $2k$ clusters returned from the top-most level of the recursion tree (similar to [4]). See Figure 1.

The key step is the merging of $4k$ centers into $2k$ centers after the two recursive calls (MergeCenters). We do this by repeatedly choosing a best pair of clusters C_i and C_j for merging, and replacing them in the clustering

Algorithm Recluster

Input: Database D, Integer k
Output: Cluster centers S

1. $S' = \mathsf{RecursiveCluster}(D, 2k)$ // Produce $2k$ clusters
2. $S = \mathsf{MergeCenters}(S', k)$ // Compress to k clusters
3. Output S

Subroutine RecursiveCluster

Input: Database D, Integer k
Output: Cluster centers S

If ($|D| \leq k$) then $S = D$
Else

1. D_1 = First half of D
2. D_2 = Second half of D
3. $S_1 = \mathsf{RecursiveCluster}(D_1, k)$
4. $S_2 = \mathsf{RecursiveCluster}(D_2, k)$
5. $S = \mathsf{MergeCenters}(S_1 \cup S_2, k)$

Output S

Subroutine MergeCenters

Input: Cluster centers S, Integer k
Output: Cluster centers S, such that $|S| = k$

While ($|S| > k$)

1. Compute the merge error for all pairs of centers in S.
2. Remove from S the pair with the lowest merge error, and insert the center of the merged cluster, with its weight as the sum of the weights of the pair.

Output S

Figure 1: The Recluster Algorithm

with $C_i \cup C_j$. A best pair of clusters is one with least error. We use a variation of the notion of error defined in Ward's algorithm [13]. Let C_1 and C_2 be two clusters being considered for a merge. Let $C.\mathsf{weight}$ denote the number of objects in cluster C. In [13] the error of $C_1 \cup C_2$ is

$$\mathsf{error}_w(C_1 \cup C_2) = \frac{C_1.\mathsf{weight} * C_2.\mathsf{weight} * \mathsf{dist}^2(C_1, C_2)}{C_1.\mathsf{weight} + C_2.\mathsf{weight}},$$

where $\mathsf{dist}(C_1, C_2)$ is the distance between the centers of C_1 and C_2. Recluster defines the error as

$$\mathsf{error}_r(C_1 \cup C_2) = C_1.\mathsf{weight} * C_2.\mathsf{weight} * \mathsf{dist}^2(C_1, C_2).$$

Usually the pair of clusters chosen for merging is the same whether we use error_w or as error_r, but this is not always the case. Our experiments show that the use of error_r makes the algorithm less susceptible to noise, although the best choice of error measure