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

In *Proceedings of the 2006 SIAM International Conference on Data Mining (SDM)*, 2006

(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

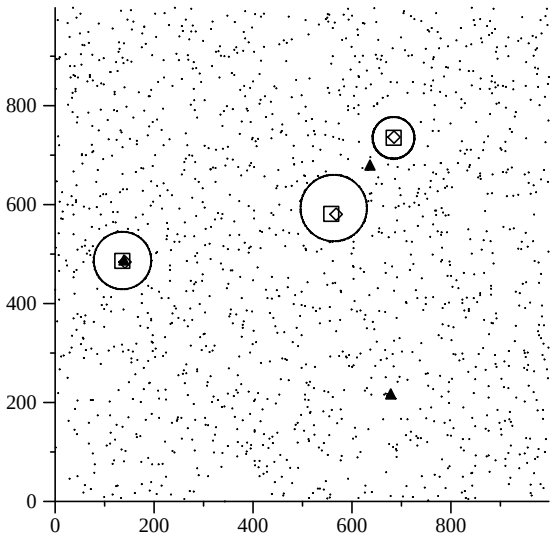

Figure 2: Cluster centers as identified by Recluster (□), the best run of k-means (◇), and the worst run of k-means (▲) on a typical noisy data set. Each circle represents a cluster of about 10,000 points.

may be application dependent. Since the MergeCenters subroutine acts as a global optimizer, our algorithm is not subject to data ordering issues.

4 Experimental Results

We ran our algorithm on a large number of synthetic data sets. Each data set uses either a uniform distribution over some intervals, or Gaussian distributions. We do not show the Gaussian results, as the performance was qualitatively indistinguishable from uniform distributions. Some of the data sets include noise at the level of 5%. In most cases, the number of clusters was relatively small (between 3 and 10); four of the data sets had 25 clusters. We tested the ability of Recluster to identify cluster centers in the presence of noise and to minimize the error sum of squares.

All algorithms were coded in Java and executed on a Dell Inspiron running Windows XP with 512MB RAM and a mobile-Pentium 3GHz processor. For each data set, we ran the k-means algorithm 10 times with different randomly chosen initial cluster centers. We compare the performance of Recluster against the best and worst performance of the k-means algorithm. We ran experiments on three types of data: random, grid, and offset grid.

Random Data. We created 50 two-dimensional data sets with 5% random noise, each of which had three clusters, with their radii and centers chosen randomly. On average, each cluster had 10,000 points. In 25 of the 50 cases, we distributed data points using a uniform distribution with each cluster; the remaining data sets used a Gaussian distribution. Results from the experiments on a typical data set are in Figure 2. Recluster does very well in identifying cluster centers, even in the presence of noise. When averaged over all data sets that used the uniform distribution, more than two runs out of the 10 runs of k-means algorithm resulted in the misidentification of cluster centers. There were, however, a few data sets (four out of these 25) in which Recluster placed the center of a cluster outside the cluster, albeit close to the cluster itself. The average running time for Recluster over all the uniform data sets (which had about 31,000 points on average) was 219 ms, and the average running time for the k-means algorithm was 532 ms.

Grid data sets. We used two data sets with 5% random noise, each consisting of 25 clusters arranged in a 5 x 5 grid, while two other data sets had 9 clusters arranged in a 3 x 3 grid. Each cluster had the same radius and the same number of uniformly distributed points (1000). Figure 3 shows the output from experiments on one of the 5 x 5 grid data sets. Recluster accurately identifies the centers of all clusters. For this data set, not even the best run of the k-means algorithm identified all 25 cluster centers. On the 9-cluster data sets, Recluster took an average of 270 ms, while k-means took an average of 420 ms. The average running time for Recluster on the 25-cluster data sets was 4530 ms, and for k-means it was 1720 ms. Recluster is slower (although far more accurate) than k-means when k gets larger because Recluster's running time is cubic in k.

Offset grid data sets. These are similar to the grid data sets, except that the clusters are randomly and slightly perturbed from grid intersection points. Recluster was quite successful in identifying all cluster centers, as was k-means on its best runs.

Minimizing the Error Sum of Squares. To check the quality of the clusters created by Recluster, we created 10 data sets, each containing between 3 and 9 clusters, with each cluster of random radius and containing a different number of points. We ran Recluster once with error_r as the error measure, and once with error_w. The ESS for the centers found by Recluster were almost identical to those the ESS for the known cluster centers. In all but two of the data sets, the ESS was the same whether we used error_r or error_w as the error function. In the two cases in which error_w was better, the increase in ESS was marginal.

5 Privacy-Preserving k-Clustering Protocol

Alice and Bob first locally use the Recluster algorithm to compute k-clusters each from their own part of the

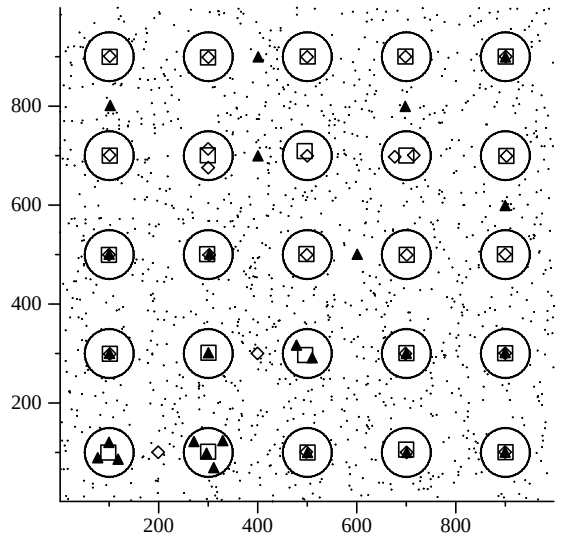

Figure 3: Cluster centers identified in a noisy grid data set by Recluster (□) and by k-means, showing the best (◇) and worst (▲) runs of the k-means algorithm.

data. Next, they randomly share their cluster centers using the permute_share protocol (see Section 5.1). All computations are done in a finite field of size N. Alice and Bob now have random shares of $2k$ cluster centers. They again use the permute_share protocol twice to prevent each party from keeping track of the cluster centers across iterations. Alice and Bob then participate in the secure merge_clusters protocol (see Section 5.2) to iteratively merge the $2k$ clusters into k clusters. At the end of m iterations, Alice and Bob obtain a random sharing of the $2k - m$ cluster centers. See Figure 4.

5.1 Secure Protocol to Permute Shares. When Alice has a vector of clusters C of the form $((c_1, w_1), \ldots, (c_k, w_k))$, this protocol helps Bob to obtain a permuted random share of the vector C. (Here each c_i is a cluster center, and w_i is its weight.) At the beginning of the protocol Alice and Bob agree on a homomorphic encryption scheme. Bob chooses a random permutation ϕ and a random vector $R = ((r_1, s_1), \ldots, (r_k, s_k))$ and outputs $\phi(C + R)$. This protocol (Figure 5) is similar to the permutation protocol introduced by Du and Atallah [2].

5.2 Secure Protocol to Merge Clusters. We now describe a protocol that securely merges m clusters into $m - 1$ cluster where the cluster centers are shared between Alice and Bob. Let $\{(c_1^A, w_1^A), \ldots, (c_m^A, w_m^A)\}$ denote Alice's share of the cluster centers and $\{(c_1^B, w_1^B), \ldots, (c_m^B, w_m^B)\}$ denote Bob's share of the

Protocol Private_k-clustering

Input: Database D of n objects, where Alice owns $(d_1, \ldots, d_m)$ and Bob owns $(d_{m+1}, \ldots, d_n)$ integer k denoting the number of clusters
Output: Assignment of cluster numbers to objects

1. Alice computes k cluster centers $(c_1, w_1), \ldots, (c_k, w_k)$ from $\{d_1, \ldots, d_m\}$ and Bob computes k cluster centers $(c_{k+1}, w_{k+1}), \ldots, (c_{2k}, w_{2k})$ from $\{d_{m+1}, \ldots, d_n\}$.
2. Alice and Bob randomly share these cluster centers with each other.

a. Bob chooses a random permutation ϕ_1 and random values $r_i, s_i, 1 \le i \le k$. Using permute_share they obtain random shares of Alice's k-cluster centers.
b. Alice chooses a random permutation ϕ_2 and random values $p_i, q_i, 1 \le i \le k$. Using permute_share they obtain random shares of Bob's k-cluster centers.
c. Alice has $(c_1^A, w_1^A), \ldots, (c_{2k}^A, w_{2k}^A)$ and Bob has $(c_1^B, w_1^B), \ldots, (c_{2k}^B, w_{2k}^B)$. Bob chooses a random permutation ϕ_3 and random values $\alpha_i, \beta_i, 1 \le i \le 2k$. Using permute_share they obtain random shares of Alice's k-cluster centers.
d. Bob adds the shares just obtained to his data. Alice chooses a random permutation ϕ_4 and random values $\gamma_i, \delta_i, 1 \le i \le 2k$. Using permute_share they obtain random shares of Bob's k-cluster centers. Alice adds the shares just obtained to her data.

3. Alice and Bob repeat the protocol to securely merge clusters k times to merge $2k$ clusters into k clusters.

Figure 4: Privacy-preserving k-clustering protocol

cluster centers, where $c_i^A = (a_{i1}^A, \ldots, a_{i\ell}^A)$, $c_i^B = (a_{i1}^B, \ldots, a_{i\ell}^B)$ for $1 \le i \le k$ and ℓ denotes the number of attributes.

Alice and Bob jointly compute the merge error for all pairs of clusters. For two clusters C_i and C_j, for $1 \le i < j \le k$, the merge error is given by $\mathsf{error}(C_i \cup C_j)$

$$= (w_i^A * w_j^A + w_i^B * w_j^B + (w_i^B * w_j^A + w_i^A * w_j^B)) + (\mathsf{dist}(C_i, C_j))^2$$

where $(\mathsf{dist}(C_i, C_j))^2$

$$= \sum_{k=1}^{\ell} (a_{ik}^A - a_{jk}^A)^2 + \sum_{k=1}^{\ell} (a_{ik}^B - a_{jk}^B)^2 + 2 \sum_{k=1}^{\ell} (a_{ik}^A - a_{jk}^A)(a_{ik}^B - a_{jk}^B)$$

The first term of the distance function is computed by Alice and the second term by Bob. Alice and Bob use a secure scalar product protocol to compute the random shares of the third term, and the random shares of $(w_i^B * w_j^A + w_i^A * w_j^B)$. We have $\mathsf{error}(C_i \cup C_j) = e_{ij}^A + e_{ij}^B$, where e_{ij}^A and e_{ij}^B are the random shares of the error known to Alice and Bob, respectively.

Alice has a $O(m^2)$-length vector (e_{ij}^A) and Bob has (e_{ij}^B) where $1 \le i < j \le k$. They securely compute

Protocol permute_share

Input: Alice has a vector of cluster centers C of the form $((c_1, w_1), \ldots, (c_k, w_k))$, Bob has a random permutation ϕ and a random vector $R = ((r_1, s_1), \ldots, (r_k, s_k))$.

Output: Alice obtains $\phi(C + R)$ as output.

1. Alice chooses a key pair (pk, sk) and sends the public key pk to Bob.

2. Alice computes the encryption of the vector C as $((E(c_1), E(w_1)), \ldots, (E(c_k), E(w_k)))$ and sends to Bob.

3. Bob uses the property of the homomorphic encryption scheme to compute $\phi((E(c_1+r_1), E(w_1+s_1)), \ldots, (E(c_k+r_k), E(w_k+s_k)))$ and sends to Alice.

4. Alice decrypts to obtain her share $\phi((c_1 + r_1, w_1 + s_1), \ldots, (c_k + r_k, w_k + s_k))$ and Bob's share is $\phi((-r_1, -s_1), \ldots, (-r_k, -s_k))$.

Figure 5: Secure protocol to permute shares

the indices i and j such that $(e^A_{ij}) + (e^B_{ij})$ is minimum using Yao's circuit evaluation [14]. Both Alice and Bob learns the indices i and j. This is efficient provided k^2 is not too large. Alice and Bob compute the merge cluster centers as random shares using a secure scalar protocol [3] and a weighted-mean protocol [7].

5.3 Efficiency and Privacy. The overall computational complexity of the privacy-preserving version of the Recluster protocol is $O(k^3\ell)$ encryptions, and $O(nk^3\ell)$ multiplications for Alice and $O(k^3\ell)$ exponentiations, $O(k^2)$ encryptions and $O(nk^3\ell)$ multiplications for Bob, where k denotes the number of clusters, n denotes the size of the database and ℓ denotes the number of attributes in the database, and c denotes the maximum number of bits for an encryption. The communication complexity is $O(k^3c\ell)$ bits, which does not depend on n. Although the computational complexity is cubic in k, for large data sets and small k, our protocol is not significantly slower than the k-means protocol for horizontally partitioned data.

Both parties compute k clusters independently from the data objects they own. They communicate with each other when they merge $2k$ clusters into k clusters. The merge_clusters protocol does the merging using secure permute_share protocol, the scalar product protocol, and the Yao's protocol. These protocols are secure. They do not leak any information. The cluster centers at the end of the merge_clusters protocol is obtained as a random shares between the two parties. When the parties like to compute the cluster centers they exchange their shares to each other. All the intermediate results are also available as random shares between the two parties. Both parties cannot learn any information from the encrypted messages that are communicated since the encryption scheme chosen is semantically secure. Hence the privacy-preserving Recluster protocol is secure and does not leak any information. The protocol achieves the same privacy as when a trusted third party is used. Earlier protocols for privacy-preserving k-clustering [12, 7, 5] are all based on the k-means algorithm. The k-means algorithm computes candidate cluster centers in an iterative fashion. All three of these protocols reveal the candidate cluster to which each object has been assigned. Our protocol does not leak any intermediate information, and is hence more private.

References

[1] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM SIGMOD*, pages 439–450, May 2000.

[2] W. Du and M. Atallah. Privacy-preserving cooperative statistical analysis. In *17th ACSAC*, pages 102–112, 2001.

[3] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On secure scalar product computation for privacy-preserving data mining. In *7th ICISC*, 2004.

[4] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[5] G. Jagannathan and R. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *11th KDD*, pages 593–599, 2005.

[6] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[7] S. Jha, L. Kruger, and P. McDaniel. Privacy preserving clustering. In *10th ESORICS*, 2005.

[8] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.

[9] S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. on Info. Theory*, 28:129–137, 1982.

[10] R. T. Ng and J. Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE TKDE*, 14(5):1003–1016, 2002.

[11] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.

[12] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *9th KDD*, 2003.

[13] J. H. Ward. Hierarchical grouping to optimize an objective function. *J. Amer. Stat. Assoc.*, 58(2):236–244, 1963.

[14] A. C. Yao. How to generate and exchange secrets. In *27th FOCS*, pages 162–167, 1986.

[15] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A new data clustering algorithm and its applications. *DMKD*, 1(2):141–182, 1997.