Secure KMeans Clustering with Fast Approximate Nearest-Neighbor Search

M.S Capstone Project

Andres Martinez Paz

June 17, 2020

Abstract

Clustering algorithms are essential for data classification in various learning systems. With a growing concern for privacy of data, there is need for fast and reliable privacy-preserving clustering algorithms. While popular algorithms, such as K-Means clustering are simple enough to be adapted into a secure setting, they are limited in usability under certain conditions, mainly dealing with high-dimensional data and applications with large number of distinct clusters. This project introduces a novel modified version of the K-Means algorithm which optimizes the scalability under the aforementioned parameters, as well as an overview on a secure version of the same algorithm. By using random data projections on high-dimensional data sets, the algorithm is able to maintain precision in the clustering, while improving heavily upon the performance of the K-Means algorithm under high-dimensions. Likewise, by replacing the original brute-force approach to the nearest-neighbor search portion of the K-Means algorithm with a fast approximate search using specialized KD Trees, the proposed algorithm also displays better complexity with respect to the number of clusters. [1][2]

1 Introduction and Motivation

The k-means algorithm is particularly popular due to its practicality and simplicity. While K-means usually performs well in low dimensions, it becomes impractical for clustering data sets in very large dimensions. With an increase of high-dimensional learning systems, there is particular need for clustering algorithms which do not suffer from the curse of dimensionality.[1] The algorithm presented in this report can be used as a good alternative to the K-means algorithm due to its ability to handle high-dimensional data with little loss in clustering accuracy. Likewise, the asymptotic complexity of the K-means algorithm becomes ineffective in secure multi-party computation due to the additional cost in time and bandwidth of secret-sharing schemes.

This project introduces an efficient adaptation to the K-means algorithm which can be easily implemented into a secure privacy-preserving clustering of multiple parties, and which employs two avenues of optimization compared to the original version of the algorithm. Firstly, a dimensionality reduction scheme is presented in the form of a random projection of the data set to a much lower dimensional space. Likewise, an algorithm for fast approximate nearest-neighbor search is introduced in which the centroids of the algorithms are organized into a collection of kd-trees which are built on randomly selected partitioning axes. These two modifications to the original algorithm result in a clustering solution which is more efficient in high dimensions, and large number of clusters.

2 Algorithm Overview

The algorithm presented in this report follows a sequence similar to that of the K-Means algorithm, with a few modification. The algorithm can be initialized in the same manner as a K-means initialization, whether that is a random initialization, or a special seeding to improve cluster quality and aid in convergence.[2] The next step in the algorithm is a random projection meant to reduce the dimensionality of the data set. This is done by generating random vectors and applying orthogonal projections to each of the data points. More detail will be given on the projection in the next section.[1]

Once the centers have been initialized, the K-means algorithm applies Lloyd iterations, which involve finding the nearest center for each of the data points and assigning the point to that particular center.[1] The search in this case is done in brute-force fashion, checking every single center for every point. In the proposed algorithm the nearest-neighbor search is replaced by a fast approximate search done through a special implementation of KD-Trees with a simplified search function, and building on random projection axes. Again, more information will be given further in this report. After each data point has been assigned to its nearest center, both algorithms follow the same sequence. For each of the centers, we compute a new center of mass for the cluster by taking the average between all assigned points. Once the new centers are found, the algorithms check for a stopping condition by comparing the distance between the new and old centers against some tolerance parameter to determine if the center has converged, in which case the algorithm terminates. After the centers have been found, one can then query the program for predictions of data points. The only difference between algorithms regarding this step is the projection of the queried point needed for the modified version of the algorithm. Otherwise, both algorithms simply perform a brute-force approach to find the nearest cluster.

3 Random Projection

In order to reduce the dimensionality of the data we can utilize randomly generated vectors to project the data set into a lower dimensional space. While this step is not required for the clustering algorithm, including it can greatly increase the performance of the algorithm, while maintaining the overall accuracy of the resulting clusters.

To perform the projection we first randomly generate d' vectors of d dimensions, where d is the number of dimensions of our data set and d' is the desired number of dimensions of the projected data set. We then collect the new coordinates of each data point projection by taking the magnitude of the orthogonal projection of said data point unto each of our randomly generated vectors. We know that the length of the projection of some vector A unto another vector B is equal to the product of the length of vector A and the cosine of the angle between vectors A and B. Likewise, we know that the dot product of vector A and vector B is equal to the product of both lengths $|A|$ and $|B|$, and the cosine of their angle, thus each coordinate for the data point projection can be obtained by computing the dot product of the data point with each projection vector and dividing the result by the length of said projection vector.

We simply perform the aforementioned computation for all projection vectors and the collected coordinates make up the projection of the data point. We can analyze the complexity of a single projection by considering that the dot product, which is between two vectors of d dimensions is computed with complexity $O(d)$, for a single data point, we perform this operation for each of the d' projection vectors, thus we project a single data point with complexity $O(dd')$. Therefore, the complexity of projecting the entire data set unto a lower dimensional space is $O(ndd')$, where n is the number of points in our data set. We can see that this projection is actually much more efficient when the desired number of dimensions in the projected space is low. However, we must also consider the accuracy of the clustering when choosing the number of dimensions for the projected data set, since a projection to a higher di-

mensional space will result in higher accuracy.

4 Nearest-Neighbor Search

Perhaps the most important part of the algorithm presented in this report is that of the nearest-neighbor search. The original K-means algorithm relies on a brute-force approach, computing the distance between each data point to each of the k centers, with a complexity $O(nkd)$, where d is the number of dimensions of the data set. Instead the approach used for this algorithm is that of building several kd-trees to organize our k centers with a simplified search algorithm in order to find candidate nearest-neighbors from which we perform a smaller brute-force approach. We build a single kd-tree by recursively partitioning the data points at each node based on a randomly chosen axis and performing a sorting of the points on said axis, then splitting the data set at the midpoint. At each node we set the midpoint as the value and build the children subtrees with each half of the data set. We consider the sorting to happen in $O(klogk)$ complexity, thus the complexity of a tree build is $O(k^2log(k))$.

The nearest-neighbor search for a single tree happens by simply traversing the tree from root to leaf, at each node choosing the next node based on whether the coordinate of the queried data point at the partitioning dimension is less than or greater than the coordinate of the midpoint chosen for the node. At each visited node, we compute the distance between the midpoint and the queried data point, keeping track of the shortest distance as we traverse the tree, simply returning the shortest distance when we arrive at a leaf. This search occurs in $O(dlog(k))$ since the distance computation occurs in $O(d)$, and the traversal is at most of $log(k)$ nodes.

We can clearly see that the search will not guarantee finding the true nearest-neighbor, but rather an approximate nearest-neighbor. In order to increase the effectiveness of the search, we build a certain number T of trees, thus we perform the search on each of the T trees to obtain candidate nearest-neighbors. Lastly, we perform a brute-force comparison between these T candidates and return the shortest distance between all. Thus, this nearest-neighbor search scheme has complexity $O(k^2log(k) + dlog(k))$. However, we only need to build the kd trees once to query every data point in our data set, thus the build complexity is amortized. Therefore, the overall complexity of assigning each data point to its nearest center happens in $O(k^2log(k)) + O(Tndlog(k)) + O(Tnd)$. We can see that this approach scales better as we increase the number of clusters compared to the traditional k-means algorithm. However, while the traditional k-means guarantees a true nearest-neighbor, this approach results in an approximate nearest-neighbor, we can thus get a more approximate search by increasing the number T of trees used.

5 Secure Setting

A secure privacy-preserving version of the algorithm is presented next. The version of the algorithm follows the previously outlined processes, with few modifications. In the two-party implementation, each player privately inputs their data set. Subsequently, the initialization of the k centers occurs in a secret-sharing scheme, by coalescing the data sets into a single secret-shared data set, performing a secret shuffle on this data set, and choosing k points at random, as centers. These centers are organized into the previously mentioned kd-trees which are to be modified to hold private center information.

The search algorithm for the functions is also modified to protect from leaking information through memory access patterns. In order to hide the tree traversal patterns, the search function occurs as a linear traversal, simply signaling children nodes for computation. As we traverse the tree in a Breadth-First fashion we check the node to see if has been marked for computation, if so we follow the same logic to signal one of its children and continue the algorithm in the same fashion. While intuitively we are removing the benefits of a logarithmic search of a tree through a linear traversal, the actual benefit lies in the reduced number of distance computations performed during the search. While the traversal indeed occurs with complexity $O(k)$, the

actual number of distance computations performed are $log(k)$, thus with this scheme, a single search operation is done with complexity $O(k + dlog(k))$ which is much better than the brute force search of complexity $O(kd)$.

While the actual implementation of the secure version of the algorithm is outside the scope of this project, this section serves to give an outline of the design for the secure algorithm. Further work is required to finalize its implementation, and evaluate it's efficiency and security guarantees.

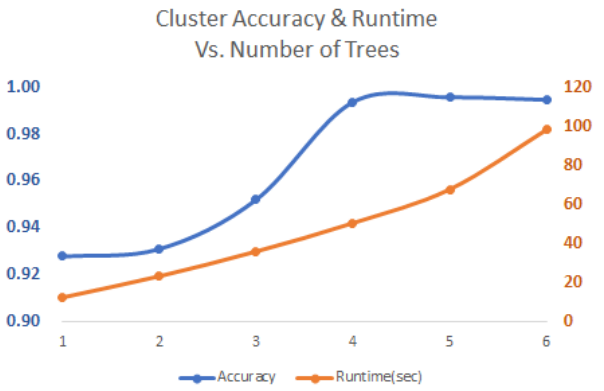

Figure 1: Clustering accuracy and runtime as number of trees used for nearest-neighbor search increases for k=20, n=50,000, dim=10, no dimensional reduction

6 Methodology

In order to evaluate the efficiency and practicality of the algorithm, a particular methodology was selected. The performance of the algorithm was directly compared to the k-means algorithm by generating synthetic data sets of varying dimensionality and number of clusters, and running both algorithms on the same data sets in an insecure setting. Generating the data sets using Scikit Learn's make_blob function returns the desired data set with ground truth clustering labelling. To compare both algorithms' scalability we measure the total runtime of each algorithm under different parameters. Likewise, we compare each algorithm's clustering accuracy by computing the centers of all clusters for both algorithms, as well as for the ground truth labelling. Then using a stable matching algorithm, a mapping is generated to identify the equivalent cluster labels given by the different algorithms. To compute an accuracy metric, we can iterate over the data set, and using the mapping from the stable matching, check if a point was assigned to the appropriate ground truth cluster by each of the algorithms. The accuracy metric represents the percentage of points correctly assigned to their cluster for each algorithm.

7 Results

Given the large number of parameters involved in the suggested algorithm, the results shown only reflect the relevant information of the algorithm, such as its scalability as the number of clusters increases, and as the dimensionality of the data increases, comparing both scenarios to the k-means algorithm. Likewise, various trials were ran evaluating the accuracy of the clustering as the projected dimensionality increases. and as the number of KD-trees used for nearest-neighbor search increases. Finally, a direct comparison is drawn between the accuracy of the K-means algorithm, and the accuracy given through different parameters of the modified version of the algorithm.

The results, shown in figures 1 thru 5 show promising behavior in the modified version of the algorithm. As expected from the asymptotic analysis of each algorithm, the scalability of the proposed algorithm with respect to the number of clusters and the number of dimensions in the data set is much better than that of the K-means algorithm. While the accuracy of the final clustering in the modified algorithm depends on various factors, a combination of parameters can be found which maximizes accuracy, while ensuring a practical, scalable algorithm. The results clearly show that for scenarios in which the data is highly dimensional, and the number of clusters in the data set is high, the proposed algorithm is

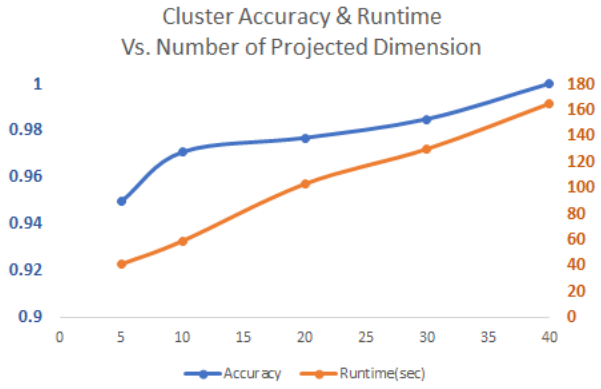

Figure 2: Clustering accuracy and runtime as number of projected dimensions increases for k=20, n=50,000, dim=10, trees=4

a better solution than the K-means algorithm.

8 Conclusion

With a growing need for reliable high-dimensional learning, there comes also a need for efficient and practical clustering algorithms which can handle high-dimensional data in a secure manner. The proposed algorithm implemented in this project meets the requirements for such a solution. While the algorithm cannot offer the accuracy guarantees given by K-means, the parametric scalability makes it a good option for fast, approximate clustering. Specifically, the algorithm shows its most promise under high-dimensional data across large number of distinct clusters. While the results are overall favorable, the algorithm does rely heavily—as do other clustering algorithms—on an efficient initialization which facilitates convergence of the centers. While the implementation and evaluation of the secure version of the algorithm was not finalized for this report, the overview given in the Secure Setting section opens the way for a simple implementation of the practical algorithm hereby introduced.

References

[1] Imad Dabura. K-means clustering: Algorithm, applications, evaluation methods, and drawbacks. *Towards Data Science. Web*, September 2017.

[2] Satyam Kumar. Understanding k-means, k-means++ and, k-medoids clustering algorithms. *Towards Data Science. Web*, June 2017.

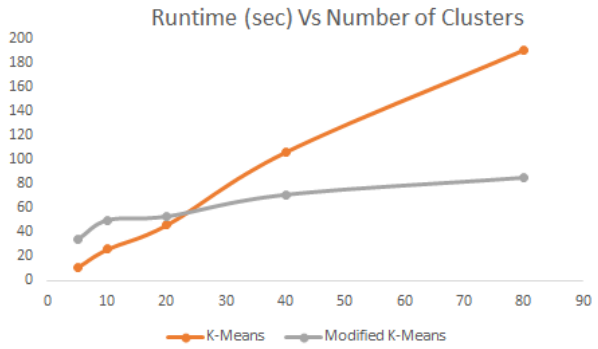

Figure 3: Algorithm runtime as number of clusters increases for both algorithms for n=50,000, dim=10, no projection, trees=4

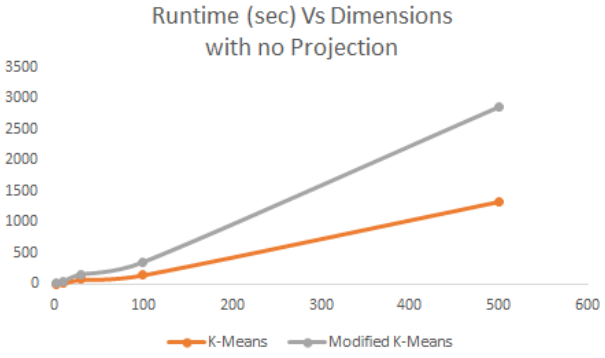

Figure 4: Algorithm runtime as dimensionality of data set increases for both algorithms for k=20, n=50,000, no projection, trees=4

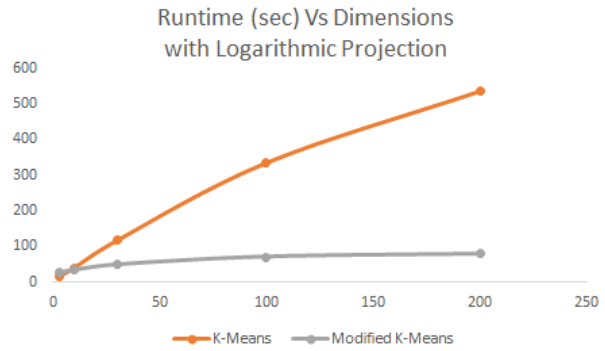

Figure 5: Algorithm runtime as dimensionality of data set increases for both algorithms for k=20, n=50,000, projected dimensions = log(d), trees=4