Privacy Preserving Two Party K Means Clustering

Vinayak Kothari Anirudh Kaushik

Abstract

Clustering is widely known problem in field of machine learning and it is solved upto to a great extent. Affinity Propagation, Agglomerative Clustering, K-Means are few examples of clustering algorithms. K Means is widely used in crime-prone areas, customer segmentation, insurance fraud detection. In all applications we find that user private data is at stake which can be misused if landed in wrong hands.

1 Introduction

K Means clustering creates cluster based on elements which are nearest to each other, it is widely used in field of image segmentation, document clustering, Customer Segmentation. But when multiple parties have to collaborate for clustering and none of them want to share their private data, they indulge in Privacy Preserving K Means, due to which complexity of problem increases and an efficient way to cluster is required, we are implementing Two Party Privacy Preserving K Means clustering. Cluster is represented by the Centroid of that Cluster which is the mean of all the elements' position in the cluster.

2 Theory

We have used the CRYPTEN [1] library for secure computation functionality. Here we describe some of CRYPTEN's secure arithmetic functions that we have used in our project -

- **Addition**

  We have used the additive sharing scheme to hide the values held by 2 parties. Assuming two parties Alice and Bob have values x and y, each party gets shares [x] and [y], say [x1] [y1] and [x2] [y2], such that it is only possible to reveal the values if and only if both parties agree to reveal their shares. In such a scheme, secure addition can be performed locally such that Alice computes [x1] + [y1] and Bob computes [x2] + [y2] and they add these shares together to find the result of x + y.

- **Subtraction**

  Subtraction is computed in a similar way to addition under additive secret sharing scheme. Alice and Bob can locally compute the shares of [x1] - [y1] and [x2] - [y2] and then add them together to get the value of x - y.

- **Multiplication**

  Using Crypten, secure multiplication of two values held by two parties, say Alice and Bob is done with the help of beaver's triples.

  First a triplet (a,b,c) is generated in such a way that c = a * b

  Let's assume, Alice has the value x and Bob has the value y, and they wish to compute z = x * y, without revealing each other's values. Furthermore, let's have u = x + y and v = x - y (this can be locally computed by Alice and Bob, assuming that x and y have been additive shared among them.)

  Then each party has a share of u and v, denoted by [u] and [v] respectively.

  Each party is, therefore, able to compute locally [d] = [u] - [a] and [e] = [v] - [b], where [a] and [b] are the shares of beaver's triples.

  Values of d and e are reconstructed, thereby making them publicly available.

  Again, the parties locally calculate [u*v] = d * [a] + e * [b] + [c] +d * e, and thus they have shares of the multiplication.

- **Square**

  Square is calculated in a similar way to multiplication. In this case random sharing of [r] and [r2] are generated, in such a way that [r2] = [r * r]. The value [x](whose square is to be calculated) is additively hidden via [r] and publicly known [e] is defined such that [e] = [x] - [r]

  Then the result of the square of (x), say z, is calculated as

  [z] = [r2] + 2 * e * [r] + e * e.

- **Division**

  In k means clustering, division operation is used during the calculation of the centroid of clusters, in this case each party already knows the number of elements present in the cluster, hence we only require to divide the distance by the publicly known cluster size.

3 Dataset and Threat Model

- **Dataset Used:** We have taken Mall Customer data as our input file which has 200 tuple and 5 attributes **[id, name, age, salary, expenses]**. We are considering salary and expenses as our dimension and we will setting K =5 which will identify our customer as

  1. **Miser - High income , low expenses**
  2. **General - Medium Income ,Medium expenses**
  3. **Target - High income , high expenses**
  4. **Careful - Low income, Low expenditure**
  5. **Spendthrift - Low income , high expenditure**

- **Threat Model:** We have considered the 2 party semi-honest setting for our project. While performing k-means clustering, we want to keep each party's data to be secret. This will ensure that all parties can know the size of the cluster, but the exact location of elements in the cluster will not be revealed during any stage.

3.1 Random initialization trap in K Means

One problem with K means is that with random initialization of clusters. Let's consider a dataset of points p1:{1,1} , p2:{1,2}, p3:{4,1}, p4:{4,2} .Select k =2 and initialize cluster to be k1:{p1}, k2:{p2} then p1 and p3 will be assigned cluster k1 , p2 and p4 will be assigned cluster k2 based on euclidean distance of points. Now comes the problem in the next iteration where we find that p3 will still be assigned to k1 and p4 will be assigned to k2 because the euclidean distance between p3 and k1 is less than p3 and k2 , and distance between p4 and k2 is less than p4 and k1. But from **Figure 1** we can see that the actual cluster formed should be k1 :{p1,p2} , k2: {p3,p4}, but the formed cluster are k1 :{p1,p3} , k2: {p2,p4} in **Figure 2** [2] .

So we solve this problem by initializing clusters in such a way, that they should be at a distance farthest from the dataset point of the initialized cluster, which will spread clusters evenly as in the **Figure 3(a)**. With this algorithm our initial cluster of above dataset would be at p1, p3 and then we apply vanilla K Means clustering which will result in more better clustering with k1:{p1,p2} , k2:{p3,p4} as in **Figure 1**

4 Implementation Results

K means is an unsupervised Machine Learning algorithm, so we compare our 2 party privacy preserving K Means clustering algorithm with the vanilla K Means clustering algorithm which will give us accurate results of how our method performed. First results are computed with random cluster initialization and then with improved cluster assignment.

A problem we faced with precision of tensor was that, when divided by scalar value and secret shared among the parties, the precision bits were lost when we decrypted the tensor and this actually created

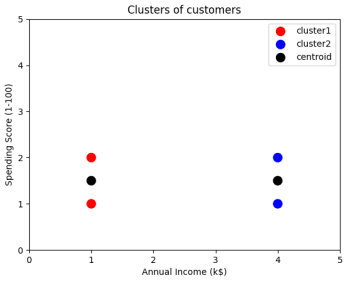


Figure 1: Correct Cluster

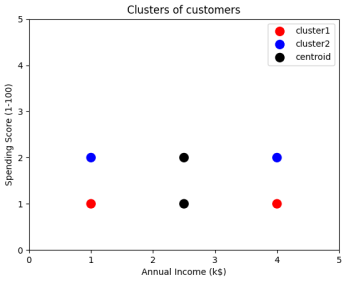
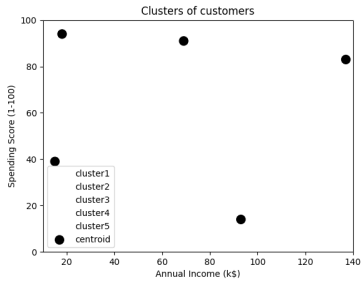
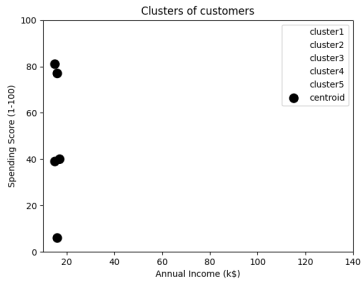


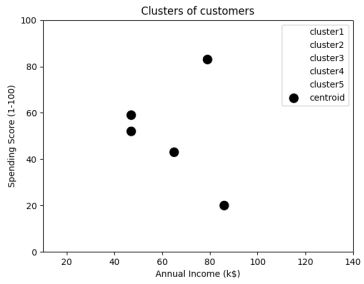
Figure 2: Incorrect Cluster



(a) Improved initialization of cluster with 200 point in dataset



(b) Random initialization of cluster with 200 point in dataset



(c) Another Random initialization of cluster with 4 point in dataset

Figure 3: Initialization of clusters

a problem in continuous operations of sum, divide, square, add, max, min which reduces the precision drastically.We observed that loss of precision is tolerable till two parties , but if we increase number of parties to three our answers diverge from minimal point.

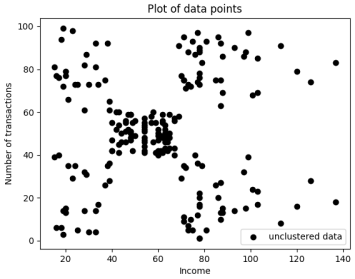
We can see from the figure that in the case of random initialization of clusters it took average 10 epoch to converge while in case of improved cluster initialization it took around 8 epoch to converge. With random initialization another problem was that sometimes code did not even converge **Figure 4(c)** but with improved methods it converged every time with the same epoch count. We ran our experiments for 20 iterations to calculate the probability, which shows our algorithm is a good improvement over random initialization. And this same trend can be observed in 2 party k means also where number of epoch over random initialization is 12 and for improved initialization is only 9, this increase can be attributed to precision loss factor while computing arithmetic functions in multi party setting.

But time computation for multi party k means is another important factor in which per epoch time taken by 1 party means is 250ms with Numpy library but for 2 party k means with use of CrypTen library is 1 minutes which needs to be improved which makes it far from deployment ready.

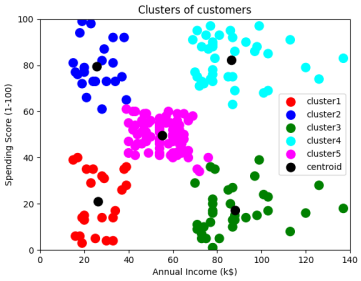
From our experiments we found out that evenly spread out clusters contract easily but spreading of clusters from one concentrated initialized area is very difficult and requires more iterations to converge as seen in the **Figure 3(b)**. So K Mean convergence is highly dependent on the initial cluster initialization. Source code is available at **GitHub link**.

5 Conclusion and Future Work

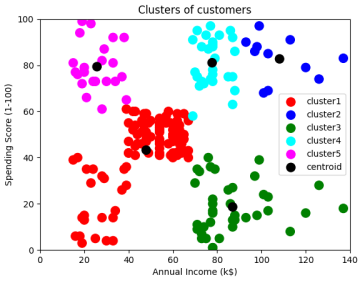
From above implementation we conclude that when number of parties < 3 our clustering performed as good as vanilla k means with miss classification rate of $< 2\%$ but when we increase number of clusters and the cluster size increases then the precision goes down which increases noise in our computation. CrpyTen library which is still under development so it is still error prone in these scenarios. In future we would like to improve efficiency of k means with parties greater that 2, and reduce noise in elements by considering 2



(a) Data set



(b) Every time these cluster formed with Improved initialization clustering



(c) Some times incorrect clusters are formed with random initialization

Figure 4: K Means final result

	Vanilla K Means	Improved K means	Vanilla 2 Party K means	Improved 2 party K Means
Time taken for 1 epoch(sec)	0.25	0.26	60	62
Average epochs to converge	10	8	12	9
Accuracy(%)	100	99	100	99
Convergence Probability	0.7	1	0.65	1

Table 1: Results for our algorithm. Note: Accuracy is calculated only for proper clusters **Figure 4:(b)**

parties at a time for computation and not by taking all parties at the same time, for example in the case for finding minimum element in the array in case of 4 parties, first minimum will be found out between 1^{st} and 2^{nd} party then this minimum will be computed with 3^{rd} and 4^{th} party. We hope this will give us better results as compared to now.

References

[1] D. Gunning, A. Hannun, M. Ibrahim, B. Knott, L. van der Maaten, V. Reis, S. Sengupta, S. Venkataraman, and X. Zhou, "Crypten: A new research tool for secure machine learning with pytorch," 2019.

[2] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," tech. rep., Stanford, 2006.

[3] "K Means algorithm."

6 Appendix

6.1 Elbow Method

We are getting K = 5 using the elbow method [3] which we run on K Mean algorithm , in this algorithm graph is formed between summation of all average distance value inside cluster also called loss value and different K value chosen. Loss is calculated after K means convergence is achieved. Optimal K is value where average distances reduction value is steep while increasing K and when reduction slows down or curve flattens like at k=6 and more, this means we are over fitting our algorithm, and finally average distance will be 0 where k= number of input data point and each input data point is assigned 1 cluster.

$$loss = \sum_{i=1}^{k} \frac{\sum_{j=0}^{Count(C[i])} C[i] - C[i][j]}{Count(C[i])}$$

C[i] - Cluster at index i C[i][j] - Elements belonging to that cluster Count(x) - Number of elements in x

6.2 Algorithm for Improved K Means

This is our modified algorithm for K means cluster initialization in which we improve allocation of initial clusters but spreading out them based on distances from already assigned center's. Actual code can be check at thus line **K Means Initialization**

Algorithm 1 Improved K Means Initialization Code. It returns initial evenly spread out clusters and then we run K Means using these initial Clusters.

Require: Input$_1 \ldots Input_N$

```
function CLUSTER INITIALIZATION(Input,K)
    Cluster[1] ← any random value from input
    for k ← 2 to K do
        Distance ←[]
        for point ← Input[0] to Input[N] do
            Distance ← Append(min(EuclideanDistance(point, Cluster[1 : k − 1]))
        index ← maxIndex(Distance)
        Cluster[k] ← Append(Input[index])
    return Cluster
```