DBSCAN Clustering Algorithm Based on Locality Sensitive Hashing

Ye Shiqiu, Zhu Qingsheng
College of Computer Science, Chongqing University, Chongqing 400044, China
862911491@qq.com

Abstract. The traditional DBSCAN clustering algorithm runs less efficiently on large data sets and high dimensional data sets. Aiming at the disadvantages of this, a core point selection algorithm and isolated points detection algorithm based on locality sensitive hashing is proposed. After that a traditional DBSCAN algorithm is run on the core points. Finally, the remaining points are assigned to the same category of the core points which both are in the same sub-cluster. The experiment results show that the proposed algorithm maintains a high correct rate on synthetic and real data sets and the efficiency is greatly improved.

1. Introduction
Clustering is an unsupervised learning method in the field of data mining. Its main task is to assign each object into several groups according to some inherent laws without prior knowledge [1]. DBSCAN[2] is one of the most widely used density-based clustering algorithms. The core idea of DBSCAN is to cluster the objects in the dataset by gradually expanding the search radius of the high-density points. However, the computational complexity of the DBSCAN on high-dimensional datasets will be high. The time complexity of DBSCAN is $O(n^2)$ without index because object has to compare to each other to find its neighbors. In order to improve the efficiency of searching the nearest neighbors, some index structures are created in advance usually, such as kd-tree, R-tree, SR-tree, PR-tree [3]. When the dimension of dataset is greater than 10, using these data structures to find neighbors is even less efficient than linear search[4].

In order to solve this problem, Indyk and Motwani proposed a high-dimensional index structure called Locality Sensitive Hashing (LSH). Its principle is to use a set of specific hash functions to build a hash table, so that under certain similarity measure conditions, the probability of similar objects mapping to the same slot in the hash table is greater than that of those with lower similarity. Therefore, the hash table contains a lot of structural information of the whole dataset. The core point selection algorithm and isolated points selection algorithm proposed in this paper are inspired by this idea. It can be concluded as follow:

1. If two objects are mapped to the same slot many times after hash mapping, the probability that the two objects are neighbors will be very high.
2. If there are almost no objects in the same slot with an object after a hash mapping, then the probability of this object is an isolated point is very high.

2. Locality Sensitive Hashing

2.1. Introduction of Locality Sensitive Hashing
Finding one or multiple data that is the most similar to a certain data from a large collection of high-dimensional data is the key and difficulty of many questions. Indyk et al. proposed an algorithm called locality sensitive hashing, which was mainly used to solve the problem of neighborhood retrieval. A lot of variant models of LSH [5] have been proposed and applied on many field. The basic idea of LSH is that after two adjacent data points in the original data space are transformed by the same mapping or projection, the probability that the two data points are still adjacent in the new data space is very high, while the probability that the non-adjacent data points are mapped to the same slot is very small. We divide the original dataset into several sub-sets by hash function mapping transformation, and the data in each sub-set are adjacent and the number of elements in the sub-set is small. Therefore, the problem of finding adjacent elements in a large set is transformed into the problem of finding adjacent elements in a very small set, which reduces the search volume greatly.

For a set of points based on the distance function \( D \) (eg Euclidean distance, Hamming distance, Manhattan distance, etc.), we have the following definition:

**Definition 1 (Locality Sensitive Hashing):** a family \( H = \{ h : S \rightarrow U \} \) of functions is called \((r_1, r_2, p_1, p_2)\)-sensitive if for any two point \( p, q \in R^d \):

- if \( d(p, q) \leq r_1 \) then \( \Pr_h[h(q) = h(p)] \geq p_1 \)
- if \( d(p, q) \geq r_2 \) then \( \Pr_h[h(q) = h(p)] \leq p_2 \)

where \( d(p,q) \) is the distance of point \( p \) and \( q \). Two restrictions should be satisfied for the locality sensitive hashing scheme: \( p_1 > p_2 \) and \( r_1 < r_2 \). In order to increase the difference between \( p_1 \) and \( p_2 \), the LSH scheme will apply multiple hash functions generally. For a given integer \( k \), \( g(p) = (h_i(p), h_2(p), ..., h_k(p)) \) is generated by family \( G = \{ g : S \rightarrow U^k \} \) for which \( h_i \in H \).

Therefore a family \( G \) has two properties:

- if \( d(p, q) \leq r_i \) then \( \Pr_{g_i}[g(q) = g(p)] \geq p_i \)
- if \( d(p, q) \geq r_i \) then \( \Pr_{g_i}[g(q) = g(p)] \leq p_i \)

2.2. \( p \)-stable distributions

In probability theory, a distribution is said to be stable if a linear combination of two independent random variables with this distribution has the same distribution, up to location and scale parameters.

**Definition 2 (p-stable distributions):** if there exists \( p \geq 0 \) such that for any \( n \) real numbers \( v_1, ..., v_n \) and independent and identically distributed variables \( X_1, ..., X_n \) with distribution \( D \), the random variable \( \sum_i v_i X_i \) has a random variable with distribution \( D \), then a distribution \( D \) over \( R \) is called \( p \)-stable

Stable distributions exist for any \( p \in (0,2] \). Locality sensitive hashing scheme based on \( p \)-stable distributions is suitable for the \( l_p(p \in (0,2]) \) norm. Each hash functions is defined as follows:

\[
h_{a,b}(v) = \frac{av + b}{W} \quad 2-1
\]

where \( W \) represents the size of the quantized hash bucket, \( a \) is a random variable selected from a \( p \)-stable distribution independently, \( b \) is a real number selected uniformly in the range \([0,W]\) which means an offset.

For any two vectors \( v_1, v_2 \), the distance between them is \( \sigma = \|v_1 - v_2\|^p \). Then we have

\[
p(\sigma) = \Pr_{a,b}[h_{a,b}(v_1) = h_{a,b}(v_2)]
= \int_0^\sigma \int_0^1 \frac{1}{\sigma} (1 - \frac{t}{\sigma}) dt
= \int_0^\sigma \sigma(1 - \frac{\sigma}{r}) = \frac{1}{2} \sigma(\sigma + 1)
\]

\[2-2\]
Where $f_p(t)$ means probability density function of p-stable distribution. The formula shows that the possibility of conflict increases as $\sigma = ||v_i - v_j||$ decreases for a fixed parameter. It means that the smaller the distance between two objects is, the greater the probability of getting the same hash value is. On the contrary, the greater the distance between two objects is, the smaller the probability of getting the same hash value is.

2.3. Searching of Nearest Neighbors
Locality Sensitive Hashing algorithm based on p-stable distribution is one of the famous LSH algorithm. P-stable distribution has been used to lots of variant models of LSH. Neighbor queries are divided into the following two processes which needs two parameter, the number of hash tables L and the number of hash functions h in each hash table. The size of the quantized hash bucket W is selected by calculating the average distance between objects and their nearest neighbors, which are chosen randomly from the dataset.

2.3.1 Pseudo code of searching the neighbours of point q
**Input:** object q, dataset D, hashing index LSHIndex  
**Output:** the candidate set of q’s neighbours  
1. function searchNN(q, D, LSHIndex)  
2. initialize candidateSet  
3. for i = 1 to L  
4. %Calculate the index of q in hashTable_i and return all points with the same index  
5. tmpset = hashTable_i(query(q))  
6. candidateSet = candidateSet ∪ tmpset  
7. end for  
8. return candidateSet  
9. end function

3. LSH-DBSCAN algorithm
The LSH-DBSCAN algorithm is mainly divided into four steps.

**Step 1:** Select points which there are almost no points in the same slot with them in the hash tables as isolated objects. This process is based on the second inspiration before.

**Step 2:** Sub-cluster partitioning algorithm is used to divide the dataset into different sub-clusters on the dataset with the isolated points removed.

**Step 3:** Selecting the core points of each sub-cluster through a certain strategy on each sub-cluster. Experiments show that a random selection of points can reach good result.

**Step 4:** Finally, the traditional DBSCAN algorithm is used to obtain the clustering result at the core point, and the remaining points are assigned the clustering results of the core points in the same sub-cluster.

As for the complexity analysis, assume that n is the number of objects in the dataset, L is denoted as the number of hash tables, h is the number of hash functions in each hash table. $T_{projection}$ is denoted as the cost for a single projection of points in the dataset. Therefore, the total cost is $nLhT_{projection}$ to build LSHIndex and the space complexity is $nL$. Searching the neighbors of object q cost $LhT_{projection}$. Thus, the time cost of searchOutliers function is $nLhT_{projection}$. The time cost of function partitionSubsets can be divided into two parts. First, it costs $nLhT_{projection}$ to query neighbor of each object q. Second calculating the conflict count of the object q with its candidate sets for in the L hash tables. Strictly speaking this is an $O(n^2)$ operation. However for the implementation of the algorithm
if an object is checked it won’t have to be executed for this function any more. And the candidate set is only a small part relative to the entire dataset. Then the core points are applied to the traditional DBSCAN algorithm and the time complexity is $O(m^2)$ where $m$ is the number of core points. At last the time complexity of assigning other points to their core point is $O(n)$  

Input: Dataset D, the parameter of function partitionSubsets $\lambda$ , the parameter of function searchOutliers outliersBound, hashing index LSHIndex.

Output: Cluster labels

1. function LSDBSCAN($\lambda$, outliersBoundary, D, LSHIndex)
   % search isolated points
2.   searchOutliers(outliersBound, D, LSHIndex)
   % partition of subclusters on data sets with outliers removed
3.   subsetsList = partitionSubsets($\lambda$, D, LSHIndex)
   % selecte core points in each subcluster
4.   corePoints = selectCorePoints(subsetsList)
5.   tmpClusters = DBSCAN(corePoints, minPts, epsilon)
6.   Clusters = assignotherPointsCluster(tmpclusters, otherPoints)
7. end function

1. function searchOutliers(outliersBoundary, D, LSHIndex)
2.   for i = 1 to size(D)
3.     tmpSet = searchNN(Pointi, D, LSHIndex)
4.     if size(tmpSet) < outliersBoundary
5.       sign all points in tmpSet as isolated points
6.   end if
7. end for
8. end function

1. function partitionSubsets($\lambda$, D, LSHIndex)
2.   initialize collisionCount = $L * \lambda$, initialize subsetsList
3.   for i = 1 to size(D)
4.     if Pointi is not checked
5.       tmpSet = searchNN(Pointi, D, LSHIndex)
6.       checkcollisionCount (tmpSet, Pointi, collisionCount)
7.     % sign all points in tmpSet
8.       check(tmpSet)
9.   end if
10.  subsetsList.add(tmpSet)
11. end for
12. return subsetsList
13. end function

4. Experimental results and analysis

Five synthetic dataset and two real dataset from UCI is selected to evaluate the performance of our proposed algorithm.

Taking data sets d6, aggregation as examples, the distribution of their core points and clustering results are shown below. Figures 4.1 to 4.2 show the distribution of the core points of the data set d6, aggregation and the clustering result of DBSCAN and LSH-DBSCAN. As the blue core points show, the core points maintain the overall structure of the original dataset well and remove most of the isolated points. The LSH-DBSCAN algorithm does not have a large speed increase when the amount of data is small. The reason is that if the number of core points is small then they can’t maintain the
structure information of the dataset so it affects the cluster result a lot. However, when the amount of data increases, the advantage of clustering by using core points is highlighted. LSH-DBSCAN can achieve better speed when the amount of data increases. Moreover, by removing some outliers and clustering with core points, the "sticky" parts between clusters can be removed. From the results of the aggregation data set, the original DBSCAN algorithm cannot distinguish the connected clusters on right. On the other hand, clustering through the core points weakens the influence of the points at the edge. From the d6 dataset, because the interval between the two "long strips" of data points is large, a few points of the two "long strip" junctions are not selected as the core points, so LSH-DBSCAN divides them into two clusters. Table 2 shows the experimental comparison results between LSH-DBSCAN and DBSCAN.

| dataset                        | number of instances | dimension | number of clusters |
|--------------------------------|---------------------|-----------|-------------------|
| d6                             | 1 400               | 2         | 4                 |
| aggregation                    | 788                 | 2         | 7                 |
| t4                             | 8 000               | 2         | 6                 |
| s1                             | 20 000              | 2         | 4                 |
| s2                             | 50 000              | 2         | 5                 |
| UCI 3D spatial network         | 28 612              | 4         | 4                 |
| UCI MoCapHandPostures          | 34 963              | 9         | 6                 |

| dataset                        | LSH-DBSCAN | DBSCAN |
|--------------------------------|------------|--------|
|                                | Avg time(s) | Accuracy | Avg time(s) | Accuracy |
| d6                             | 1.7        | 70.8%   | 1.2        | 96.8%    |
| aggregation                    | 0.9        | 97.8%   | 0.5        | 73.5%    |
| t4                             | 3.3        | 89.1%   | 3.1        | 90.3%    |
| s1                             | **4.70**   | 98.4%   | **19.10**  | 98.8%    |
| s2                             | **73.1**   | 96.3%   | **282.8**  | 97.3%    |
| UCI 3D spatial network         | **22.8**   | 67.6%   | **110.1**  | 68.8%    |
| UCI MoCapHandPostures          | **96.7**   | 50.2%   | **125.8**  | 52.6%    |

Table 1. Dataset information

![distribution of core points](image1.png)  
(a) distribution of core points

![cluster results of DBSCAN](image2.png)  
(b) cluster results of DBSCAN

![cluster results of DBSCAN LSH-DBSCAN](image3.png)  
(c) cluster results of DBSCAN LSH-DBSCAN

Fig.1  Experiment results of dataset d6
5. Conclusion

This paper proposes a LSH-based clustering algorithm. The experimental results show that the proposed algorithm can effectively improve the speed performance when the data set is large, while maintaining the good accuracy as the traditional DBSCAN algorithm. Although clustering research based on LSH has achieved some success, there are still some problems to be solved. Based on the current work, the future research work is as follows: (1) Further improve the efficiency of the algorithm so that the speed can be increased when the amount of data is small while maintaining the clustering effect. (2) Combine LSH with other clustering algorithms, such as MST [6]. (3) LSH scheme has natural and good parallelism so we can do some research on Distributed Clustering Algorithm based on Hadoop[7] and spark framework.

References

[1] Sarle W S . Finding Groups in Data: An Introduction to Cluster Analysis. by Leonard Kaufman; Peter J. Rousseau[J]. 1991.
[2] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in KDD, Portland, Oregon, USA, August 2–4, 1996, pp. 226–231
[3] L. Arge, M. De Berg, H.J. Haverkort, K. Yi, The priority R-tree: a practically efficient and worst-case optimal R-tree, in Proceedings of the 2004 ACM SIGMOD International Conference.
[4] R. Weber, H.-J. Schek, S. Blott, A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, in VLDB, New York, USA, August 24–27, 1998, pp. 194–205
[5] Indyk P., Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality[A]. Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing[C]USA: ACM Press, 1998. 604-613
[6] Wang X, Wang X L, Chen C, et al. Enhancing minimum spanning tree-based clustering by removing density-based outliers[J]. DIGITAL SIGNAL PROCESSING, 2013, 23(5):1523-1538.
[7] Fu X, Wang Y, Ge Y, et al. Research and Application of DBSCAN Algorithm Based on Hadoop Platform[M]// Pervasive Computing and the Networked World. Springer International Publishing.