Pattern Recognition Letters 32 (2011) 1572–1580

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Improving DBSCAN's execution time by using a pruning technique on bit vectors

Selim Mimaroglu *, Emin Aksehirli

Department of Computer Engineering, Bahcesehir University, Ciragan Caddesi, 34353 Besiktas, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 10 May 2010
Available online 15 June 2011
Communicated by W. Pedrycz

Keywords:
Clustering
DBSCAN
Binary methods
Pruning

ABSTRACT

Clustering is the process of assigning a set of physical or abstract objects into previously unknown groups. The goal of clustering is to group similar objects into the same clusters and dissimilar objects into different clusters. Similarities between objects are evaluated by using the attribute values of objects. There are many clustering algorithms in the literature; among them, DBSCAN is a well known density-based clustering algorithm. We improve DBSCAN's execution time performance for binary data sets and Hamming distances. We achieve considerable speed gains by using a novel pruning technique, as well as bit vectors, and binary operations. Our novel method effectively discards distant neighbors of an object and computes only the distances between an object and its possible neighbors. By discarding distant neighbors, we avoid unnecessary distance computations and use less CPU time when compared with the conventional DBSCAN algorithm. However, the accuracy of our method is identical to that of the original DBSCAN. Experimental test results on real and synthetic data sets demonstrate that, by using our pruning technique, we obtain considerably faster execution time results compared to DBSCAN.

1. Introduction

Clustering is the process of organizing objects into groups that have similar members; a distance metric is used for evaluating dissimilarity. Clustering, which has broad range of applications, is also known as unsupervised classification. The goal is to place objects that are similar to each other in the same cluster, and objects that are dissimilar in different clusters.

Clustering has a long and rich history in a variety of scientific fields (Jain, 2010). Some of the clustering algorithms partition objects into groups and are therefore known as partitional clustering algorithms. For example, *k*-means (MacQueen, 1967) is a well-known partitional clustering algorithm that divides a set of objects into *k* clusters where *k* is a user specified parameter. Another type of clustering is hierarchical clustering; agglomerative methods are well-known techniques of this type. Yet another type of clustering is density based clustering: DBSCAN (Ester et al., 1996) is a popular algorithm that can correctly cluster arbitrarily shaped data sets when provided with the right parameters.

DBSCAN is a very popular density based method that it is still commonly used as a reference to compare the accuracies of new methods, such as in (Yousri et al., 2009; Garai and Chaudhuri, 2004; Zhong et al., 2008; Liu et al., 2008). We improve DBSCAN's execution time performance by using a novel pruning technique, as well as bit vectors and binary operations.

The Hamming distance measures the degree of difference between symbols that occur in the same positions. This measure is used widely on sequences such as words, gene expressions, proteins, web clicks, and natural and social events for conducting data mining activities. The Hamming distance is also preferred in natural language processing and text searching for detecting misspellings and correcting them automatically. There are many applications of the Hamming distance; two interesting applications of robotics and compression that use the Hamming distance can be found in (Ros and Sutton, 2004; Haikonen, 2007). Some very popular edit distances and the Jaccard distance are built on top of the Hamming distance.

Continuous valued data attributes can be transformed into categorical and binary forms. *Equal Width Binning* divides the data range into equal width intervals where each interval defines a category. *Equal Frequency Binning* divides the data range into intervals that contains equal numbers of objects. Clustering methods, such as *k*-means, can be used to discretize the data as well. For discretization, supervised methods can be used as well where discretization is conducted according to data labels. In these methods, numbers of bins and clusters are determined according to the data set and the requirements of the application. Categorical attributes can be converted into binary form easily by listing all of the categorical values in the list of attributes and by representing the current value with 1, and the other values with 0.

Our paper is structured as follows: Section 2 contains related work. In Section 3, the conventional DBSCAN algorithm is presented. Section 4 introduces our improvement to DBSCAN, and Section 5 provides experimental evaluations. In the final section, we present conclusions and future directions.

* Corresponding author. Tel.: +90 212 381 05 55; fax: +90 212 381 05 50.
E-mail addresses: selim.mimaroglu@bahcesehir.edu.tr (S. Mimaroglu), emin.aksehirli@bahcesehir.edu.tr (E. Aksehirli).

0167-8655/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2011.06.003

2. Related work

In this section we provide a non-exhaustive list of algorithms that improve the execution time performance of DBSCAN using methods such as space partitioning, sampling, and reducing the number of core points. We also describe the strengths and weaknesses of these methods.

Partitioning the data set to reduce the search space and hence improve the execution time of DBSCAN is a well-known technique. To partition the data set, El-Sonbaty et al. (2004) proposes the use of CLARANS; the DBSK algorithm (Rui and Chunhong, 2008) uses *k*-means, PDBSCAN uses the statistical characteristics of the data, and a recent method proposed in (Jiang and Li, 2009) uses a one-pass clustering algorithm. Space indexing techniques such as KD-tree and R-tree are used for fast retrieval of an object's neighbors in a data set, where indexing techniques can be classified as space partitioning methods. In general, partitioning is regarded as a pre-processing step with its own requirements and limitations. Besides mechanical requirements of partitioning methods such as extra memory and extra computation time, data set dependent requirements such as determining the right input parameters makes these methods less appealing.

The execution time of DBSCAN can be reduced by using only a subset of the data set, which is known as sampling. Sampling-based DBSCAN, SDBSCAN (Zhou et al., 2000), runs DBSCAN on a randomly selected subset of objects to form clusters. Rough-DBSCAN (Viswanath and Babu, 2009) employs the well-known Leader clustering algorithm and rough set theory for sampling and then producing approximate clusters. Using low sampling rates for obtaining good execution time can degrade the accuracy substantially.

FDBSCAN (Zhou et al., 2000) and IDBSCAN (Borah and Bhattacharyya, 2004) reduce the execution time by utilizing smart core detection methods. KIDBSCAN (Tsai and Liu, 2006) locates the high-density areas using *k*-means and introduces core points with respect to density rankings. Sampling based improvement by reducing the number of core points is introduced in (Tsai and Sung, 2010). SPARROW (Folino et al., 2009) reduces the number of core points using a method that is inspired by the flocking mechanism of birds. Smart detection of core points and sampling on the core points may result in anomalies such as falsely dividing the original clusters and false identification of border points as noise.

Parallel approaches of DBSCAN are introduced in (Zhou et al., 2000; Sakellariou et al., 2001; Guo et al., 2002). Although parallel implementations may improve the execution time performance locally at each processor, combining the results into the final output is not trivial.

3. The DBSCAN algorithm

DBSCAN (Density Based Spatial Clustering of Applications with Noise), which is shown in Algorithm 1, is a simple and effective density-based clustering algorithm that can identify arbitrary shape clusters and noise (Ester et al., 1996). In DBSCAN, the following main steps are performed: (1) Label all objects as core, border, or noise points with respect to the input parameters, (2) Put an edge between all core points that are neighbors, (3) Label connected core points as a cluster, and (4) Include all of the border points within the neighborhood of a cluster into the same cluster.

An object is a core point if the number of objects with its ε radius is at least *MinPts*. A border point is not a core point, but is located within the ε radius of a core point. A noise point is an object that is neither a core point nor a border point. In Fig. 1, the number of points within the ε radius of point p is 7. In Fig. 2, p_1 is a core point, p_2 is a border point, and p_3 is noise with respect to ε and *MinPts* = 7.

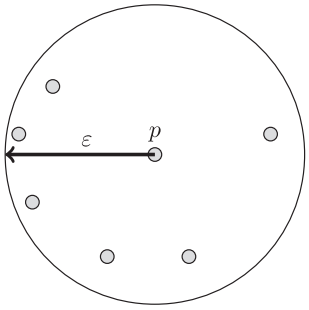

Fig. 1. Center based density in DBSCAN.

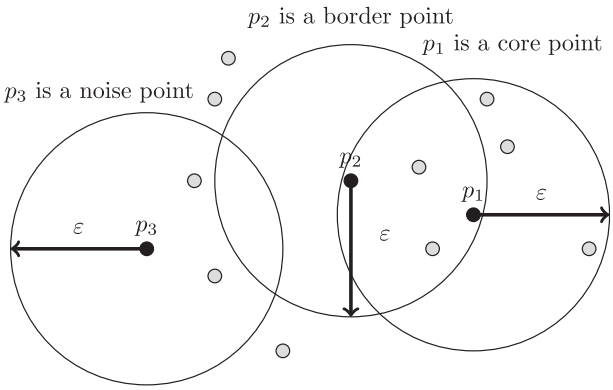

Fig. 2. A data set labeled with respect to ε and *MinPts* = 7.

Algorithm 1. DBSCAN algorithm

Input: *D*: data set, ε: radius, *MinPts*: minimum number of points
Output: Π: Clustering

```
1: clusterId = 0
2: for all unvisited point p ∈ D do
3:    mark p as visited
4:    N = getNeighbors(p,ε)
5:    if sizeof(N) < MinPts then
6:       mark p as noise point
7:    else
8:       clusterId + +
9:       add p to cluster clusterId
10:      for all point p′ ∈ N do
11:         if p′ is not visited then
12:            mark p′ as visited
13:            N′ = getNeighbors(p′,ε)
14:            if sizeof(N′) ⩾ MinPts then
15:               N = N ∪ N′
16            if p′ does not belong to a cluster then
17:               add p′ to cluster clusterId
18: return Π
```

4. Binary approach for DBSCAN

The DBSCAN algorithm can find arbitrary shape clusters in a data set accurately when provided with the correct ε and *MinPts* values. The time complexity of the DBSCAN algorithm is $O(n^2)$, where n is the number of data points. If the distance matrix is stored, the space complexity is also $O(n^2)$. However, it is possible to compute the distance matrix on the fly and to reduce the space