A Neighborhood-Based Clustering by Means of the Triangle Inequality

Marzena Kryszkiewicz and Piotr Lasek

Institute of Computer Science, Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warsaw, Poland
{mkr,p.lasek}@ii.pw.edu.pl

Abstract. Grouping data into meaningful clusters is an important task of both artificial intelligence and data mining. An important group of clustering algorithms are density based ones that require calculation of a neighborhood of a given data point. The bottleneck for such algorithms are high dimensional data. In this paper, we propose a new TI-k-Neighborhood-Index algorithm that calculates k-neighborhoods for all points in a given data set by means the triangle inequality. We prove experimentally that the NBC (Neighborhood Based Clustering) clustering algorithm supported by our index outperforms NBC supported by such known spatial indices as VA-file and R-tree both in the case of low and high dimensional data.

1 Introduction

Grouping data into meaningful clusters is an important task of both artificial intelligence and data mining. An important group of clustering algorithms are density based ones that require calculation of a neighborhood of a given data point. The bottleneck for such algorithms are high dimensional data. In [4], we have offered a solution to this problem in the case of the DBSCAN (Density Based Clustering with NOISE) algorithm [1], which requires the calculation of a neighborhood within a given radius *Eps* (*Eps-neighborhood*) for each data point. Our solution was based on properties of the triangle property. In this paper, we examine a problem of an efficient calculation of the set of all *k* nearest neighboring points (*k-neighborhood*) for each data point. In the new task, the value of a neighborhood radius is not restricted. The theoretical solution we offer in this paper is also based on properties of the triangle inequality. Based on this solution, we offer a new TI-k-Neighborhood-Index algorithm that calculates *k*-neighborhoods for all points in a given data set. The usefulness of our method is verified by using it in the NBC (Neighborhood Based Clustering) clustering algorithm [7]. We prove experimentally that NBC supported by our index outperforms NBC supported by such known spatial indices as VA-file [2] and R-tree [3] both in the case of low and high dimensional data.

The paper has the following layout. Section 2 recalls the notions of an Eps-neighborhood and k-neighborhood. In Subsection 3.1, we recall a theoretical basis for calculating an Eps-neighborhood for a point within a given radius Eps, we proposed in [4]. In Subsection 3.2, we offer a theoretical basis for calculating k-neighborhood for a point based on the triangle inequality. In Section 4, we offer the TI-k-Neighborhood-Index algorithm for calculating an index that stores k-neighborhoods for all points in a given data set. Section 5 reports the performance of the NBC algorithm depending on a used index including our proposed index. Section 6 concludes the obtained results.

2 Basic Notions

In the sequel, the distance between two points p and q will be denoted by distance(p,q). Please, note that one may use a variety of distance metrics. Depending on an application, one metric may be more suitable than the other. In particular, if Euclidean distance is used, a neighborhood of a point has a spherical shape; when Manhattan distance is used, the shape is rectangular. For simplicity of the presentation, in our examples we will refer to Euclidean distance without loss of generality. Below, we recall definitions of an *Eps-neighborhood of a point* and *k-neighborhood of a point.*

Eps-neighborhood of a point p (denoted by $N_{Eps}(p)$) is defined as the set of points q in dataset D that are different from p and distant from p by no more than Eps; that is,

$$N_{Eps}(p) = \{q \in D \mid q \neq p \wedge \text{distance}(p,q) \leq Eps\}.$$

Let p be a point in D. The set of all points in D that are different from p and closer to p than q will be denoted by CloserN(p, q) that is,

$$\text{CloserN}(p, q) = \{q' \in D \mid q' \neq p \wedge \text{distance}(q',p) < \text{distance}(q,p)\}.$$

Clearly, Closer(p, p) = $\emptyset$.

The *k-neighborhood of a point p* (kNB(p)) is defined as the set of all points q in D such that the cardinality of the set of points different from p that are closer to p than q does not exceed k-1; that is,

$$\text{kNB}(p) = \{q \in D \mid q \neq p \wedge |\text{CloserN}(p, q)| \leq k\text{-}1\}.$$

Please note that for each point p, one may determine a value of parameter Eps in such a way that $N_{Eps}(p)$ = kNB(p). In particular, $N_{Eps}(p)$ = kNB(p) for Eps = distance(q,p), where q is most distant neighbor of point p in kNB(p).

3 Using the Triangle Inequality for Efficient Determination of Neighborhoods

3.1 Efficient Determination of Eps-Neighborhoods

Let us start with recalling the triangle inequality property:

Property 3.1.1. (Triangle inequality property). For any three points p, q, r:

$$\text{distance}(p,r) \leq \text{distance}(p,q) + \text{distance}(q,r)$$

Property 3.1.2 presents its equivalent form, which is more suitable for further considerations.

Property 3.1.2. (Triangle inequality property). For any three points p, q, r:

$$\text{distance}(p,q) \geq \text{distance}(p,r) - \text{distance}(q,r).$$

Now, we recall the results related to Eps-neighborhood we formulated in [4].

Lemma 3.1.1. Let D be a set of points. For any two points p, q in D and any point r:

$$\text{distance}(p,r) - \text{distance}(q,r) > Eps \Rightarrow q \notin N_{Eps}(p) \wedge p \notin N_{Eps}(q).$$

Proof. Let distance(p,r) – distance(q,r) > Eps (*). By Property 3.1.2, distance(p,q) ≥ distance(p,r) – distance(q,r) (**). By (*) and (**), distance(p,q) > Eps, and distance(q,p) = distance(p,q). Hence, $q \notin N_{Eps}(p)$ and $p \notin N_{Eps}(q)$.

By Lemma 3.1.1, if we know that the difference of distances of two points p and q to some point r is greater than Eps, we are able to conclude that $q \notin N_{Eps}(p)$ without calculating the actual distance between p and q.

Theorem 3.1.1. Let r be any point and D be a set of points ordered in a non-decreasing way with respect to their distances to r. Let p be any point in D, q_f be a point following point p in D such that distance(q_f,r) – distance(p,r) > Eps, and q_b be a point preceding point p in D such that distance(p,r) – distance(q_b,r) > Eps. Then:

a) q_f and all points following q_f in D do not belong to $N_{Eps}(p)$.
b) q_b and all points preceding q_b in D do not belong to $N_{Eps}(p)$.

3.2 Efficient Determination of k-Neighborhoods

Now, we will offer a theoretical basis useful for determining k-neighborhood of any point p (kNB(p)).

Theorem 3.2.1. Let r be any point and D be a set of points ordered in a non-decreasing way with respect to their distances to r. Let p be any point in D and Eps be a value such that $|N_{Eps}(p)| \geq k$, q_f be a point following point p in D such that distance(q_f,r) – distance(p,r) > Eps, and q_b be a point preceding point p in D such that distance(p,r) – distance(q_b,r) > Eps. Then:

a) q_f and all points following q_f in D do not belong to kNB(p).
b) q_b and all points preceding q_b in D do not belong to kNB(p).

Proof. Let r be any point and D be a set of points ordered in a non-decreasing way with respect to their distances to r.

a) Let p be any point in D, Eps be a value such that $|N_{Eps}(p)| \geq k$ (*), and q_f be a point following point p in D such that distance(q_f,r) – distance(p,r) > Eps. Then by Theorem 3.1.1a, q_f and all points following q_f in D do not belong to $N_{Eps}(p)$. Hence, there are at least k points different from p that are distant from p by no more than Eps (by *), and q_f and all points following q_f in D are distant from p by more than Eps. Thus, q_f and all points following q_f in D do not belong to kNB(p).

b) The proof is analogous to the proof of Theorem 3.2.1a.

Example 1. Let r be a point (0,0). Figure 1 shows sample set D of two dimensional points. Table 1 illustrates the same set D ordered in a non-decreasing way with respect to the distance of its points to point r. Let us consider a determination of the kNB of point p = F for k=3. Let us assume that we have calculated the distances between F, and points H, G, and C, respectively, and they are as follows: distance(F,H) = 1.64, distance(F,G) = 1.25, distance(F,C) = 1.77. Let Eps = max(distance(H), distance(F,G), distance(F,C)); i.e., Eps = 1.77. This means, that F,H,G,C $\in$ N_{Eps}(p). Thus, p has at least k = 3 points different from itself in its Eps-neighborhood. Now, we note that the first point q_f following point F in D such that distance(q_f,r) – distance(F,r) > Eps is point A (distance(A,r) – distance(F,r) = 5.8 – 3.2 = 2.6 > Eps), and the first point q_b preceding point p in D such that distance(F,r) – distance(q_b,r) > Eps is K (distance(F,r) – distance(G,r) = 3.2 – 0.9 = 2.3 > Eps). By Theorem 3.2.1, the points A, K as well as the points that follow A (here, point B) and precede K in D (here, no point precedes K) do not belong to kNB(F).

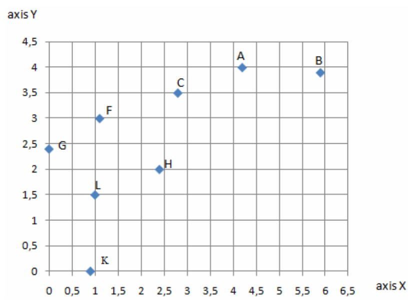

Fig. 1. Set of points D

Table. 1. Ordered set of points D from Fig. 1 with their distance to reference point r(0,0)

q∈D	X	Y	distance(q,r)
K	0,9	0,0	0,9
L	1,0	1,5	1,8
G	0,0	2,4	2,4
H	2,4	2,0	3,1
F	**1,1**	**3,0**	**3,2**
C	2,8	3,5	4,5
A	4,2	4,0	5,8
B	5,9	3,9	7,1

In the sequel, a point r to which the distances of all points in D have been determined will be called a *reference point*.

4 Building k-Neighborhood Index by Using Triangle Inequality

In this section, we present the TI-k-Neighborhood-Index algorithm that uses Theorem 3.2.1 to determine k-neighborhoods for all points in a given dataset D and store them as a k-neighborhood index. The algorithm starts with calculating the distance for each point in D to a reference point r, e.q. to the point with all coordinates equal to 0. Then the points are sorted w.r.t. their distance to r. Next, for each point p in D the TI-k-Neighborhood function is called that returns kNB(p). The function identifies first those k points q following and preceding point p in D for which the difference between distance(p,r) and distance distance(q,r) is smallest. These points are considered as candidates for k nearest neighbors of p. Then radius Eps is calculated as the maximum of the real distances of these points to p. It is guaranteed that real k nearest neighbors lie within this radius from point p. Then the remaining points preceding and following point p in D (starting from points closer to p in the ordered set D) are checked as potential k nearest neighbors of p until the conditions specified in Theorem 3.2.1 are fulfilled. If so, no other points in D are checked as they are guaranteed not to belong to kNB(p). In order to speed up the algorithm, the value of Eps is modified each time a new candidate for a k nearest neighbor is identified.

```
Algorithm TI-k-Neighborhood-Index(set of points D, k);
/* assert: r denotes a reference point, e.g. with all coordinates = 0 */
/* assert: There are more than k points in D                          */
  for each point p in set D do p.dist = Distance(p,r) endfor;
  sort all points in D non-decreasingly wrt. attribute dist;
  for each point p in the ordered set D starting from
      the first point until last point in D do
      insert (position of point p, TI-k-Neighborhood(D, p, k))
          to k-Neighborhood-Index
  endfor
```

```
  function TI-k-Neighborhood(D, point p, k)
  b = p; f = p;
  backwardSearch = PrecedingPoint(D, b);
  forwardSearch = FollowingPoint(D, f);
  k-Neighborhood = {}; i = 0;
  Find-First-k-Candidate-Neighbours-Forward&Backward(D, p, b, f,
      backwardSearch, forwardSearch, k-Neighborhood, k, i);
  Find-First-k-Candidate-Neighbours-Backward(D, p, b,
      backwardSearch, k-Neighborhood, k, i);
  Find-First-k-Candidate-Neighbours-Forward(D, p, f,
      forwardSearch, k-Neighborhood, k, i);
  p.Eps = max({e.dist| e ∈ k-Neighborhood});
  Verify-k-Candidate-Neighbours-Backward(D, p, b, backwardSearch,
      k-Neighborhood, k);
  Verify-k-Candidate-Neighbours-Forward(D, p, f, forwardSearch,
      k-Neighborhood, k);
  return k-Neighborhood
```

```
  function PrecedingPoint(D, var point p)
  if there is a point in D preceding p then
      p = point immediately preceding p in D; backwardSearch = true
  else backwardSearch = false endif
  return backwardSearch
```

```
  function FollowingPoint(D, var point p)
  if there is a point in D following p then
      p = point immediately following p in D; forwardSearch = true
  else forwardSearch = false endif
  return forwardSearch
```

```
  function Find-First-k-Candidate-Neighbours-Forward&Backward(D,
      var point p, var point b, var point f,
      var backwardSearch, var forwardSearch, var k-Neighborhood,
      k, var i)
  while backwardSearch and forwardSearch and (i < k) do
      if p.dist - b.dist < f.dist - p.dist then
          dist = Distance(b, p); i = i + 1;
          insert element e = (position of b, dist)
          in k-Neighborhood holding it sorted wrt. e.dist;
          backwardSearch = PrecedingPoint(D, b)
      else
          dist = Distance(f, p); i = i + 1;
          insert element e = (position of b, dist)
          in k-Neighborhood holding it sorted wrt. e.dist;
          forwardSearch = FollowingPoint(D, f);
      endif
  endwhile
```

```
function Find-First-k-Candidate-Neighbours-Backward(D,var point p,
  var point b, var backwardSearch, var k-Neighborhood, k, var i)
while backwardSearch and (i < k) do
    dist = Distance(b, p); i = i + 1;
    insert element e = (position of b, dist)
    in k-Neighborhood holding it sorted wrt. e.dist;
    backwardSearch = PrecedingPoint(D, b)
endwhile
```

```
function Find-First-k-Candidate-Neighbours-Forward(D, var point p,
  var point f, var forwardSearch, var k-Neighborhood, k, var i)
while forwardSearch and (i < k) do
    dist = Distance(f, p); i = i + 1;
    insert element e = (position of b, dist)
    in k-Neighborhood holding it sorted wrt. e.dist;
    forwardSearch = FollowingPoint(D, b)
endwhile
```

```
function Verify-k-Candidate-Neighbours-Backward(D, var point p,
    var point b, var backwardSearch, var k-Neighborhood, k)
while backwardSearch and ((p.dist – b.dist) ≤ p.Eps) do
    dist = Distance(b, p);
    if dist < p.Eps then
        i = |{e ∈ k-Neighborhood| e.dist = p.Eps}};
        if |k-Neighborhood| - i ≥ k - 1 then
            delete each element e with e.dist = p.Eps
                from k-Neighborhood;
            insert element e = (position of b, dist) in
                k-Neighborhood holding it sorted wrt. e.dist;
            p.Eps = max({e.dist| e ∈ k-Neighborhood});
        else
            insert element e = (position of b, dist) in
                k-Neighborhood holding it sorted wrt. e.dist;
        endif
    elseif dist = p.Eps
        insert element e = (position of b, dist) in
            k-Neighborhood holding it sorted wrt. e.dist
    endif
    backwardSearch = PrecedingPoint(D, b)
endwhile
```

```
function Verify-k-Candidate-Neighbours-Forward(D, var point p,
    var point f, var forwardSearch, var k-Neighborhood, k)
while forwardSearch and ((f.dist – p.dist) ≤ p.Eps) do
    dist = Distance(f, p);
    if dist < p.Eps then
        i = |{e ∈ k-Neighborhood| e.dist = p.Eps}};
        if |k-Neighborhood| - i ≥ k - 1 then
            delete each element e with e.dist = p.Eps
                from k-Neighborhood;
            insert element e = (position of b, dist) in
                k-Neighborhood holding it sorted wrt. e.dist;
            p.Eps = max({e.dist| e ∈ k-Neighborhood});
        else
            insert element e = (position of b, dist) in
                k-Neighborhood holding it sorted wrt. e.dist;
        endif
    elseif dist = p.Eps
        insert element e = (position of f, dist) in
            k-Neighborhood holding it sorted wrt. e.dist
    endif
    forwardSearch = FollowingPoint(D, f)
endwhile
```

5 Experimental Results

In this section, we report the run times of clustering with the NBC versions supported by k-neighboorhood index (being the result of the TI-k-Neighboorhood-Index algorithm), R-tree and VA-file. The versions are denoted as TI-NBC, R-tree-NBC and VA-file-NBC, respectively. The reported run times include the time of creating an index. In the experiments, we used a number of datasets (and/or their subsamples) of different cardinality and dimensionality. In particular, we used widely known datasets such as: birch [6], SEQUOIA 2000 [5], and kddcup 98 [8] as well as datasets generated automatically (random) or manually.

Table 2. Datasets used in experiments and run times (in milliseconds) of examined algorithms for k=10. Notation: dim. – number of dimensions, card. – number of points, N/A – results not available within at least 12 hours

No.	dataset	dim.	card.	R-tree-NBC	VA-file-NBC	TI-NBC
1	birch	2	100000	N/A	N/A	154 657
2	sequoia_2000	2	1252	1 719	1 546	78
3	sequoia_2000	2	2503	3 062	7 171	188
4	sequoia_2000	2	3910	4 750	21 360	329
5	sequoia_2000	2	5213	6 235	38 891	499
6	sequoia_2000	2	6256	7 547	63 250	655
7	sequoia_2000	2	62556	NA	NA	38 750
8	manual	2	2658	3 844	10 375	172
9	manual	2	14453	21 297	NA	2 296
10	random	3	50000	148 047	NA	65 656
11	random	5	100000	NA	NA	1 374 297
12	random	10	10000	NA	NA	44 297
13	random	10	20000	NA	NA	177 797
14	random	10	50000	NA	NA	1 421 797
15	random	20	500	2 969	NA	281
16	random	40	500	5 328	NA	500
17	random	50	10000	NA	NA	232 547
18	random	50	20000	NA	NA	1 111 531
19	covtype	54	150000	NA	NA	32 220 735
20	kddcup_98	56	56000	NA	NA	1 371 062
21	random	100	1000	NA	NA	4 641
22	random	100	10000	NA	NA	449 813
23	random	100	20000	NA	NA	1 860 921
24	random	200	500	NA	NA	2 297
25	random	200	1000	NA	NA	9 047

The run times are presented in Table 2. In several cases, it was impossible to cluster data based on R-tree or VA-file indices within 12 or more hours. In the cases, in which it was possible, TI-NBC outperformed both R-tree-NBC (up to 1 order of magnitude) and VA-file-NBC (up to 2 orders of magnitude). In addition, TI-NBC turned out capable to cluster high dimensional data (up to a few hundreds dimensions).

6 Conclusions

In the paper, we have proposed the new method for determining k-neighborhood of a given point based on the triangle inequality. We used this method to propose a new algorithm for creating index with k-neighborhoods for all points in a data set. We proved that clustering with the NBC algorithm that uses our method is more efficient than NBC supported by such known spatial indices such as R-tree or VA-file. Unlike NBC supported by R-tree or VA-file, our method enables clustering in the case of large high dimensional datasets.

References

[1] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Database with Noise. In: Proc. of KDD'96, pp. 226–231 (1996)

[2] Blott, S., Weber, R.: A Simple Vector Approximation File for Similarity Search in High-dimensional Vector Spaces, Technical Report 19, ESPRIT project HERMES, vol. 9141, Technical Report number 19 (March 1997)

[3] Guttman, A.: R-Trees: A Dynamic Index Structure For Spatial Searching. In: Proc. of ACM SIGMOD, Boston, pp. 47–57 (1984)

[4] Kryszkiewicz, M., Lasek, P.: TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality. In: Szczuka, M. (ed.) RSCTC 2010. LNCS, vol. 6086, pp. 60–69. Springer, Heidelberg (2010)

[5] Stonebraker, M., Frew, J., Gardels, K., Meredith, J.: The SEQUOIA 2000 Storage Benchmark. In: Proc. of ACM SIGMOD, Washington, pp. 2–11 (1993)

[6] Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: A New Data Clustering Algorithm and its Applications. Data Mining and Knowledge Discovery 1(2), 141–182 (1997)

[7] Zhou, S., Zhao, Y., Guan, J., Huang, J.Z.: A Neighborhood-Based Clustering Algorithm. In: Proc. of PAKDD, pp. 361–371 (2005)

[8] `http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html`