Divide-and-conquer algorithm for creating neighborhood graph for clustering

Olli Virmajoki and Pasi Fränti
Department of Computer Science, University of Joensuu, PB 111, FIN-80101 Joensuu, Finland.

ovirma@cs.joensuu.fi, franti@cs.joensuu.fi

Abstract

K-nearest neighbor graph has been used for reducing the number of distance calculations in PNN-based clustering. The bottleneck of the approach is the creation of the graph. In this paper, we develop a fast divide-and-conquer method for graph creation based on algorithm previously used in the closest pair problem. The proposed algorithm is then applied to agglomerative clustering, in which it outperforms previous projection-based algorithm for high dimensional spatial data sets.

1. Introduction

Agglomerative clustering is popular method for generating the clustering hierarchically by a sequence of merge operations. Ward’s method [1] selects the cluster pair to be merged that minimizes the increase in the distortion function value. In vector quantization, it is known as pairwise nearest neighbor (PNN) method [2]. The main drawback of PNN method is its slowness. The straightforward implementation requires $O(N^2)$ time whereas a faster implementation exists [3] but still lower bounded by $\Omega(N^2)$. The main source of computation originates from the search of the nearest neighbor cluster.

In a previous work, it was shown that neighborhood graph can be applied for reducing the number of distance calculations in PNN [4]. Concept referred as k-nearest neighbor graph (kNN graph) was introduced. In the graph, every node represents a cluster, and the edges correspond to the pointers to the neighbor clusters. The graph is utilized as a search structure: every time we need to search for the nearest neighbor, we consider only the clusters that are neighbors in the graph structure. Thus, the use of the graph approximates the $O(N)$ time full search by a faster $O(k)$ time search method.

The potential improvement due to the neighborhood graph was found to be significant in [4] but the graph creation is the bottleneck of the algorithm. In this work, we propose a new algorithm based on divide-and-conquer technique proposed for the closest pair problem [5]. Experimental results indicate that the proposed approach works well in the case of high-dimensional data sets.

2. Graph-based clustering

The clustering problem is defined here as a combinatorial optimization problem. Given a set of $N$ data vectors $X=\{x_1, x_2, \ldots, x_N\}$, partition the data set into $M$ clusters so that a given distortion function is minimized. Partition $P=\{p_1, p_2, \ldots, p_N\}$ defines the clustering by giving for each data vector the index of the cluster where it is assigned to. A cluster $s_a$ is defined as the set of data vectors that belong to the same partition $a$.

The clustering is then represented as the set $S=\{s_1, s_2, \ldots, s_M\}$. In vector quantization, the output of the clustering is a codebook $C=\{c_1, c_2, \ldots, c_M\}$, which is usually the set of cluster centroids. We assume that the vectors belong to Euclidean space, and use the mean square error (MSE) as the distortion function.

2.1. Pairwise nearest neighbor method

The pairwise nearest neighbor (PNN) method [1,2] generates the clustering hierarchically by a sequence of merge operations. At each step, two nearby clusters are merged. The method uses greedy strategy by choosing the cluster pair that increases the MSE least. A fast variant with linear memory consumption stores for every cluster a pointer to its nearest neighbor cluster [3]. Nevertheless, there are $O(\tau N)$ distance calculations to be performed after every merge operation, which results in $O(\tau N^2)$ time algorithm, where $\tau$ is the number of clusters to be updated.

2.2. K-nearest neighbor graph

We define k-nearest neighbor graph (kNN graph) as a weighted directed graph, in which every node represents a single cluster, and the edges correspond to the pointers to the neighbor clusters. The distance of clusters is defined by the merge distortion function [1,2].

The graph is utilized as a search structure: every time we need to search for the nearest neighbor, we consider only the clusters that are neighbors in the graph structure. Thus, the use of the graph approximates the $O(N)$ time full search by a faster $O(k)$ time search method.
2.3. Utilization of the graph

The main structure of the algorithm is given in Fig. 1. The algorithm starts by initializing every data vector as its own clusters, and by constructing the neighborhood graph. The algorithm iterates by removing nodes from the graph until the desired number of clusters has been reached.

At first, the edge with smallest weight is found, and the nodes \((s_a, s_b)\) are merged. The algorithm creates a new node \(s_{ab}\) from the clusters \(s_a\) and \(s_b\), which are removed from the graph. The corresponding edge costs are updated. The algorithm must also calculate cost values for the outgoing edges from the newly created node \(s_{ab}\). The \(k\) nearest neighbors is found among the \(2k\) neighbors of the previously merged nodes \(s_a\) and \(s_b\). The time complexity of the iterations is \(O(\varepsilon N \log N)\). A sample nearest neighbor graph is illustrated in Fig. 2.

3. Creation of the graph

The graph can be constructed by brute force by considering all pairwise distances but at the cost of \(O(N^2)\) time. We therefore consider two faster methods: the mean-distance ordered partial search (MPS), and a new divide-and-conquer technique.

3.1. MPS for searching \(k\) neighbors

The MPS method was originally proposed for k-means clustering in [6], generalized to the PNN in [7], and used for the graph creation in [4]. The MPS method stores the component sums of each cluster centroid (code vectors), which correspond to the projections of the vectors to the diagonal axis of the vector space. Then, given the cost function value of the best candidate found so far, vectors outside the radius defined by a given pre-condition can be excluded in the calculations.

For finding the \(k\) nearest clusters, the condition of the graph was relaxed so that we find any \(k\) neighbors instead of the nearest ones. In particular, we use the exact MPS method for finding the nearest neighbor but stop the search immediately when it has been found. In addition to this, we maintain ordered list of the \(k\) best candidates found so far. The rest of the neighbors are then chosen simply from the list of the candidates. Even though the method is faster than the brute force, it still dominates the running time in the experiments made in [4]. Thus, faster graph creation is still desired.

3.2. Closest pair problem

We adopt the idea from a solution given to the closest pair problem [5]. The closest pair problem is stated as follows: given \(N\) points in \(d\)-dimensional space, find the two whose mutual distance is the smallest. The problem can be solved by recursive algorithm:

1. Divide \(X\) into \(X_1\) and \(X_2\) by the median hyper plane \(H\) normal to some axis.
2. Recursively solve the problem for \(X_1\) and \(X_2\).
3. Compute \(\delta = \min(\delta_1, \delta_2)\), where \(\delta_1\) and \(\delta_2\) are the found distances in \(X_1\) and \(X_2\).
4. Let \(X_3\) be the set of points that are within \(\delta\) of \(H\).
5. Use the \(\delta\) -sparsity condition to recursively examine all pairs in \(X_3\).

It has been shown that, in the case of 2-dimensional vector space, only a constant number of points can be neighbor in any cell in the set \(X_3\) [8]. Assuming that the same primary axis is used in the division, the points can be pre-sorted and the analysis step can be performed in linear time. It has been proven that the algorithm takes \(O(N \log N)\) time and the algorithm generalizes to multi-dimensional spaces but at the cost \(O(N \log^{d-1} N)\) time [9], where \(d\) is the number of dimensions.

---

**Fig. 1.** Structure of the Graph-PNN.

**Fig. 2.** Illustration of a kNN graph for \(k=4\).
Divide-and-Conquer($X$, $k$, $c_k$) $\rightarrow$ kNN

IF ($|X| > c_k$) THEN
  $X_1$, $X_2$, $proj$ $\leftarrow$ Divide($X$);
  kNN1 $\leftarrow$ Divide-and-conquer($X_1$, $k$, $c_k$);
  kNN2 $\leftarrow$ Divide-and-conquer($X_2$, $k$, $c_k$);
  kNN $\leftarrow$ kNN1 $\cup$ kNN2;
ELSE
  kNN $\leftarrow$ BruteForce($X$, kNN, $k$);
END-IF
RETURN kNN;

GenerateThirdSet($X$, kNN, proj) $\rightarrow$ $X_3$

$X_3$ $\leftarrow$ $\emptyset$;
FOR $i$ $\leftarrow$ 1 TO $|X|$ DO
  $\delta$ $\leftarrow$ ProjectionDistance($X[i]$, proj);
  IF $\delta < k$ THEN
    $X_3$ $\leftarrow$ $X_3$ $\cup$ $X[i]$;
  END-IF
RETURN $X_3$;

Fig. 3. Sketch of the divide-and-conquer algorithm.

3.3. Divide-and-conquer for k-nearest neighbors

We introduce next an algorithm applicable for finding $k$-nearest neighbors based on the above divide-and-conquer approach with the following differences. Firstly, we search several closest pairs for every vector in the data set. Secondly, we use principle component analysis (PCA) for calculating the projection axis with the maximum deviation. Thirdly, we use a distance-based heuristic for selecting the vectors to be included in the third sub set.

The pseudo code of the algorithm is given in Fig. 3. At each step of the recursion, we divide the data set $X$ into two sub sets $X_1$ and $X_2$ of equal sizes as follows. We first calculate the principle axis of the data vectors in $X$, and then select ($d$-1)-dimensional hyper plane $H$ perpendicular to the principal axis. The hyper plane is selected so that approximately half of the vectors belong to one side of the space, and the rest to the other side. Once the dividing procedure has been done, the two sub problems $X_1$ and $X_2$ are solved recursively. Sub problems smaller than $c_k$ are solved by brute force search.

After the sub problems have been solved, we generate a third sub set $X_3$ consisting of vectors that are closer to the dividing hyper plane $H$ than to its nearest neighbor in the corresponding sub set ($X_1$ or $X_2$). By using the control parameter $c$ we can control the number of vectors chosen in the sub set. Once the sub set is created, the algorithm is recursively applied for $X_3$. Finally, the results of the three sub problems are combined. Fig. 4 illustrates the division of the set $X$ to three overlapping sub sets.

The time complexity of the proposed divide-and-conquer algorithm is estimated here by a recurrence $T(N) = 3T(N/2) + O(Nd^2)$ assuming that the size of the third sub set is less than equal to that of the other sub sets $X_1$ and $X_2$. The second term originate from the calculation of the principal axis. The rest of the calculations can be performed in linear time. The recurrence solves to $O(d^2N^{0.58}\log N)$. It might be possible to squeeze the complexity to $O(N\log N)$ by selecting the dividing hyper plane by some simpler method, and by making tighter bounds for the third sub set. Note that the size of the $X_3$ can be controlled by the parameter $c$.

4. Experiments

We consider three data sets from [3] with the number of clusters fixed to $M=256$. The illustration in Fig. 5 shows the amount of work required by the graph creation and the PNN iterations with the different number of neighborhood size ($k$). In the following, we fix the neighborhood size to $k=4$.
Table 1. Running times of the graph creation.

<table>
<thead>
<tr>
<th>Graph/PNN</th>
<th>Bridge</th>
<th>Time</th>
<th>MSE</th>
<th>House</th>
<th>Time</th>
<th>MSE</th>
<th>Miss America</th>
<th>Time</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force</td>
<td>34</td>
<td>171.17</td>
<td>881</td>
<td>6.43</td>
<td>89</td>
<td>5.44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brute Force /fast</td>
<td>16</td>
<td>172.20</td>
<td>446</td>
<td>6.33</td>
<td>43</td>
<td>5.48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPS</td>
<td>3</td>
<td>171.11</td>
<td>18</td>
<td>6.37</td>
<td>44</td>
<td>5.44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Divide-and-Conquer</td>
<td>2</td>
<td>171.80</td>
<td>49</td>
<td>6.58</td>
<td>7</td>
<td>5.44</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Comparison of the Graph-PNN to other methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Bridge</th>
<th>Time</th>
<th>MSE</th>
<th>House</th>
<th>Time</th>
<th>MSE</th>
<th>Miss America</th>
<th>Time</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast PNN</td>
<td>Full search</td>
<td>79</td>
<td>168.92</td>
<td>1574</td>
<td>6.27</td>
<td>229</td>
<td>5.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+PDS+MPS+Lazy</td>
<td>9</td>
<td>168.92</td>
<td>190</td>
<td>6.26</td>
<td>106</td>
<td>5.37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph PNN</td>
<td>Full MPS</td>
<td>3</td>
<td>170.28</td>
<td>19</td>
<td>6.33</td>
<td>45</td>
<td>5.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Limited search MPS</td>
<td>3</td>
<td>170.56</td>
<td>14</td>
<td>6.51</td>
<td>6</td>
<td>5.58</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide-and-conquer</td>
<td>2</td>
<td>170.69</td>
<td>51</td>
<td>6.37</td>
<td>8</td>
<td>5.43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph PNN + GLA</td>
<td>Full MPS</td>
<td>4</td>
<td>166.23</td>
<td>20</td>
<td>6.14</td>
<td>47</td>
<td>5.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Limited search MPS</td>
<td>4</td>
<td>166.38</td>
<td>15</td>
<td>6.18</td>
<td>9</td>
<td>5.34</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide-and-conquer</td>
<td>2</td>
<td>165.81</td>
<td>52</td>
<td>6.14</td>
<td>10</td>
<td>5.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLA</td>
<td>Standard</td>
<td>13</td>
<td>179.95</td>
<td>23</td>
<td>7.77</td>
<td>20</td>
<td>5.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+PDS+MPS+Activity</td>
<td>2</td>
<td>180.02</td>
<td>3</td>
<td>7.80</td>
<td>8</td>
<td>5.95</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 6. Time-distortion performance of the Graph-PNN.

The running times of the graph creation are summarized in Table 1, and the overall results in Table 2. Comparative results are given for the fast exact PNN [3], and the fast exact PNN with several speed-up methods as proposed in [7]. Two k-means variants (GLA and Fast GLA) are also included. The results show that the divide-and-conquer technique is faster than the MPS with the 16-dimensional data sets (Bridge and Miss America) but slower with the 3-dimensional data set (House). The time-distortion performance is illustrated in Fig. 6 for Miss America with the two graph creation algorithms.

5. Conclusion

The proposed divide-and-conquer technique provides fast method for the creation of the graph, and it improves the previous method in the case of the high dimensional data sets. In the 3-dimensional color clustering, on the other hand, the method was slower than the existing MPS method.

6. References