Fast PNN-based Clustering Using K-nearest Neighbor Graph

Pasi Fränti, Olli Virmajoki and Ville Hautamäki
Department of Computer Science, University of Joensuu, PB 111, FIN-80101 Joensuu, Finland.
franti@cs.joensuu.fi, ovirma@cs.joensuu.fi, villeh@cs.joensuu.fi

Abstract

Search for nearest neighbor is the main source of computation in most clustering algorithms. We propose the use of nearest neighbor graph for reducing the number of candidates. The number of distance calculations per search can be reduced from O(N) to O(k) where N is the number of clusters, and k is the number of neighbors in the graph. We apply the proposed scheme within agglomerative clustering algorithm known as the PNN algorithm.

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 distortion function value. In vector quantization, it is known as the *pairwise nearest neighbor* (*PNN*) method [2].

The main drawback of the PNN is its slowness. The original implementation requires $O(N^3)$ distance calculations. An order of magnitude faster algorithm has been introduced in [3] but the method is still lower bounded by $\Omega(N^2)$. The main source of computation originates from the search of the nearest neighbor cluster.

Another approach is to use graph theoretical methods. In [4], a complete undirected graph is created where the nodes correspond to the data vectors, and the edge costs to vector distances according to a given *distortion* measure. The resulting graph can be trimmed to a *minimal spanning tree*. Clustering can then be generated by iteratively dividing the cluster by removing longest edges from the graph. In the final graph, clusters are defined by the separate components in the graph. This can be seen as a variant of a split-based clustering with *single-linkage* criterion.

In this work, we introduce fast agglomerative clustering algorithm motivated by the graph-based approaches. In our approach, however, we process the data at the cluster level so that every node in the graph represents a cluster, not a single vector. The edges of the graph represent inter cluster connections between nearby clusters. The graph is used as a search structure for reducing the number of distance calculations.

The proposed approach has two specific problems to solve: (1) how to generate the graph efficiently, and (2) how to utilize it. Standard solutions for minimum spanning tree take $O(N^2)$ time, which would prevent any speed-up. We propose solution for the first problem by considering Mean-distance ordered partial search [5]. We study the second sub-problem in detail and propose double-linked list, and use heap for efficient search for the cluster pair to be merged. We will show by experiments that a relatively small neighborhood size is sufficient for preserving the good quality clustering results.

2. Pairwise nearest neighbor method

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.

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 is given in [3].

3. 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 pointers to neighbor clusters. Every node has k neighbors. The distance of clusters is defined by the merge distortion

Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)
0-7695-1978-4/03 $ 17.00 © 2003 IEEE

function. Note that this is not the only possible definition: others have been given in [6], [7].

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.

3.1. MPS for searching nearest neighbor

The graph can be constructed by brute force but at the cost of O(N^2) time. We therefore propose a faster method based on the *Mean-distance ordered partial search* (MPS). It was originally proposed for K-means clustering (GLA) in [5] but generalized to the PNN in [8].

The MPS method stores the component sums of each cluster centroid (code vectors). The component sums correspond to the projections of the vectors to the diagonal axis of the vector space. In typical data sets, the code vectors are highly concentrated along the diagonal axis, and therefore, the distance of their component sums highly correlate to their real distance. 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, see Fig. 1.

The pre-condition is utilized as follows. The vectors are sorted according to their component sums, and then proceed in the order given by the sorting. The search starts from the given cluster, and proceeds bi-directionally along the projection axis. If the pre-condition holds true, the calculation the candidate cluster can be rejected.

3.2. MPS for searching *k* neighbors

For finding the k nearest clusters, we relax the condition of the graph and find any k neighbors instead of the nearest ones. This is a reasonable modification because the optimality of the graph cannot be guaranteed during the process of the PNN algorithm. Thus, by relaxing the definition of the k-nearest neighbor graph, speed-up can be obtained at a slight increase in the distortion.

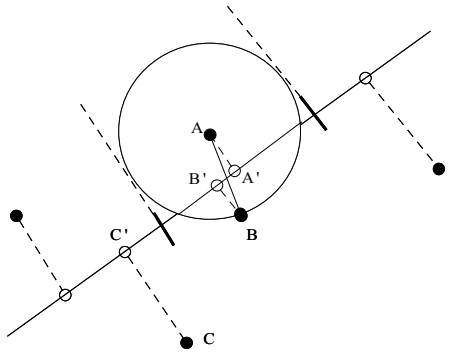

Fig. 1. Vectors (black dots) and their projections (empty dots) according to the component sums.

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 no matter whether they are actually the k-1 nearest or not. It is expected that the rest of the candidates are nearby vectors although not necessarily the nearest ones. Even if some links were missing, vectors in the same cluster are most likely to be connected anyhow.

4. Graph-PNN

The proposed Graph-PNN is based on the exact PNN method but we utilize the graph structure in the search of nearest neighbor clusters.

4.1. Simple implementation

The main structure of the algorithm is given in Fig. 2. 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 and 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 merge procedure is illustrated in Fig. 3 for a sample 2NN graph (k=2).

```
GraphPNN(X, M) → S
    FOR i←1 to N DO
        s_i ← {x_i};
    FOR ∀(s_i; i ∈ [1, N]) DO
        Find k nearest neighbors;
    REPEAT
        (s_a, s_b) ← SearchNearestClustersInGraph(S);
        s_ab ← Merge(s_a, s_b);
        Find the k nearest neighbors for s_ab;
        Update the nodes that had s_a and s_b as neighbors;
    UNTIL |S|=M;
```

Fig. 2. Structure of the Graph-PNN.

4.2. Double linked list

The PNN iterations take O(1) time to find the smallest distance if we use heap structure. The update of the data structures and recalculation of the distances for merged node takes O($2k^2 + \log N + kN + N$) time. The first term ($2k^2$) originates from the update of the data structures and

Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)
0-7695-1978-4/03 $ 17.00 © 2003 IEEE

recalculation of the distances. The second term ($\log N$) comes from the update of the heap structure. The third and fourth terms ($kN+N$) comes from updating edges that pointed to the obsolete nodes, and the time needed to recalculate those edge values.

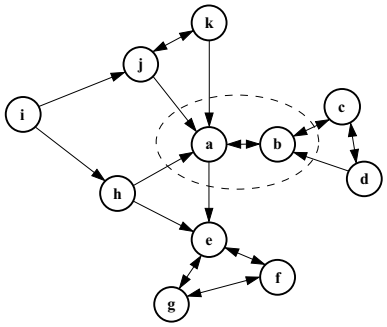

Fig. 3. Illustration of the graph where *a* and *b* are to be merged.

To sum up, one step requires $O(kN)$ time, which sums up to $O(\tau N^2)$, where τ is the number of incoming pointers. In general, this is too much and we therefore consider the double linked list (Fig. 4), in which we maintain for every node two lists: the first list points to the neighbor clusters, and the second list contains so called "back pointers" to clusters that have the node as their nearest neighbors. In this way, we can eliminate $O(N)$ time loops, and the time complexity becomes $O(\tau N \log N)$, see Tables 1 and 2.

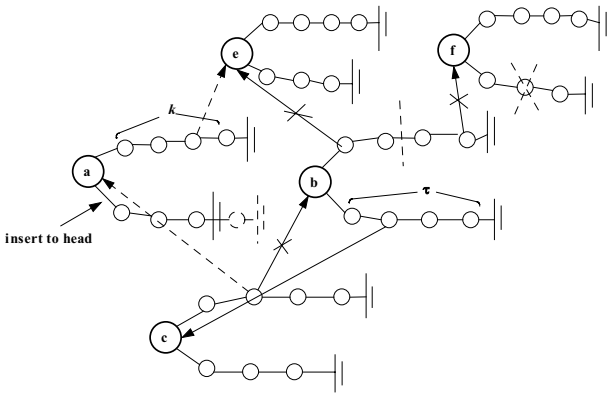

Fig. 4. Illustration of the update of the linked list in the merge procedure of the clusters *a* and *b* in the neighborhood graph.

5. Experiments

We consider three data sets from [3], [10], and number of clusters fixed to M=256. The illustrations in Fig. 5 show that the PNN iterations can be performed efficiently and that the graph creation is the bottleneck of the algorithm. The results also indicate that a very small neighborhood size, such as k=3, is sufficient for obtaining high quality clustering.

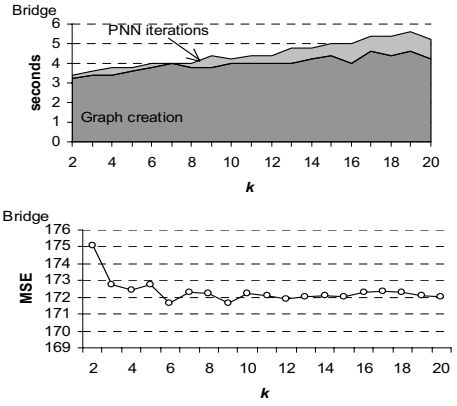

Fig. 5. Effect of the neighborhood size on running time.

The running times and the number of distance calculations are summarized in Table 3. Comparative results are given for the fast exact *PNN* [3], and the fast exact *PNN* with several speed-up methods as proposed in [8]. The results show that the *graph PNN* is significantly faster than the fast exact *PNN*. The graph creation is evidently a bottleneck in the *Graph-PNN*. We therefore consider limiting the search of MPS by a control parameter, see Fig. 6.

Comparative results are shown in Table 4 including *Fast PNN* [3], and its faster variant [8] using three practical speed-up techniques such as the *PDS*, *MPS* and *Lazy evaluation* of the distances. The *GLA* has also two variants: the original method [9], and a faster variant that uses PDS, MPS and activity detection for speed-up [10]. Results are given also for *Graph-PNN + GLA*, in which the data is first processed by the *Graph-PNN* and the result is input to the *GLA*.

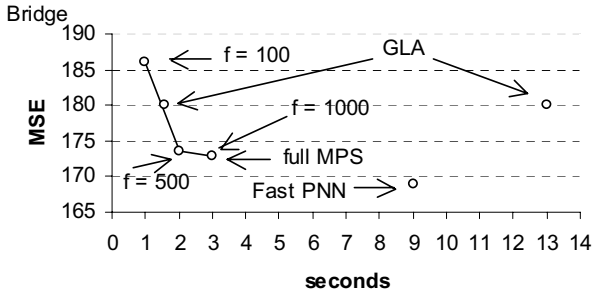

Fig. 6. Time-distortion performance of the *limited search MPS*.

6. Conclusion

Fast Graph-based PNN has been proposed. We found out that a relatively small neighborhood size (k = 2, 3, 4 or 5) can produce clustering results close to the exact PNN method but with a significantly smaller number of distance calculations and shorter running time.

Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)
0-7695-1978-4/03 $ 17.00 © 2003 IEEE

Table 1: Estimated number of steps and distance calculations of the PNN iterations (*Bridge*).

Steps:	Fast PNN		Graph PNN (simple)		Graph PNN (double-linked)	
	Steps	Distances	Steps	Dists	Steps	Dists
SearchNearest( )	N	-	1	-	1	-
Merge(a, b)	N	-	$2k^2 + \log N$	$2k$	$2k^2 + \tau k + \log N$	$2k$
FindNeighbors(a, b)	N	-	$k N$	-	τk	-
RemoveLast( )	N	-	$k + 2log N$	-	$\log N$	-
UpdateDistances( )	$N(1+\tau)$	τN	$N + \tau \log N / k$	τ	$\tau (1+ \log N / k)$	τ

Table 2: Observed number of steps and distance calculations of the PNN iterations (*Bridge*).

	Fast PNN		Graph PNN (simple)		Graph PNN (double-linked)	
	Steps	Distances	Steps	Distances	Steps	Distances
SearchNearests( )	8 357 760	-	3 840	-	3 840	-
Merge(a, b)	8 357 760	-	67 362	8 210	108 410	8 217
FindNeighbors(a, b)	8 357 760	-	16 715 520	-	30 868	-
RemoveLast( )	8 349 185	-	90 979	-	45 514	-
UpdateDistances( )	48 538 136	40 166 328	8 478 587	11 285	145 722	11 261
Total	81 960 601	40 166 328	25 356 288	19 495	334 354	19 478

Table 3. Summary of the running times and the number of distance calculations.

		Bridge		*House*		*Miss America*	
		Distance calculations	Run time	Distance calculations	Run time	Distance calculations	Run time
Fast PNN		48 552 888	79	2 237 460 562	1574	128 323 740	229
Fast PNN	+MPS+PDS+lazy	6 167 439	9	37 752 863	190	83 323 889	106
Graph PNN	Graph creation	2 341 547	3	19 017 163	17	32 440 442	46
	Iterations	24 431	< 1	166 475	1	40 446	< 1
	Total	2 365 978	3	19 183 638	18	32 480 888	46

Table 4. Comparison of the Graph-PNN with other methods.

		Bridge (N=)		*House*		*Miss America*	
		Run time	MSE	Run time	MSE	Run time	MSE
Fast PNN	Full search	79	168.92	1574	6.27	229	5.36
	+PDS+MPS+Lazy	9	168.92	190	6.26	106	5.37
Graph PNN	Full MPS	3	172.96	18	6.47	46	5.49
	Limited search MPS	3	173.46	13	6.59	5	5.67
Graph PNN + GLA	Full MPS	4	166.85	19	6.14	48	5.31
	Limited search MPS	3	167.06	14	6.16	8	5.38
GLA	Full search	13	179.95	22.8	7.77	19.7	5.95
	+PDS+MPS+Activity	1.6	180.02	3	7.80	8.3	5.95

7. References

[1] J.H. Ward, "Hierarchical grouping to optimize an objective function," *J. Amer. Statist.Assoc.*, 58, 236-244, 1963.

[2] W.H. Equitz, "A new vector quantization clustering algorithm," *IEEE-ASSP*, 37(10), 1568-1575, Oct. 1989.

[3] P. Fränti, T. Kaukoranta, D.-F. Shen and K.-S. Chang, "Fast and memory efficient implementation of the exact PNN," *IEEE-IP*, 9(5), 773-777, May 2000.

[4] J.C. Cover and G.J.S. Ross, "Minimum spanning trees and simple-linkage cluster analysis," *Applied Statistics*, 18, 54-64, 1969.

[5] S.-W. Ra and J.K. Kim, "A fast mean-distance-ordered partial codebook search algorithm for image vector quantization," *IEEE-CS*, 40(9), 576-579, September 1993.

[6] S. Arya and D.M. Mount, "Algorithm for fast vector quantization," *IEEE Data Compresion Conference*, Snowbird Utah, 381-390, 1994.

[7] A.D. Constantinou, R.D. Bull and C.N. Canagarajah, "A new class of VQ codebook design algorithms using adjacency maps," *SPIE Electronics Imaging 2000*, San Jose, 3974, 625-634, 2000.

[8] O. Virmajoki, P. Fränti, T. Kaukoranta, "Practical methods for speeding-up the pairwise nearest neighbor method", *Optical Engineering*, 40(11), 2495-2504, November 2001.

[9] Y. Linde, A. Buzo and R.M. Gray, "An Algorithm for Vector Quantizer Design," *IEEE-COM*, 28(1), 84-95, Jan. 1980.

[10] T. Kaukoranta, P. Fränti and O. Nevalainen, "A fast exact GLA based on code vector activity detection", *IEEE-IP*, 9(8), 1337-1342, Aug. 2000.