Hybrid Approaches for Clustering

Laxmi Kankanala and M. Narasimha Murty

Indian Institute of Science, Bangalore, India
{laxmi,mnm}@csa.iisc.ernet.in

Abstract. Applications in various domains often lead to very large and frequently high-dimensional data. Successful algorithms must avoid the curse of dimensionality but at the same time should be computationally efficient. Finding useful patterns in large datasets has attracted considerable interest recently. The primary goal of the paper is to implement an efficient Hybrid Tree based clustering method based on CF-Tree and KD-Tree, and combine the clustering methods with KNN-Classification. The implementation of the algorithm involves many issues like *good accuracy*, *less space* and *less time*. We will evaluate the time and space efficiency, data input order sensitivity, and clustering quality through several experiments.

1 Introduction

Classification process attempts to generate the description of the classes, and those descriptions helps to classify the unknown data points. Data clustering identifies the sparse and the crowded places, and hence discovers the overall distribution patterns of the dataset. Besides, the derived clusters can be visualized more efficiently and effectively than the original dataset.[1]

A pattern classifier has two phases. They are: *design phase* where abstractions are created; and *classification phase*, where the classification of test patterns is done using these abstractions. Corresponding to these two phases, we have *design time* and *classification time*. In classification based on neighbourhood classifiers, there is no design phase; so, zero design time. However, the classification phase could be computationally expensive.

With an increasing number of new database applications dealing with very large high dimensional data sets, data mining on such data sets has emerged as an important research area. There are a number of different clustering algorithms that are applicable to very large data sets, and a few that address large high dimensional data.

In this paper we propose two algorithms. One is a tree based clustering algorithm which combines CF-Tree[6] and KD-Tree[2] to get moderate space and time complexities. And the other one is combination of clustering with classification using CF-Tree, KD-Tree and KNNC.

1.1 Review of Literature

In this section, we briefly discuss the two tree based clustering algorithms, CF-Tree and KD-Tree. And also a classification algorithm, KNNC.

CF-Tree. ClusterFeature-Tree is based on the principle of agglomerative clustering, that is, at any given stage there are smaller subclusters and the decision at the current stage is to merge subclusters based on some criteria. It maintains a set of *cluster features*(CF) for each subcluster in the form of tree.

Each CF is given by a vector (n, ls, ss) ; where n is the number of data objects in CF, ls is the linear sum of the data objects, and ss is the square sum of the data objects in CF. CF-Tree is a height-balanced tree with two parameters: branching factor B and threshold T . Each nonleaf node contains at most B entries of the form $[CF_i, child_i]$, where $i \in 1, 2, \dots, B$, $child_i$ is a pointer to its i^{th} child node, and CF_i is the CF of the subcluster represented by this child. A nonleaf node represents a cluster made up of all the subclusters represented by its entries. And leaf node contains at most L entries, each of the form $[CF_i]$, where $i = 1, 2, \dots, L$. In addition, each leaf node has two pointers, “prev” and “next” which are used to chain all leaf nodes together for efficient scans. A leaf node also represents a cluster made up of all the subclusters represented by its entries. But all entries in a leaf node must satisfy a threshold requirement, with respect to a threshold value T the diameter has to be less than T .

CF-Tree is a data dependent structure and the parameters T and B need to be tuned. This generally causes increased design times. It has been observed that it takes considerably more space than many other approaches.

KD-Tree. *KD-Tree* is a binary tree structure for storing a finite set of points from a k -dimensional space, generally to handle spatial data, in a simple way. Associated with each internal node N there is a coordinate x and a value v based on which the points are divided into either of the children. Like binary search trees, all points to the left of N will have the x -coordinate less than v and the points to the right will have the greater than (or equal to) that. To construct a KD-Tree for a given set of points, we start with a coordinate and continue until each leaf node satisfies a given set of constraints like all dimensions are used, all the points in the node are similar, etc.

This approach takes lot of time as for every decision we need to scan the whole data.

KNNC. K-Nearest Neighbor (KNN)[5] classification is a very simple, yet powerful classification method. The key idea behind KNN classification is that similar observations belong to similar classes. Thus, one simply has to look for the class designators of a certain number of the nearest neighbors and weigh their class numbers to assign a class number to the unknown. The weighing scheme of the class numbers is often a majority rule, but other schemes are conceivable. k should be kept small, since a large k tends to create misclassifications unless the individual classes are well-separated. It can be shown that the performance of a KNN classifier can be same as at least half of the best possible classifier for a given problem, under certain conditions.

One of the major drawbacks of KNN classifier is that the classifier needs all available data. This may lead to considerable overhead, if the training data set is large.

So in this paper, we combine the above approaches to overcome the drawbacks.

1.2 Outline of the Paper

The paper is organized as follows. Section 2 gives the details about the new clustering algorithm based on CF-Tree and KD-Tree. Section 3 gives description about KNN classification with CF-Tree and KD-Tree clustering. Section 4 has experimental results of our implementations of the algorithms. And we conclude in Section 5.

2 CFKD-Tree Algorithm

CF-Tree algorithm takes more space and less time compared to KD-Tree. So to achieve advantages of both the approaches we propose to combine CF-Tree and KD-Tree algorithms into a new algorithm, CFKD-Tree Algorithm. CFKD-Tree Algorithm has two phases, first phase builds level restricted CF-Tree and the second phase builds a KD-Tree for each leaf node of the CF-Tree. Level restriction can be done by tuning the branching factor(B) and threshold(T). As the leaves of the CF-Tree form smaller clusters, KD-Tree can perform better in terms of running time and at the same time we can achieve space savings. In the KD-Tree phase, we have used variance as the decision parameter for dimension selection. And we take mean as the basis for splitting. See Figure 1.

Fig. 1. CFKD-Tree

```

\ * CFKD-Tree construction for dataset  $\mathbb{D}$  * \
CFKD( $\mathbb{D}$ ,  $\mathbf{B}$ ,  $\mathbf{T}$ )
{
  1. Construct Tree using CF-Tree( $\mathbb{D}$ ,  $\mathbf{B}$ ,  $\mathbf{T}$ )
  2. for each cluster  $C$  of Tree
    – construct KD-Tree( $C$ )
}

```

CFKD-Tree algorithm improves the time and accuracy of KD-Tree algorithm, and takes lesser space than CF-Tree.

CFKD-Tree algorithm can be done in other way also. First we construct a constrained KD-Tree and after getting the clusters, we construct CF-Tree for each cluster. It will take lesser time than former but sometimes gives poor accuracy.

KD-Tree can be constrained by restricting the number of points in each cluster, by fixing the radius of the cluster and taking only few dimensions for splitting.

3 CF-KNNC and KD-KNNC Algorithms

We also propose a combination of a tree based clustering and KNNC with a view to combine advantages of KNNC, CF-Tree and KD-Tree algorithms. KNNC essentially provides good accuracy, however takes longer to run.

3.1 CF-KNNC Algorithm

This algorithm has two phases, design and classification. Where the design phase constructs level restricted CF-Tree, and in classification phase it runs KNNC on the selected leaves of the CF-Tree. To improve the execution time we added a heuristic where KNNC is avoided if all the patterns in a cluster have the same class label. We will find KNNs for only those patterns which belong to border clusters i.e., the clusters which contains data points of multiple classes.

```

\ * Design phase of CF-KNNC * \
CF-KNNC( $\mathbb{D}$ ,  $\mathbf{B}$ ,  $\mathbf{T}$ )
{
  1. Construct CF-Tree( $\mathbb{D}$ ,  $\mathbf{B}$ ,  $\mathbf{T}$ )
}

\ * Classification phase of CF-KNNC * \
\ * Classifying pattern  $p$  using the tree constructed in design phase of CF-KNNC,
and  $k$  is the number of nearest neighbors to be find * \
Classify( $p$ , tree,  $k$ )
{
  1. Find appropriate cluster  $C$  in tree
  2. if all points of  $C$  are in same class
    – return the label of that class
}

```

```

3. else
    - apply KNNC on  $C$ 
    - return the label
}

```

3.2 KD-KNNC Algorithm

This algorithm also has two phases, design and classification. In the first phase we construct a constrained KD-Tree by restricting the number of points in the cluster, by fixing the radius and taking only few dimensions for decision making. In the second phase, we find KNNs for the border clusters as in CF-KNNC.

```

\ * Design phase of KD-KNNC * \
KD-KNNC( $\mathbb{D}$ )
{

```

```

1. Construct constrained KD-Tree( $\mathbb{D}$ )

```

```

}

```

```

\ * Classification phase of KD-KNNC * \
\ * Classifying pattern  $p$  using the tree constructed in design phase of KD-
KNNC, and  $k$  is the number of nearest neighbors to be find * \
Classify( $p, tree, k$ )
{

```

```

1. Find appropriate cluster  $C$  in tree for  $p$ 

```

```

2. if  $C$  is a border cluster

```

```

    - apply KNNC on  $C$ 
    - return the label

```

```

3. else

```

```

    - return the label of that  $C$ 

```

```

}

```

For the above example, Figure 2, we apply KNNC only on clusters 3 and 4.

4 Experimental Results

4.1 Setup

All the experiments are performed on a Intel(R) Pentium(R) 4 CPU 3.20GHz PC with 1GB main memory.

Fig. 2. KNNC with Clustering

Table 1 gives the details of the data sets which have been used for our experiments. KDD is 10% of Network Intrusion Detection Data used in KDDCUP’99. OCR is a digital handwritten data. Sonar is a sonar signals data. Covtype is forest cover type data.

Table 1. Description of Datasets

Dataset	Training Patterns	Testing Patterns	Attributes	Classes
OCR [4]	6670	3333	192	10
SONAR [3]	104	104	60	2
Covtype [3]	15120	565892	54	7
KDD [3]	494021	311029	36	5

4.2 Results

Tables 2, 3, 4 gives experimental results of CFKD, CF-KNN, KD-KNN respectively.

Table 2. Comparison of CF-Tree, KD-Tree and CFKD-Tree Algorithms

Dataset	CF			KD			CF-KD		
	Space (KB)	Time (sec)	Acc. (B, T)	Space (KB)	Time (sec)	Acc.	Space (KB)	Time (sec)	Acc. (B, T)
OCR	40965.5	4.72	85.5386 (18, 2.9)	5060.68	3.13	45.6046	45938.5	14.52	82.4782 (20, 2.5)
SONAR	236.17	0.013	79.8077 (5, 0.69)	28.82	0.012	62.5000	220.43	0.096	78.8462 (35, 0.65)
Covtype	30702.1	403.49	63.2009 (140, 0.3)	3839.0	15.39	63.9056	28850.3	504.74	66.7035 (160, 0.2)
KDD	368301	130.019	97.1986 (11, 1.3)	80787.6	976.22	96.9944	316437	149.37	97.1837 (12,1.12)

Table 3. Comparison of CF-Tree, KNNC and CF-KNNC Algorithms

Dataset	CF			KNNC		CF-KNNC		
	Space (KB)	Time (sec)	Acc. (B, T)	Time (sec)	Acc.	Space (KB)	Time (sec)	Acc. (B, T)
OCR	40965.5	4.72	85.5386 (18, 2.9)	41.37	92.4992	40965.5	4.72	85.6886 (18, 2.9)
SONAR	224.16	0.016	82.6923 (11, 0.65)	0.006	80.7600	128.02	0.035	84.6154 (22, 1.10)
Covtype	30702.1	403.49	63.2009 (140, 0.3)	5298.49	64.1391	27960.6	371.99	63.683 (190, 0.35)
KDD	368301	130.01	97.1986 (11, 1.3)	17271.2	97.1656	362338	156.72	97.4431 (12, 0.7)

Table 4. Comparison of KD-Tree, KNNC and KD-KNNC Algorithms

Dataset	KD			KNNC		KD-KNNC		
	Space (KB)	Time (sec)	Acc.	Time (sec)	Acc.	Space (KB)	Time (sec)	Acc.
OCR	5060.68	3.13	45.6046	41.37	92.4992	5035.4	23.00	85.7486
SONAR	28.82	0.012	62.5000	0.006	80.7600	24.84	0.0026	83.6538
Covtype	3839.0	15.39	63.9056	5298.49	64.1391	3252.99	421.781	64.5084
KDD	80787.6	976.22	96.9944	17271.2	97.1656	75279.2	1026.11	96.9944

5 Conclusions

In this paper, we discussed CF-Tree, KD-Tree and CFKD-Tree clustering algorithms; KNNC, CF-KNNC, KD-KNNC classification algorithms. It is seen that CFKD-Tree takes lesser space than CF-Tree and gives better accuracy than KD-Tree especially for large datasets. KNNC with Tree based clustering reduces the classification time of KNNC. It takes lesser space and provides better accuracy. CF-KNNC gives better accuracy than CF-Tree and much lesser time than KNNC. KD-KNNC gives better accuracy than KD-Tree and much lesser time than KNNC. If the training data is very large, CF-Tree or KD-Tree takes lot of space. That can be avoided at the cost of accuracy. So to handle that type of datasets we can use CF-KNNC or KD-KNNC, which take moderate time and space and with good accuracy. Our results suggest that Hybrid methods perform moderately compared to individual algorithms.

References

1. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. *ACM Comput. Surv.* 31(3), 264–323 (1999)
2. Moore, A.W.: An introductory tutorial on kd-trees (October 8, 1997)
3. Newman, D.J., Hettich, S., Blake, C.L., Merz, C.J.: UCI repository of machine learning databases (1998)
4. Viswanath, P., Narasimha, M., Bhatnagar, S.: Partition based pattern synthesis technique with efficient algorithms for nearest neighbor classification. *Pattern Recognition Letters* 27(14), 1714–1724 (2006)
5. Zhang, B., Srihari, S.N.: Fast K-nearest neighbor classification using cluster-based trees. *IEEE Trans. Pattern Analysis and Machine Intelligence* 26(4), 525–528 (2004)
6. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering method for very large databases. In: Jagadish, H.V., Mumick, I.S. (eds.) *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, Canada, pp. 103–114 (June 4–6, 1996)