Available online at www.sciencedirect.com
SCIENCE DIRECT
Pattern Recognition Letters 27 (2006) 187–200
Pattern Recognition Letters
www.elsevier.com/locate/patrec

Maxdiff kd-trees for data condensation

B.L. Narayan *, C.A. Murthy, Sankar K. Pal

Machine Intelligence Unit, Indian Statistical Institute, 203, B.T. Road, Kolkata 700 108, West Bengal, India

Received 10 May 2004; received in revised form 10 August 2005
Available online 4 October 2005

Communicated by R.P.W. Duin

Abstract

Prototype selection on the basis of conventional clustering algorithms results in good representation but is extremely time-taking on large data sets. kd-trees, on the other hand, are exceptionally efficient in terms of time and space requirements for large data sets, but fail to produce a reasonable representation in certain situations. We propose a new algorithm with speed comparable to the present kd-tree based algorithms which overcomes the problems related to the representation for high condensation ratios. It uses the Maxdiff criterion to separate out distant clusters in the initial stages before splitting them any further thus improving on the representation. The splits being axis-parallel, more nodes would be required for the representing a data set which has no regions where the points are well separated.

Keywords: Prototype selection; Data representation; Multiresolution kd-trees

1. Introduction

Data mining may be viewed as applying pattern recognition and machine learning principles on voluminous (both in size and dimension) heterogeneous data. The existing techniques for pattern recognition usually fail to scale up to large data sets, with the time for analysis becoming a major issue (Provost and Kolluri, 1999). These problems are being handled either by designing algorithms that perform reasonably well on large data sets or by converting the larger data sets into smaller ones which are manageable within the usual pattern recognition framework. In this article, we focus on this second aspect and deal with the problem of reducing the number of samples, which is known as *data condensation*, *data reduction* or *prototype selection*.

The objective of data condensation is to obtain a subset of the original data which contains almost all the information present in the original data set. In other words, a representative sample set (or a set of prototypes) of the original data needs to be obtained. There are several approaches for obtaining prototypes from a given data set. The *condensed nearest neighbor rule* (Hart, 1968) condenses the data set only from the perspective of the nearest neighbor algorithm. The obtained set of points results in the same nearest neighbor rule as the one obtained using the complete data set. In this respect, proximity graphs have also been employed (Sanchez et al., 1997) where the data points are first edited before performing the condensation. In general, it is desirable to select prototypes that would be useful for all sorts of mining tasks. Randomly selecting prototypes may result in a loss of information regarding the structure present in the data unless the size of the randomly sampled set is quite large (Catlett, 1991). On the other hand, selecting prototypes randomly after clustering the data set would preserve the clustering information to a greater extent. Such an approach for prototype selection may be found in (Decaestecker, 1997), where gradient descent and deterministic annealing are used to find training

* Corresponding author. Tel.: +91 33 2575 3109; fax: +91 33 2577 6680.
E-mail addresses: bln_r@isical.ac.in (B.L. Narayan), murthy@isical.ac.in (C.A. Murthy), sankar@isical.ac.in (S.K. Pal).

0167-8655/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2005.08.015

set consistent subsets. The CFF algorithm provided by Cuevas et al. (2000) finds the subset of the data belonging to a level set and computes agglomerative clusters.

It may be noted that the clustering obtained for the purpose of condensation need not be the clustering in the usual sense, i.e., for the sake of condensation, it is acceptable to have more number of clusters than are actually present in the given data. For example, given a data with three apparent clusters, it is acceptable to have twelve clusters by splitting each of the original clusters into four subclusters and thus obtaining twelve prototypes. Henceforth, by clustering, unless otherwise stated, we would be referring to clustering for the purpose of condensation and not obtaining the exact number of clusters present in the data.

kd-trees (Friedman et al., 1997) (short for k dimensional binary search trees) are used for condensation due to the above mentioned reason. kd-trees are formed by splitting the data at each node along a particular dimension using some criterion. Each node owns a hyperrectangle that covers the data at that node. The split corresponds to partitioning the hyperrectangle of the node with each part going to a child node. Multiresolution kd-trees (mrkd-trees) (Moore, 1999) store the first and second order summary statistics of the data at each node (Moore and Lee, 1998). Each manner of truncating the tree results in a particular representation of the original data. Generally, the midpoint or the median along the most spread dimension is chosen for splitting a node. kd-tree based condensation is essentially very fast as each split can be performed in $O(kn)$ time, where n is the number of data points at the current node. Thus, a kd-tree can be formed in $O(kN\log m)$ time, where N and m are the number of data points in the original data set and the number of terminal (or leaf) nodes, respectively. Hierarchical clustering methods are relatively slower as they optimize some objective function while splitting one of the existing clusters at each stage.

As the number of clusters increases, the representation of the original data by the selected prototypes is expected to be increasingly better and when the number of clusters is equal to the number of data points, the representation is the best possible one. All data condensation algorithms satisfy the above. The available algorithms differ from each other in terms of the amount of condensation obtained for producing the same quality of representation.

We mention some other methods whereby a representation similar to that by kd-trees may be obtained. AD-trees store the data set in terms of all the possible feature values (Moore and Lee, 1998). Tree structured vector quantizers (Balakrishnan et al., 1995) obtain prototypes at multiple levels, and speed up the codebook search by searching through a tree structure.

The amount of condensation is measured in terms of condensation ratio which is defined as $\frac{N-m}{N}$. As mentioned earlier, for lower condensation ratios, most of the algorithms tend to result in similar quality of representation. In what follows, we show that the kd-tree based representation may not be appropriate for high condensation ratios (though it is acceptable for lower condensation ratios). When the given data has an inherent clustering in it, and the clusters possess neither equal size nor do they contain equal number of data points, the initial splits while forming a kd-tree are performed in such a manner that some of the nodes cover portions of two or more clusters. The summary statistics stored at those nodes are, therefore, inapt.

The above drawback of the kd-tree based condensation algorithms prompts us to describe a new kd-tree based condensation algorithm where the splits are performed at the positions with the largest difference in consecutive order statistics (or the sorted data values) along a particular dimension. Since finding the exact splitting position (or pivot) involves $O(n\log n)$ operations (for each dimension), we suggest an approximate way of finding the pivot in $O(n)$ time. The time and space complexity of the proposed algorithm is the same as that of the usual kd-trees. We show, experimentally, that the performance of the proposed algorithm is better than other competitive algorithms whenever the data has well-separated clusters with different sizes (i.e., the sizes of the regions occupied by the clusters) and unequal a priori probabilities (i.e., the number of points contained in them).

In the next section, we discuss in detail how kd-trees can be employed for condensed representation of data. We also provide examples to show why the existing methods of splitting may not be appropriate. In Section 3, we describe the proposed algorithm and explain its advantages over other methods. We provide the experimental results and a critical analysis of our algorithm in Section 4. Finally, we conclude the article with some discussion regarding the scope for improvement over the proposed method.

2. Condensed representation with kd-trees

The objective of a data condensation algorithm is to take as input, an original data set and output a smaller representative data set of the input. There is a trade-off between the size of the representative data set and its faithfulness. The larger the size of the condensed set, the better is its representation of the original data set.

As mentioned earlier, the cluster centroids (or medoids) may be considered to constitute the condensed data set. The problem of data condensation, however, is not the same as clustering, where it is necessary to have only as many clusters as are present in the data. In data condensation, one can afford to, and often needs to have many more representative points in the condensed set than the inherent number of clusters. This is so as a single point need not necessarily represent the cluster properly owing to the various shapes a cluster can have. CURE (Guha et al., 2001) chooses multiple prototypes from each cluster, but the complexity of each iteration of the algorithm is high, as is the case with the conventional clustering algorithms. These are good at clustering when the number of clusters and the number of features are small, otherwise, they fail to provide the output in a reasonable amount of time. This renders partitional clustering algorithms infeasible for data condensation.

Hierarchical clustering algorithms are faster than partitional clustering algorithms. Moreover, it is not necessary to specify the number of clusters to be obtained. Instead, a stopping criterion has to be specified for terminating the splitting/merging process. Hierarchical clustering can again be either divisive or agglomerative (Zhao and Karypis, 2002; Karypis et al., 1999). The objective for condensation being obtaining as small a representative set as possible from a large data set, agglomerative algorithms would take much longer to provide the desired condensed set. Although graph based clustering algorithms can detect clusters of arbitrary shapes and sizes and may be considered for hierarchical clustering, they have a high complexity owing to the complex decision process involved for splitting/merging at each stage. This makes the application of such techniques on large data sets unsuitable as the time taken is too high for data mining applications. In this context, it is desirable to have a hierarchical clustering algorithm which needs very little time, say, O(kn), for splitting/merging.

The above requirement brings to the fore the need for using a kd-tree based algorithm for data condensation. A kd-tree is a data structure that stores a given data set through its partitioning. It is a binary search tree where the data belongs to $\mathbb{R}^k$. Each node of the tree contains some information about a part of the data set. The root of the tree contains the information about the entire data. The data at each node is further partitioned into two parts, and the information about each of them is stored in a child node. The information stored at a node may be one or more of the following summary statistics, namely, the number of data points, the centroid, the covariance, the bounding hyperrectangle, the parameters of the best fitting distribution, and so forth (Moore and Lee, 1998; Zhang et al., 1996; Wang et al., 1997).

The partitioning of the data is always performed parallel to the coordinate axis. A dimension is first chosen, usually on the basis of maximum spread or variance. Then, along the chosen dimension, a pivot is computed. Usually, this pivot may be the median or the midpoint (of the maximum and the minimum) of the data along the selected dimension. All points with values less than the pivot in the selected dimension are put into the left child node and the rest into the right child node. The partitioning process proceeds till such time when it is deemed unnecessary to split the node anymore, either due to all the points being very close to each other or the number of points at the node being too few. The computation of the pivot involves O(kn) operations (where n is the number of data points at the current node), whereby, the construction of the kd-tree takes O($kN\log m$) time. Usually, some summary statistics are stored at that node such that a reasonable representation of the data can be obtained. Once the tree is formed, the data points contained in each leaf node may be replaced by a single point (or prototype), thus reducing the size of the data set. Various representations of the original data set can be obtained by truncating the kd-tree in different ways.

In general, the above mentioned choices of pivots are quite efficient, but, when the data is well-clustered with the clusters having unequal a priori probabilities and different sizes, the split performed by kd-trees leads to a loss of information about this structure. The basic drawback of dividing the data on the basis of the median (or, for that reason, any other measure of central tendency) is that, when cluster sizes are different, a leaf node can consist of portions of two well separated clusters, thus making the representation poorer. By choosing the median as the pivot for splitting, it is being implicitly assumed that the clusters consist of equal number of data points, which may not be the case, in general. In the case of choosing the midpoint as the pivot, an assumption is being made regarding the equal sizes of the clusters.

As an example, we consider a small synthetic data set with 90,000 data points and two features. The data consists of four well-separated clusters. For the purpose of plotting (Fig. 1), we have reduced the number of data points to 9000. The original data set looks roughly the same except that it is denser and the number of points in the overlapping region between the neighboring Gaussian subclusters is higher. The condensed representations obtained by the kd-tree based methods are shown in Figs. 2 and 3 with the number of leaf nodes taken to be 4, 16, 32 and 256, where the midpoint and median, respectively, are used for splitting. A Gaussian is fitted to the data in each leaf node and is plotted as an ellipse. The lengths of the major and minor axes are three times the standard deviation along the corresponding dimensions. For the sake of clarity, rectangles (cells) corresponding to the leaf nodes are shown only in Figs. 2a and 3a.

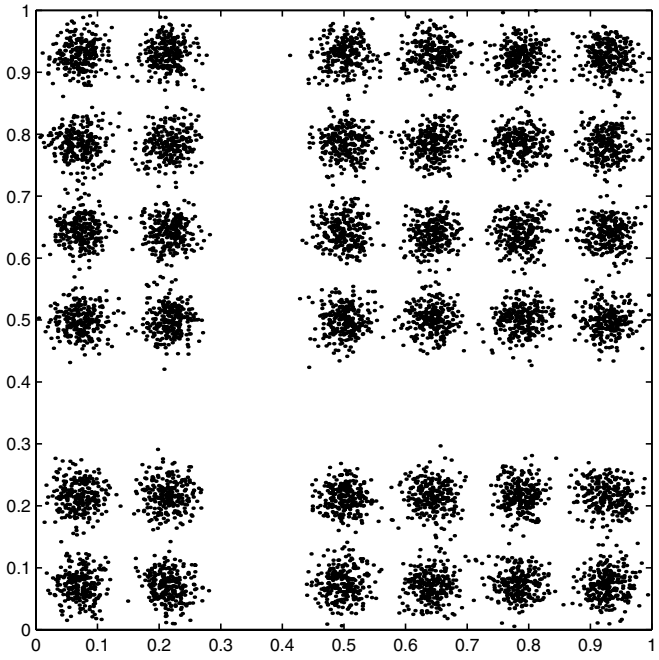

Fig. 1. Synthetic data set.

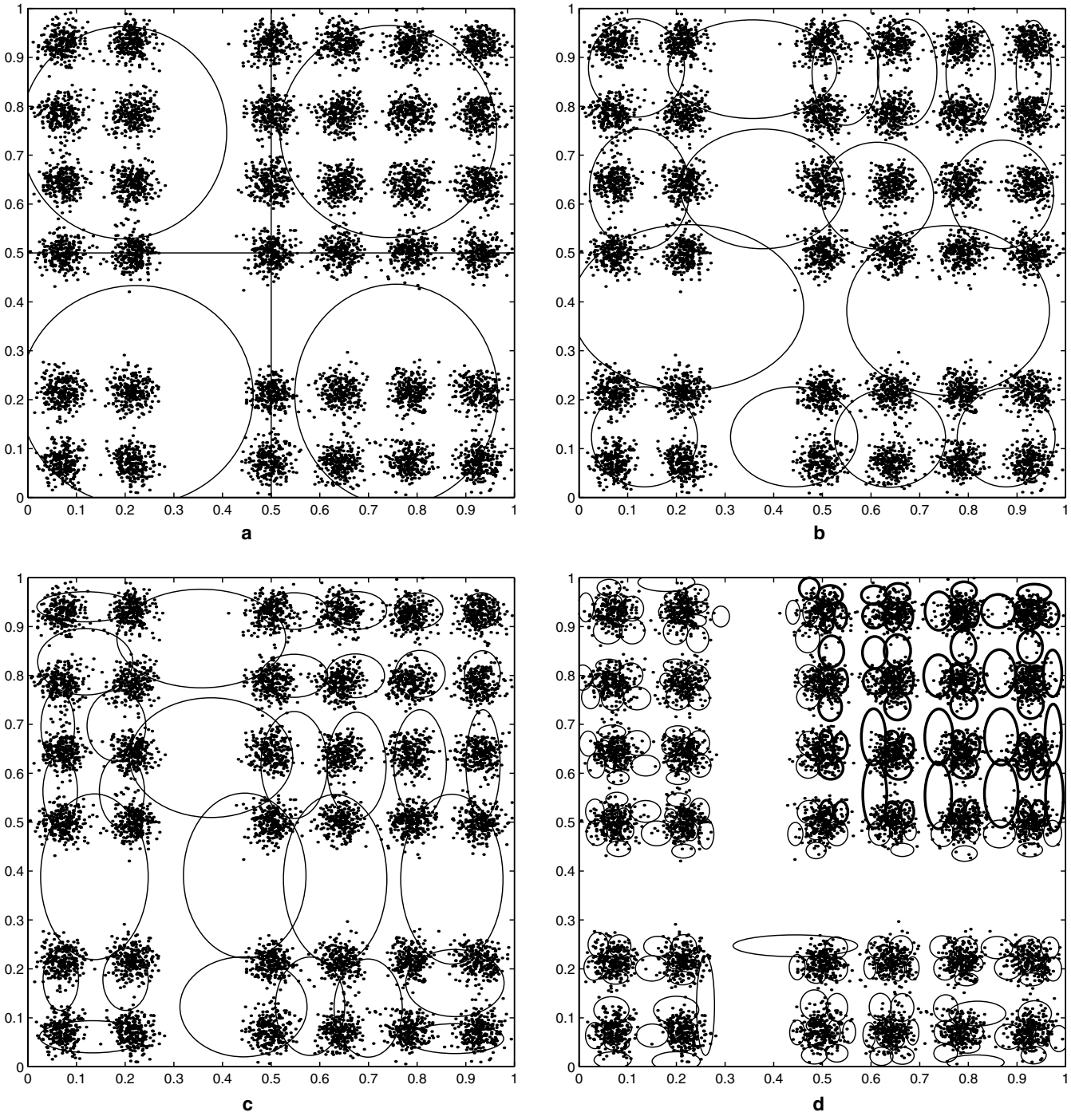

Fig. 2. Clusters obtained by the kd-tree based method for the synthetic data set using the midpoint for splitting with the number of leaf nodes equal to: (a) 4, (b) 16, (c) 32 and (d) 256.

It is seen from Figs. 2 and 3 that the resulting clusters (cells) do not reflect the inherent structure of the input data set. For example, some of the cells (or nodes) comprise two or more groups of points that are apart from each other.

With the increase in the number of splits, the clustering information that is initially lost is gradually regained as the partition is refined. This is seen from the sequences Fig. 2a–d and Fig. 3a–d. Obviously, the initial choice of splits has an impact on the subsequent condensed representations. For example, Fig. 2a has better splits than Fig. 3a, thereby producing a better representation of clusters in Fig. 2d as compared to that in Fig. 3d. This necessitates the proper selection of the initial splits.

Let us now consider Fig. 4, as an example. Here, the initial splits (Fig. 4a) are so chosen that the natural grouping of the data remains intact. As a result, the subsequent representations obtained are also better than those in Figs. 2 and 3. In the next section we describe an algorithm that

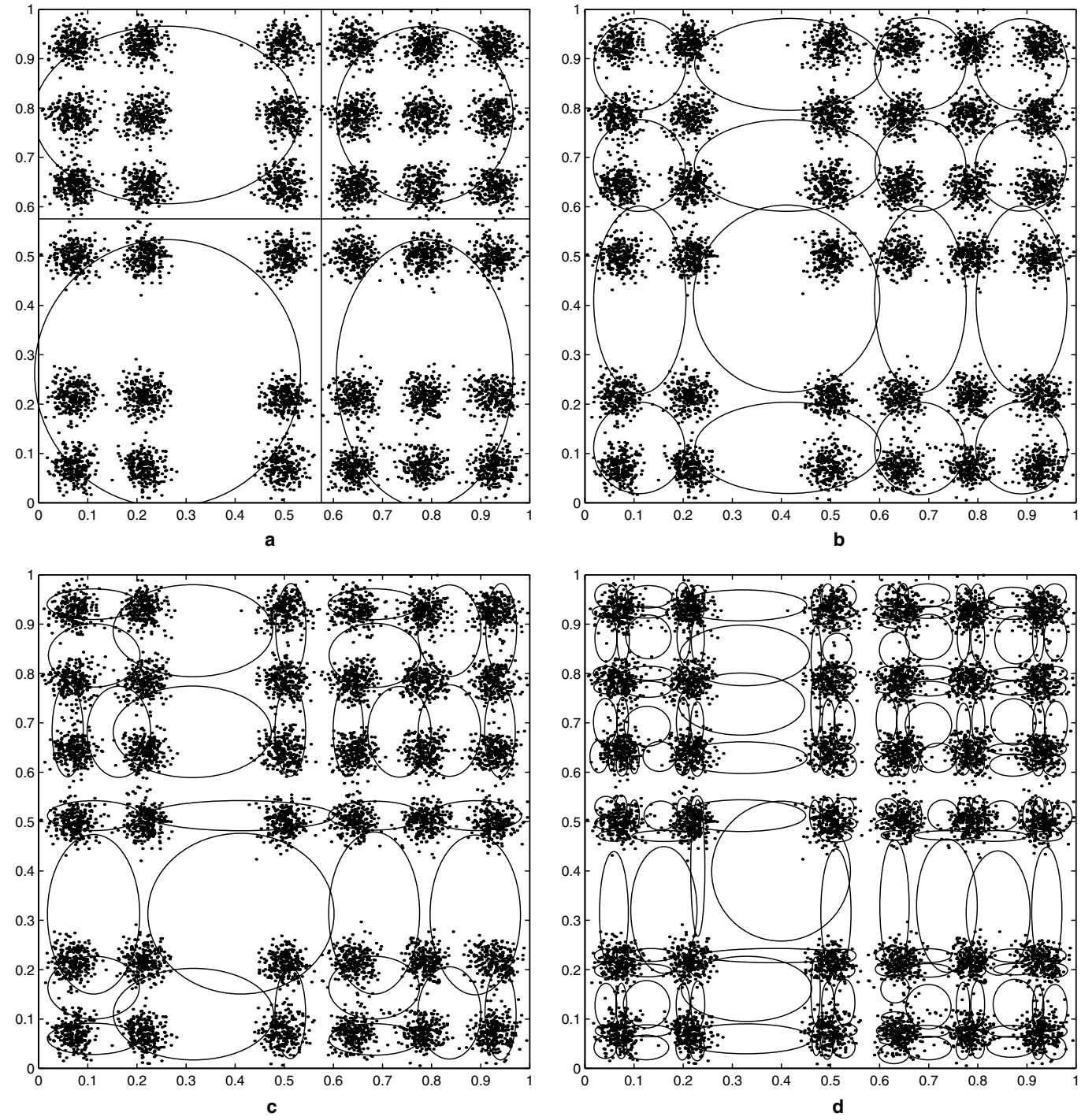

Fig. 3. Clusters obtained by the kd-tree based method for the synthetic data set using the median for splitting with the number of leaf nodes equal to: (a) 4, (b) 16, (c) 32 and (d) 256.

can achieve this characteristic in order to represent the data set for high condensation ratios.

3. Hierarchical data condensation technique

Before laying out the proposed algorithm, we provide the intuition behind it. Then, we discuss its implementation aspects.

3.1. Principle

Any given function can be approximated by a mixture of Gaussians (Broomhead and Lowe, 1988). This approximation improves monotonically as the number of component Gaussians is increased. Multimodal data, especially when the modes are well apart, cannot be satisfactorily approximated by a single Gaussian function. The data around each

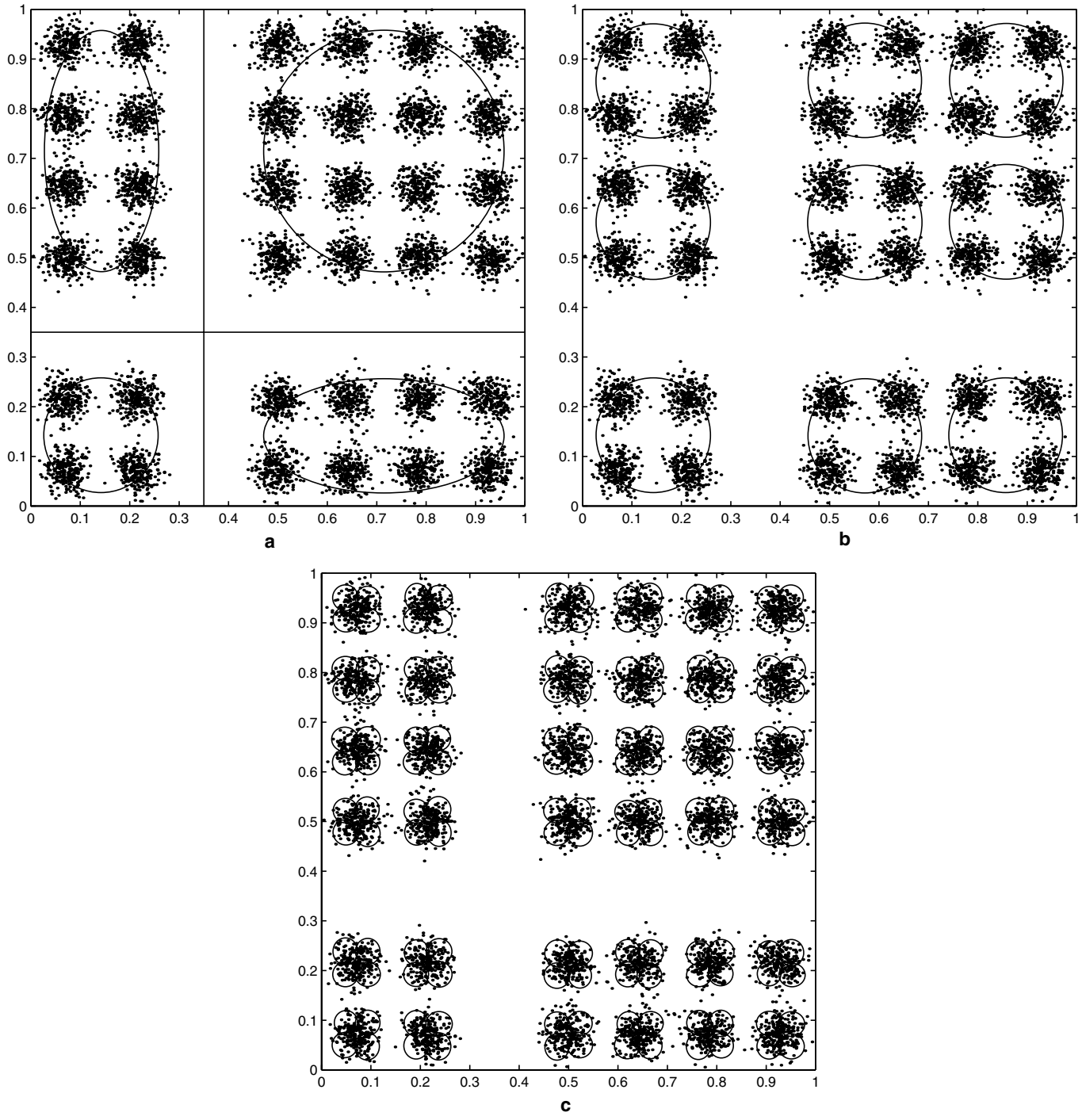

Fig. 4. Clusters obtained by the proposed method with the number of leaf nodes equal to: (a) 4, (b) 9 and (c) 144.

mode, however, can be reasonably represented by a Gaussian function. Therefore, for the purpose of representation of a multimodal data by Gaussians, the patterns around the different modes should be separated first and the summary statistics be obtained from each of the clusters. Valley-seeking (Fukunaga, 1990) is widely used for isolating such modes while clustering. This procedure, however, is extremely time consuming and, therefore, not suitable for large data sets.

In the proposed methodology, we employ an approximate version of the valley-seeking algorithm. This involves determining the position along each dimension where the difference between the values of consecutive data points (when ordered along that dimension) is maximum. Let Maxdiff denote the maximum of such differences over all the dimensions. Maxdiff is considered to be significant if its value exceeds some pre-determined threshold t_0. If it is significant, it indicates multimodality and the data in the

node is split about the Maxdiff position. Otherwise, splitting is performed by taking the mean along the dimension with maximum variance as the pivot. The splitting is terminated when the maximum variance along each dimension is found to be less than another threshold, say t_1.

We now present the algorithm in detail.

3.2. Algorithm

(1) Input number of features (k), number of data points (N) and an $N \times k$ data matrix $X = ((X_{ij}))_{i=1,\ldots,N,\ j=1,\ldots,k}$. The ith row of X consists of the feature values of the ith sample.
(2) Input thresholds t_0 and t_1.
(3) Normalize X.
(4) Create a root node that contains all the data points.
(5) Find the maximum difference between consecutive order statistics with respect to feature i (i.e., $\text{Maxdiff}_i = \max_j\{X_{(j+1),i} - X_{(j),i}\}$), $i = 1,2,\ldots,k$, and note its position (i.e., $\text{Pos}_i = \text{argmax}_j\{X_{(j+1),i} - X_{(j),i}\}$). Here, $X_{(j),i}$ denotes the jth element, after the X values are sorted in ascending order along the ith feature. Determine the maximum of such differences over all the features ($\text{Maxdiff} = \max_i\{\text{Maxdiff}_i\}$). Compute the splitting dimension, $F = \text{argmax}_i\{\text{Maxdiff}_i\}$. Then, $X_{(\text{Pos}_F),F}$ represents the value of the Pivot.
(6) If Maxdiff criterion is satisfied, (that is, if Maxdiff $\geqslant t_0$), decide to split the current node. The left child node contains the points $\{X_i : X_{i,F} \leqslant \text{Pivot}\}$ and the right child contains the remaining data points.
(7) Else (i.e., data is not well separated), check if the variance of the data along some feature is greater than the threshold t_1. If yes, split node with respect to the mean of the feature with highest variance. Else, declare that the node cannot be split further.
(8) Repeat Steps 5, 6 and 7 for each newly formed node.

We now provide an example, for the sake of illustration, where the proposed algorithm is applied to the synthetic data (Fig. 1) with t_0 and t_1 both chosen to be 0.1. The data has two features, say X and Y. All the points are first arranged in increasing order of their X values. The Maxdiff_X value is computed as the maximum difference between two consecutive X-values. It turns out to be 0.16 in this case, and occurs at the location. Similarly, Maxdiff_Y is computed by considering the Y values of the data points, which happens to be 0.155. Maxdiff is taken to be the maximum of the two and equals 0.16 and is greater than t_0, and hence the data points would be split into two parts. Since $\text{Maxdiff}_X > \text{Maxdiff}_Y$, we choose X as the dimension for splitting. So, all points with X-values less than 0.35 (points to the left of the vertical line in Fig. 4a) are kept in one node, whereas, the rest are stored in a different node. The above process is repeated on these two nodes and, this time, both of them are split on the basis of the Y values, because Maxdiff_X is close to 0 for both of them. The four nodes obtained in this manner are shown in Fig. 4a. No further splitting is possible either on the basis of the Maxdiff or variance. So, the algorithm terminates.

We now discuss the implementation aspects.

3.3. Implementation aspect

Here, we explain some of the computational aspects, namely, normalization, computation of Maxdiff, computing t_0, robustness to outliers and the complexity of the algorithm.

3.3.1. Normalization

The algorithm being very much dependent on the spread of the feature values, we first normalize them by dividing the values of each feature by its range. This is performed only once, at the beginning of the algorithm.

3.3.2. Computation of Maxdiff

Computing Maxdiff involves determining Maxdiff_i for each $i = 1,\ldots,k$. Let us now consider a particular feature axis i for the following discussion. The computation of Maxdiff_i is based on the order statistics and it requires sorting of data along the ith feature axis. Since it is time consuming, we find an approximate value ($\text{Maxdiff}_i^{(\text{H})}$) of Maxdiff_i by forming a histogram along ith axis with a large number of equal sized bins and then looking for the location of the maximum number of consecutive empty bins. $\text{Maxdiff}_i^{(\text{H})}$ is calculated as

$$\text{Maxdiff}_i^{(\text{H})} = (\text{Maximum number of consecutive empty bins}) \times (\text{bin width}). \quad (1)$$

This idea is similar to that of the Maxdiff histograms described in (Poosala, 1997), except that we fix the number of bins to be more than the number of points. This keeps the computational complexity of the proposed scheme the same as that of kd-trees (as finding $\text{Maxdiff}_i^{(\text{H})}$ involves $O(n)$ operations). Note that, higher the number of bins, the better the approximation.

If n is the number of data points in a given node, and a histogram with $n+1$ bins is formed, then, by the pigeonhole principle, at least one bin would be empty. Moreover, the following inequality holds:

$$|\text{Maxdiff}_i^{(\text{H})} - \text{Maxdiff}_i| < \text{Range}_i * \frac{2}{n+1}, \quad (2)$$

where Range_i is the range of the ith feature values of the data contained in the node under consideration. This inequality has the following implication. If the maximum number of consecutive empty bins in the histogram is M and there are no other $M-1$ consecutive empty bins, then the position of $\text{Maxdiff}_i^{(\text{H})}$ is the same as that of Maxdiff_i.

3.3.3. Computing t_0

t_0 is the threshold considered for checking the significance of Maxdiff_i. Equivalently, since Maxdiff_i and M are related, we can consider a threshold, say t'_0, for checking the significance of M. If the data is uniformly distributed, then the histogram would have almost all the bins occupied and a few bins empty (i.e., M is small). On the other hand, if a large number of consecutive bins are empty (i.e., M is large), with bins on both sides occupied, then a multimodal distribution becomes likelier than a unimodal distribution.

Let us assume that the data set arises from a uniform distribution and that there are $n+1$ bins in the histogram. Under this assumption, the probability of having a particular set of M consecutive empty bins, denoted by p_M, is equal to $\left(1-\frac{M}{n+1}\right)^n$. Note that, if $M \ll n$, and n is large, then $\left(1-\frac{M}{n+1}\right)^n \approx \left(1+\frac{1}{n+1}\right)^{-Mn} \approx e^{-M}$.

Threshold t'_0 should be such that the above probability is very small if $M > t'_0$. Given a value ϵ, $p_M < \epsilon$ is equivalent to $M > -\log\epsilon$. So, t'_0 may be chosen to be $-\log\epsilon$ where ϵ is a predetermined error value (or p-value). This threshold is seemingly free of n because of having the number of bins almost the same as the number of data points. The probability, however, is not free of n as we have assumed that $M \ll n$.

3.3.4. Robustness to outliers

To make the algorithm more robust to outliers and also to preempt the formation of a degenerate tree, we split a node only if each of the child nodes inherits at least α% of the number of points at the present node. When the number of points in a node is small enough (i.e., $n < 100/\alpha$), the restriction no longer holds as $\frac{\alpha}{100n}$ becomes 0. Note that this does not add to the complexity of the algorithm since the search for consecutive empty bins starts after the total count of points in the bins to the left becomes greater than $\frac{\alpha}{100n}$, and proceeds till this count reaches $n - \frac{\alpha}{100n}$. In our experiments, we have chosen the value of α to be 5.

3.3.5. Complexity

The complexity of the algorithm is $O(kN\log m)$. This is so because we need to go down the tree only an O(1) number of levels (assuming that the number of well-separated clusters is less than $\log m$), performing $O(kN)$ number of operations at each level, before the tree structure becomes the same as that for the usual kd-trees.

4. Experimental results

We have conducted extensive experiments to demonstrate the aforesaid features of the algorithm and its superiority in comparison to other competitive algorithms. Five data sets are chosen for our experiments. Among them, two are real life data sets (forest covertype and multiple features) and the remaining three have been artificially generated by us. The description of the data sets is as follows:

- Synthetic: A simple two-dimensional data set to show the motivation behind the proposed algorithm (Fig. 1). It consists of 90,000 data points. There are four well separated clusters of different sizes. The number of points in the four clusters are 10,000, 20,000, 20,000, and 40,000. Each cluster consists of Gaussian subclusters of size 2500. All the Gaussian subclusters have the same variance–covariance matrix. During our experiments, we have also treated each of the Gaussian subclusters as a different class (that is, we considered the data set to have 36 classes).
- Forest covertype: This data, collected by the Department of Forest Sciences of Colorado State University, consists of 581,012 points with 54 feature values. Of these, the last 44 feature values are binary. Binary features are invariant under normalization, and hence these features always have the Maxdiff property. Any of these features may be randomly chosen as the splitting dimension. Hence, all the initial splits would be performed in terms of the binary features only, and the results would vary with the order of the chosen binary features. So, we have made use of only the first 10 feature values, which are all real-valued. The objective is to classify the samples into 7 land cover types. About 80% of the patterns is concentrated in just two of the classes. Classes are not well separated. This data set is available from the UCI KDD Archive (URL: http://ftp.ics.uci.edu/pub/machine-learning-databases/covtype).
- Multiple features: Digit data set consists of handwritten numerals extracted from a collection of Dutch utility maps. There are ten classes, each with 200 patterns and 649 features. Classes are well separated along certain features. This data set too, is available from the UCI KDD Archive (URL: http://ftp.ics.uci.edu/pub/machine-learning-databases/mfeat/).
- Twonorm: Data set containing 100,000 points with 20 features is generated from an equal mixture of two multivariate normals with means μ and $-\mu$, and unit variance–covariance matrix, where $\mu = \left(\frac{2}{\sqrt{20}}, \frac{2}{\sqrt{20}}, \ldots, \frac{2}{\sqrt{20}}\right)'$. We have generated this data set because the earlier twonorm data (Breiman, 1998) has only 7400 data points which is too small. Misclassification rate with respect to the Bayes decision rule for each of the 2 twonorm data sets has been theoretically computed as 2.3% (Breiman, 1998).
- Shapes: This is an artificially generated data set of size 100,000 with 16 features containing different clusters of various shapes. The two-dimensional projections of the various shapes are shown in Figs. 5 and 6. The constituent shapes, their positions and the number of data points in them are summarized in Table 1. The coordinates of the centers along all the remaining 13 axes have been taken to be 0 and hence the shapes are separable only with the first three features. The second set of shapes has been superimposed on the first set along

the third dimension (Fig. 7). Five of the seven clusters are well separated.

These data sets have been normalized prior to their use. Normalization involves obtaining the range of the data along each feature and dividing by it.

In order to measure how good the representation is, we have computed, for various values of condensation ratios, the entropy and purity values for the resulting clusters. The entropy and purity at leaf node S_j are defined as

$$E(S_j) = -\frac{1}{\log q}\sum_{i=1}^{q} p_i^{(j)} \log p_i^{(j)}$$

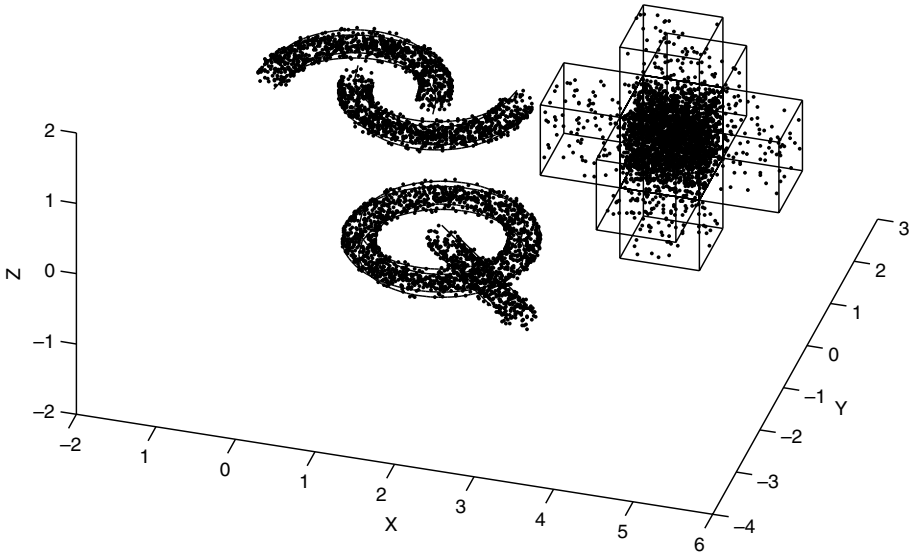

Fig. 5. Three dimensional projection of the lower portion of the shapes data set.

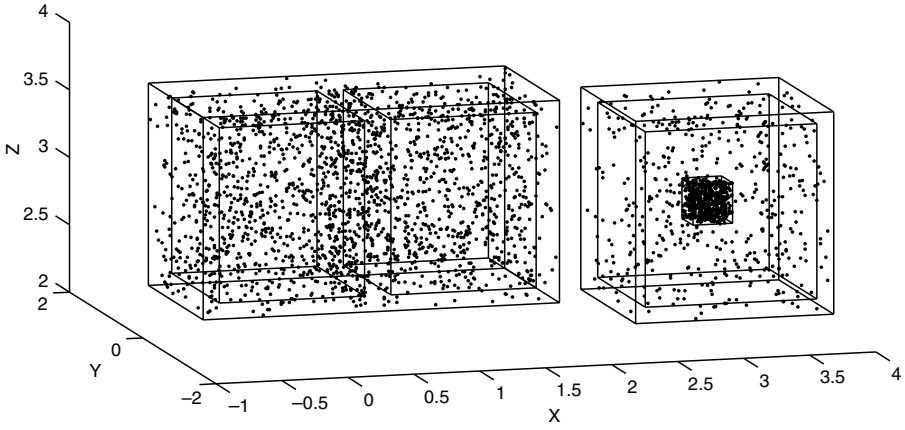

Fig. 6. Three dimensional projection of the upper portion of the shapes data set.

Table 1
Constituents of the shapes data set (100,000 × 16)

Class	Shape	Center			Length/radius	Width	No. of points	Comments
		X	Y	Z				
0	Horseshoe	0	1	0	1	0.2	10,000	Top
1	Horseshoe	1	1.2	0	1	0.2	10,000	Bottom
2	Hypercross	4	1	0	1	–	30,000	33 Hypercubes
3	Q-shape	2	−2	0	1	0.2	20,000	Ring + cylinder
4	Two hollow hypercubes	0	0	3	1.5	0.3	20,000	Two holes
5	Hollow hypercube	3	−1	3	1.5	0.2	5000	One hole
6	Hypercube	3	−1	3	0.3	–	5000	Within class 5

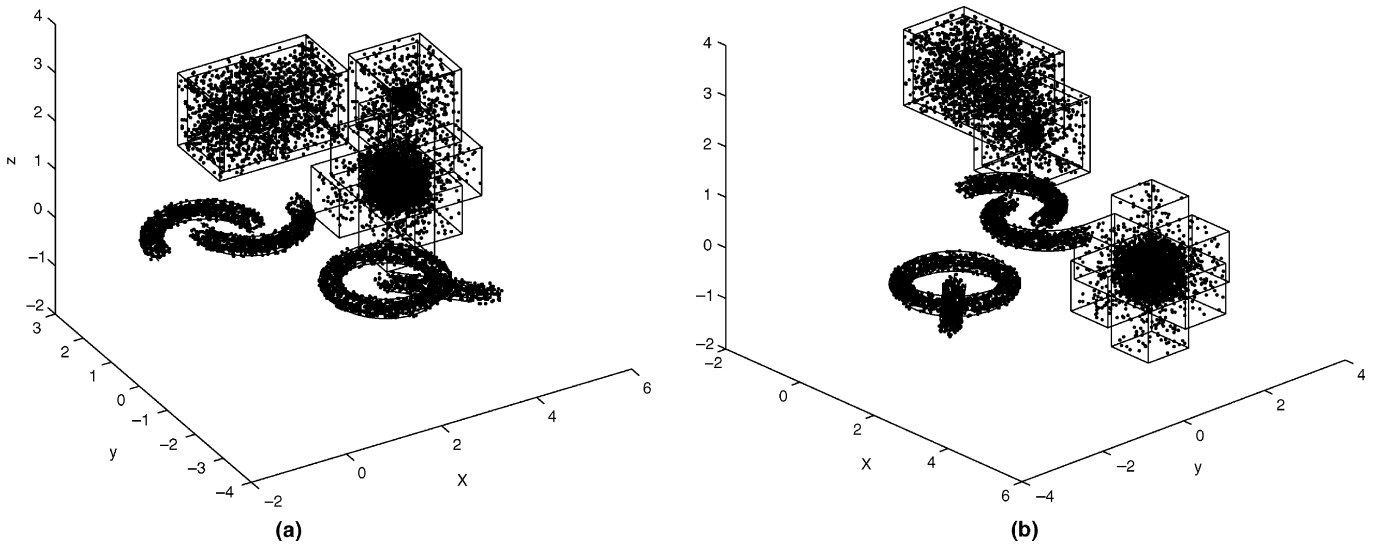

Fig. 7. Three dimensional projections of the shapes data set.

and

$$\Pi^{(j)} = \max_{1 \leqslant i \leqslant q} p_i^{(j)},$$

respectively, where, q is the number of classes present in the data set and $p_i^{(j)}$ is the proportion of points from the ith class at the jth leaf node.

The entropy values at all the leaf nodes can be combined into a single value of entropy ($E(T)$) as

$$E(T) = \sum_{j=1}^{m} \frac{N_j}{N} E(S_j),$$

where, N_j is the number of points at the jth leaf node, and m is the total number of nodes in the tree T.

The purity values can be combined similarly.

$$\Pi = \sum_{j=1}^{m} \frac{N_j}{N} \Pi^{(j)}$$

The purity and entropy values signify the amount of confusion at each node. If most of the points at a node belong to the same class, the purity for that node is high while the entropy is low. In such a case, the chosen prototype can be said to be a better representative of the points in that node. On the other hand, if a node consists of data from several classes, the summary statistics or the prototype at that node would not be quite appropriate and this would be reflected by the high entropy and low purity values.

We have provided the comparisons with two kd-tree based algorithms, one employs the median for splitting and the other uses the midpoint of the maximum and minimum of the data. The performance of our algorithm is also compared with the results obtained by the software CLUTO (version 2.1, August 2002). We have run only the partitional clustering algorithms provided in the package that are comparable to the suggested algorithm in terms of the time and space complexity. These algorithms formulate the clustering as an optimization problem and have been shown to provide superior performance for high-dimensional data sets (Zhao and Karypis, 2002). The methods that have been chosen are "Repeated Bisection" (RB) and "Direct" with the similarity measure taken as "cosine". As the names suggest, the Direct method obtains the direct partitioning, whereas the Repeated Bisection method bisects one of the current clusters, thus increasing the number of clusters by one each time. The optimizing criterion is chosen to be $\mathbb{I}_2 = \sum_{i=1}^{k} \sum_{d_i \in S_r} \cos(d_i, C_r)$ (Zhao and Karypis, 2001), where C_r is the centroid of the cluster S_r. $\mathbb{I}_2$ has been shown to be an extremely good choice in (Zhao and Karypis, 2001). Some other direct clustering options are available in CLUTO, which differ from the above in terms of the criterion functions being used (Zhao and Karypis, 2001). One such criterion measures the similarity in terms of correlation instead of the cosine.

It may be noted that the condensation ratios mentioned in the first column of each table are approximations. For example, the condensation ratios of 99.99%, 99.90% and 99.00% for the forest covertype data indicate that the condensed sets consisted of around 60, 600 and 6000 points, respectively.

We make the following observations from the results shown in Tables 2–7:

- The results on the synthetic data set (Tables 2 and 3) bring out the positive aspects of our algorithm. We have considered two cases here. When the number of classes in the original data has been taken to be 4, the proposed algorithm had immediately (that is, after three splits), managed to separate out each of the individual classes. Then it goes about splitting in the usual kd-tree like manner. For 36 classes too, the splitting is performed in exactly the same manner. The first three splits result in distinguishing between the four groups of subclusters

Table 2
Entropy and purity values for the synthetic data set

C.R. (%)	Measure	Proposed	kd-tree		RB	Direct
			Midpoint	Median	Cos	Cos
99.995	Entropy	0.000000	0.296187	0.459148	0.315925	0.324103
	Purity	1.000000	0.844700	0.694411	0.866389	0.861089
99.990	Entropy	0.000000	0.212901	0.287021	0.290798	0.297284
	Purity	1.000000	0.869600	0.805311	0.866389	0.862600
99.980	Entropy	0.000000	0.136256	0.229567	0.274186	0.274911
	Purity	1.000000	0.898256	0.840311	0.867411	0.875233
99.965	Entropy	0.000000	0.118930	0.129321	0.266798	0.266811
	Purity	1.000000	0.905111	0.909433	0.875089	0.872156
99.720	Entropy	0.000000	0.002020	0.032130	0.258943	–
	Purity	1.000000	0.998745	0.977145	0.876056	–

Table 3
Entropy and purity values for the synthetic data set (36 classes)

C.R. (%)	Measure	Proposed	kd-tree		RB	Direct
			Midpoint	Median	Cos	Cos
99.995	Entropy	0.644755	0.684629	0.620069	0.665241	0.661717
	Purity	0.111111	0.111111	0.111111	0.111111	0.111111
99.990	Entropy	0.393711	0.498080	0.491094	0.530422	0.532435
	Purity	0.250000	0.250222	0.221567	0.220644	0.220767
99.980	Entropy	0.272053	0.415538	0.362149	0.448190	0.445646
	Purity	0.467878	0.346667	0.441644	0.343041	0.354356
99.965	Entropy	0.027762	0.288461	0.273382	0.409388	–
	Purity	0.983611	0.533422	0.551522	0.411956	–
99.720	Entropy	0.024437	0.043149	0.083838	0.385748	–
	Purity	0.983611	0.939600	0.855189	0.437422	–

Table 4
Entropy and purity values for the shapes data set

C.R. (%)	Measure	Proposed	kd-tree		RB	Direct
			Midpoint	Median	Cos	Cos
99.995	Entropy	0.106839	0.241239	0.427238	0.193112	0.263869
	Purity	0.848731	0.763460	0.581940	0.840170	0.799270
99.990	Entropy	0.100798	0.220158	0.260392	0.191196	0.257943
	Purity	0.884010	0.770730	0.756310	0.840170	0.799710
99.980	Entropy	0.041009	0.083621	0.189572	0.166943	0.248269
	Purity	0.946665	0.917010	0.798290	0.853050	0.804520
99.700	Entropy	0.040998	0.035684	0.062650	0.085124	–
	Purity	0.946669	0.958380	0.937730	0.928610	–

Table 5
Entropy and purity values for the twonorm data set

C.R. (%)	Measure	Proposed	kd-tree		RB	Direct
			Midpoint	Median	Cos	Cos
99	Entropy	0.631685	0.428959	0.430268	0.149530	–
	Purity	0.829389	0.867729	0.868521	0.976450	–
90	Entropy	0.280813	0.283638	0.288154	0.097862	–
	Purity	0.910843	0.908509	0.907166	0.976568	–

Table 6
Entropy and purity values for the multiple features data set

C.R. (%)	Measure	Proposed	kd-tree		RB	Direct
			Midpoint	Median	Cos	Cos
99.50	Entropy	0.463810	0.603196	0.649859	0.413146	0.394242
	Purity	0.524000	0.477000	0.431000	0.654000	0.634000
98.75	Entropy	0.297209	0.475559	0.536370	0.271233	0.219438
	Purity	0.763500	0.579000	0.532000	0.770500	0.821500
90.00	Entropy	0.104486	0.311849	0.338157	0.077724	–
	Purity	0.898000	0.687500	0.668000	0.930000	–

Table 7
Entropy and purity values for the covertype data set

C.R. (%)	Measure	Proposed	kd-tree		RB	Direct
			Midpoint	Median	Cos	Cos
99.99	Entropy	0.440437	0.423651	0.420830	0.443468	–
	Purity	0.540266	0.557610	0.623676	0.556083	–
99.90	Entropy	0.373562	0.368220	0.375693	0.368620	–
	Purity	0.629824	0.634780	0.654592	0.643112	–
99.00	Entropy	0.285360	0.273801	0.293886	–	–
	Purity	0.721487	0.726596	0.715214	–	–

from each other. Having done that, the data being homogeneous at each of the four nodes, the usual kd-tree based algorithms perform well.

- The proposed algorithm has performed extremely well on the shapes data set (Table 4) which has exactly the same characteristics where the proposed algorithm is expected to work better than the kd-tree based algorithms. Despite the complicated nature of the shapes of the classes involved, our algorithm, whenever possible, has managed to separate out the classes first. The horseshoe type structures could not be separated from each other using Maxdiff as the criterion, but they are gradually segregated as the number of clusters increased.
- All the methods that split parallel to the rectangular coordinate axis do not perform well on the twonorm data set (Breiman, 1998). Our algorithm too, is no exception (Table 5). It may be noted that this is an extreme example, where the data points are not separable along any of the dimensions, even after being partitioned a few times. In general, after a few initial splits, individual nodes may contain data points that satisfy the Maxdiff criterion. In the case of the twonorm data set, the Maxdiff criterion is never satisfied and all the splits are performed with respect to the mean along the axis with maximum variance. The repeated bisection method using the cosine value as the similarity manages to separate the two classes very well. This is reflected by the purity value (0.976568) when the condensation ratio is 90% as this means that only around 2.3% of the data has been wrongly classified. This error is very close to the Bayes error.
- For the forest covertype data (Table 7), splitting with respect to Maxdiff has not taken place due to the lack of separation between the various classes. All the existing methods have resulted in similar entropy and purity values.
- The quality of the representations obtained by the proposed algorithm for the multiple features data set (Table 6) is fairly superior to those of the kd-tree based algorithms while being comparable to those of the algorithms in CLUTO. The proposed algorithm is much faster compared to the algorithms in the CLUTO package.
- The direct clustering approach (Zhao and Karypis, 2002) is not appropriate, both in terms of time and the quality of clustering, when the number of clusters is large, say, greater than 20. Therefore, the results have not been shown in such cases. Also, when the data size and the number of desired clusters are both large, the repeated bisection method takes too long to provide the output. Such entries, too, have been left blank.
- We provide the time taken by the various algorithms applied to the given data sets in Table 8. As expected, the proposed algorithm requires almost the same time as the kd-tree based methods, which is far less than that needed by the other methods. The proposed algorithm, in its present state, requires memory large enough to accommodate the data set. The leaf nodes of the constructed tree store the indices of the all the data points in that node. So, the tree data structure requires space proportional to the sum of the number of data points

Table 8
Time requirements

Data set	C.R. (%)	Proposed	kd-mid	kd-median	RB (cos)	Direct (cos)
Synthetic	99.995	0.363	0.351	0.281	3.744	5.19
	99.720	1.201	1.050	1.119	14.922	>1000
Shapes	99.995	3.226	3.120	2.496	33.280	46.136
	99.700	10.869	9.489	10.139	135.186	>1000
Twonorm	99.000	16.457	14.254	15.544	207.258	>1000
	90.000	21.641	18.574	20.729	276.384	>1000
Covertype	99.990	29.198	25.900	26.548	353.970	>1000
	99.000	59.319	51.000	56.668	755.577	>1000
Multiple features	98.75	0.161	0.169	0.101	1.34	0.936
	90.00	0.498	0.450	0.438	5.834	40.440

and the number of selected prototypes. Hence, for data sets with several features, the space requirement is O(Nk).

4.1. Some remarks

The representation obtained by each of the algorithms steadily improves with the number of points in the condensed data set. For very large values of the condensation ratio, when the number of clusters obtained is smaller than the number of classes, all the algorithms tend to perform similarly. For example, in the synthetic data set with 36 classes, all the algorithms start off with similar values of purity and entropy. As the number of points in the condensed data set increases to the total number of data points, the entropy and purity values obtained by each algorithm tend to 0 and 1, respectively. When the data has well separated clusters, the fall in the entropy values corresponding to the proposed algorithm is very high initially. A similar phenomenon has been observed by Kanungo et al. (2002), where the clustering process was faster if the chosen data set had well-separated clusters.

We have observed that the proposed method outperforms other methods whenever the Maxdiff criterion is satisfied. When the given data set has no significant Maxdiff value, the results obtained are comparable to those provided by the other methods.

Categorical data have not been considered for our experiments as they require some special treatment during the process of normalization. When a data set having both continuous and binary features is normalized, the binary features would dominate in terms of the Maxdiff value and hence, the initial splits, inevitably, would be performed with respect to the binary features. This might not be suitable in certain situations.

As the proposed algorithm preserves the inherent clustering information, it can be employed for nearest-neighbor searching in the context of nearest neighbor classification, where one only needs to know the class to which the nearest neighbor belongs. Due to the reduction in confusion in the initial stages itself, the class information may be extracted more accurately employing the proposed method than by using the usual kd-tree based techniques.

5. Conclusions

We have put forward a methodology for obtaining a reasonable representation of the original data where the time and space requirements are approximately the same as that for the construction of the usual kd-trees. Since splits with the choice of the pivot as the median or the mid-point produce nodes with portions of distant clusters in them, the summary store at the node is unable to represent the data in that node. We preempt the creation of such nodes by initially splitting the nodes using the Maxdiff criterion. If the Maxdiff criterion is not satisfied at a node, the subtree formed at that node is the usual kd-tree. The algorithm does not take care of problems arising out of being confined only to the splits parallel to the coordinate axis. The approximate valley seeking techniques can, possibly, be extended to data with classes that are not well separated. Moreover, modifications for the case of categorical variables is needed.

Acknowledgements

The authors thank Dr. Pabitra Mitra for his discussions on related topics and the anonymous reviewers for their helpful comments.

References

Balakrishnan, M., Pearlman, W.A., Lu, L., 1995. Variable-rate tree-structured vector quantizers. IEEE Trans. Inform. Theory 41 (4), 917–930, July.

Breiman, L., 1998. Arcing classifiers. Ann. Statist. 26 (3), 801–849.

Broomhead, D.S., Lowe, D., 1988. Multivariable functional interpolation and adaptive networks. Complex Syst. 2, 321–355.

Catlett, J., 1991. Megainduction: Machine learning on very large databases. Ph.D. thesis, Department of Computer Science, University of Sydney, Australia.

Cuevas, A., Febrero, M., Fraiman, R., 2000. Estimating the number of clusters. Canad. J. Statist. 28 (2), 367–382.

Decaestecker, C., 1997. Finding prototypes for nearest neighbour classification by means of gradient descent and deterministic annealing. Pattern Recognition 30 (2), 281–288.

Friedman, J.H., Bentley, J.L., Finkel, R.A., 1997. An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Software 3 (3), 209–226.

Fukunaga, K., 1990. Introduction to Statistical Pattern Recognition, second ed. Academic Press.

Guha, S., Rastogi, R., Shim, K., 2001. Cure: An efficient clustering algorithm for large databases. Inform. Syst. 26 (1), 35–58.

Hart, P.E., 1968. The condensed nearest neighbor rule. IEEE Trans. Inform. Theory 14, 515–516.

Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y., 2002. An efficient *k*-means clustering algorithm: Analysis and implementation. IEEE Trans. PAMI 24 (7), 881–892.

Karypis, G., Han, E.H., Kumar, V., 1999. Chameleon: A hierarchical clustering algorithm using dynamic modeling. IEEE Comput. 32 (8), 68–75.

Moore, A.W., 1999. Very fast EM-based mixture model clustering using multiresolution kd-trees. In: Proc. Neural Information Processing Systems Conf., pp. 543–549.

Moore, A.W., Lee, M.S., 1998. Cached sufficient statistics for efficient machine learning with large datasets. J. Artif. Intell. Res. 8, 67–91.

Poosala, V., 1997. Histogram-based estimation techniques in databases. PhD Thesis, University of Wisconsin, Madison.

Provost, F.J., Kolluri, V., 1999. A survey of methods for scaling up inductive algorithms. Data Min. Knowl. Disc. 3 (2), 131–169.

Sanchez, J.S., Pla, F., Ferri, F.J., 1997. Prototype selection for the nearest neighbour rule through proximity graphs. Pattern Recogition Lett. 18 (6), 507–513.

Wang, W., Yang, J., Muntz, R., 1997. Sting: A statistical information grid approach to spatial data mining. In: Proc. 23rd Internat. Conf. on Very Large Data Bases, Athens, Greece, pp. 186–195.

Zhang, T., Ramakrishnan, R., Livny, M., 1996. Birch: An efficient data clustering method for very large databases. In: Proc. ACM SIGMOD Conf. on Management of Data, Montreal, Canada, pp. 103–114.

Zhao, Y., Karypis, G., 2001. Criterion functions for document clustering: Experiments and analysis. Technical Report 01–40, Department of Computer Science, University of Minnesota, Minneapolis, MN.

Zhao, Y., Karypis, G., 2002. Evaluation of hierarchical clustering algorithms for document datasets. In: CIKM '02, Proc. 11th Internat. Conf. on Information and Knowledge Management, pp. 515–524.