Available online at www.sciencedirect.com

ScienceDirect

Pattern Recognition Letters 28 (2007) 965–973

Pattern Recognition Letters

www.elsevier.com/locate/patrec

A method for initialising the *K*-means clustering algorithm using *kd*-trees

Stephen J. Redmond [*], Conor Heneghan

Department of Electronic Engineering, University College Dublin, Belfield, Dublin 4, Ireland

Received 28 October 2005; received in revised form 29 July 2006
Available online 14 January 2007

Communicated by A. Fred

Abstract

We present a method for initialising the *K*-means clustering algorithm. Our method hinges on the use of a *kd*-tree to perform a density estimation of the data at various locations. We then use a modification of Katsavounidis' algorithm, which incorporates this density information, to choose *K* seeds for the *K*-means algorithm. We test our algorithm on 36 synthetic datasets, and 2 datasets from the UCI Machine Learning Repository, and compare with 15 runs of Forgy's random initialisation method, Katsavounidis' algorithm, and Bradley and Fayyad's method.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Clustering; *K*-means algorithm; *Kd*-tree; Initialisation, Density estimation

1. Introduction and review

Clustering has long been the basis of many knowledge discovery tasks such as machine learning, statistics, data mining, and pattern recognition. There are two main branches of clustering; *Hierarchical* and *Partitional* (Jain et al., 1999). In this paper we concentrate on partitional clustering, and in particular a popular partitional clustering method called *K*-means clustering.

Partitional clustering involves partitioning a given set of data points into a number of distinct groups, termed *clusters*. Throughout this paper, by *data* we will be referring to *n* data points spanning an *m* dimensional space and we denote *K* as the number of clusters the data will be partitioned into. Partitional clustering attempts to partition the data points into clusters such that the similarity between the points in each cluster is maximal by some global measure. That is, if the points in each of the *K* clusters are maximally similar then changing a point's membership to another cluster will only serve to reduce the chosen global measure. Finding the optimum partition for a given global measure is known to be an NP-complete problem. However, there exist several suboptimal search strategies which find very competitive solutions within a reasonable amount of time. We describe one of the more popular of these methods next, the *K*-means algorithm.

1.1. *K*-means clustering algorithm

The algorithm of choice for many clustering tasks is the *K*-means algorithm (MacQueen, 1967). The *K*-means algorithm attempts to find the cluster centres, $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K)$, such that the sum of the squared distances (this sum of squared distances is termed the *Distortion, D*) of each data point $(\boldsymbol{x}_i)$ to its nearest cluster centre $(\boldsymbol{c}_k)$ is minimised, as shown in Eq. (1), where *d* is some distance function. Typically *d* is chosen as the Euclidean distance. A pseudo-code for the *K*-means algorithm is shown in Algorithm 1

[*] Corresponding author. Tel.: +353 87 9035170; fax: +353 1 2830921.
E-mail addresses: Stephen.Redmond@ee.ucd.ie (S.J. Redmond), Conor.Heneghan@ucd.ie (C. Heneghan).

0167-8655/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2007.01.001

$$D = \sum_{i=1}^{n} \left[\min_{k=(1\ldots K)} d(\boldsymbol{x}_i, \boldsymbol{c}_k) \right]^2 \qquad (1)$$

Unfortunately the *K*-means algorithm is a local optimisation strategy and as such is sensitive to the choice of the initial positions of the cluster centres, $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K)$ (Pena et al., 1999). These initial centre locations are often termed the *seeds* for the *K*-means algorithm. The accepted initialisation technique has been to complete repeated runs of Forgy's algorithm (c.f. Section 1.2) and choose the final clustering which results from the run that returns the smallest value of the Distortion. However, while repeated runs of the Forgy method appears to be the *de facto* approach for initialising the *K*-means algorithm, many other techniques have been proposed. We now provide a brief summary of some of these techniques in Section 1.2.

Algorithm 1. *K*-means Algorithm

```
(1) Initialise $K$ centre locations $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K)$.
(2) Assign each $\boldsymbol{x}_i$ to its nearest cluster centre $\boldsymbol{c}_k$.
(3) Update each cluster centre $\boldsymbol{c}_k$ as the mean of all $\boldsymbol{x}_i$
    that have been assigned as closest to it.
(4) Calculate $D = \sum_{i=1}^{n} [\min_{k=(1\ldots K)} d(\boldsymbol{x}_i, \boldsymbol{c}_k)]^2$.
(5) If the value of $D$ has converged, then return
    $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K)$; else go to Step 2.
```

1.2. A review of some initialisation methods

One of the earliest references to initialising the *K*-means algorithm was by Forgy in 1965 (see Anderberg, 1973). We shall term this approach the Forgy Approach (FA). Forgy simply suggested that *K* instances should be chosen from the database at random and used as the seeds. This approach takes advantage of the fact that if we choose points randomly we are more likely to choose a point near a cluster centre by virtue of the fact that this is where the highest density of points is located. However, there is no guarantee that we will not choose two seeds near the centre of the same cluster, or that we will not choose some poorly situated outlying point. In fact, repeated runs of this method is presently the standard method of initialising the *K*-means algorithm. We note that the primary negative aspect of repeated runs is the increased time taken to obtain a solution.

In 1967, McQueen introduced what was is akin to an online learning strategy to determine a set of cluster seeds (MacQueen, 1967). The first *K* points in the database are chosen as the preliminary seeds. Then, the next data point in the database is assigned to the cluster represented by the nearest seed. The centroid of that cluster is updated to be equal to the mean of all points assigned to it. The next data point is introduced, and the mean of its nearest cluster is update accordingly. This process is repeated until all data points are assigned to a cluster. The important point to note is that once a data point is assigned to a cluster its membership does not change. Hence, once all data points have been assigned, a point may be assigned to a cluster whose centroid is not the nearest. The *K* resulting centroids are used to seed the *K*-means algorithm. In a very large database this method can be somewhat burdensome, as a mean vector must be calculated every time a new instance is added.

Tou and Gonzales (1974) suggest the Simple Cluster-Seeking (SCS) method. Initialise the first seed with the first instance in the database. Calculate the distance between this seed and the next point in the database, if it is greater than some threshold then select it as the second seed, otherwise move to the next instance in the database and repeat. Once the second seed is chosen move to the next instance in the database and calculate the distance between it and the two seeds already chosen, if both these distances are greater than the threshold then select as the third seed. We continue until *K* seeds are chosen. We note that this method is dependent on the order of the points in the database, and, more critically, we must decide on a user-defined threshold value.

Linde et al. (1980) proposed a Binary Splitting (BS) method which was intended for use in the design of Vector Quantiser codebooks. It calls upon a hierarchical use of the *K*-means algorithm. The *K*-means algorithm is first run for $K = 1$. The cluster centre found, $\boldsymbol{c}$, is split into two clusters, $\boldsymbol{c} + \boldsymbol{\epsilon}$ and $\boldsymbol{c} - \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is some small random vector. The *K*-means algorithm is run again with these two points as seeds. When convergence is reached, the two cluster centres are again split to make four seeds and the *K*-means algorithm is run again. The cycle of split and converge is repeated until a fixed number of clusters is reached, or until each cluster contains only one point. This method clearly carries increased computational complexity, since after each split the *K*-means algorithm must be run. In addition, the quality of the clustering after each split depends upon the choice of $\boldsymbol{\epsilon}$, since this determines the direction in which each cluster will be split.

Kaufman and Rousseeuw (1990) suggest that we select the first seed as the most centrally located instance. Next we examine which of the points in the database, which when chosen as the next seed, will produce the greatest reduction in the Distortion, *D* (c.f. Eq. (1)). Once the second is chosen we choose the third seed in the same fashion and continue until *K* seeds are chosen. Again the first obvious drawback of this algorithm is the considerable amount of computation involved in choosing each seed. If this algorithm is to be considered useful for large databases a subsample of the instances must be used instead when finding the seeds (He et al., 2004).

Babu and Murty (1993) published a method of near-optimal seed selection using genetic programming. The population consists of various seed selections. The fitness of each seed selection is assessed by running the *K*-means algorithm until convergence and then calculating the Distortion value. The fittest solutions will then reproduce to create a second generation of solutions. We repeat this process until a predetermined number of generations have been created. While this method can find the optimum solution in many cases it is seriously hampered by the need for repeated runs of the K-means algorithm; K-means must be run for each solution of each generation, which in a large database quickly becomes infeasible. In addition, a problem with genetic algorithms is that the results vary significantly with the choice of population size, and crossover and mutation probabilities (Jain et al., 1999).

Huang and Harris (1993) proposed the Direct Search Binary Splitting (DSBS) method. This method is similar to the Binary Splitting algorithm above except that the splitting step is enhanced through the use of Principle Component Analysis (PCA). This technique employs PCA to choose the ϵ vector of Linde et al. deterministically in an attempt to improve the clustering quality.

Katsavounidis et al. (1994) proposed what has been termed by some as the KKZ algorithm. This algorithm starts by choosing a point x, preferably one on the 'edge' of the data, as the first seed. The point which is furthest from x is chosen as the second seed. The distance of all points to the nearest of these two seeds is calculated. The point which is the furthest from its nearest seed is chosen as the third seed. We repeat choosing the furthest point from its nearest seed until K seeds are chosen. This method has one obvious pitfall. Any noise in the data, in the form of outlying data points, will pose difficulties for this procedure. Any such outlying points will be preferred by the algorithm but will not necessarily be near any cluster centre.

Daoud and Roberts (1996) proposed two methods, the first of which we discuss now. The method suggests that we divide the whole input domain into M disjoint subspaces, $S_j, j = 1, \ldots, M$. The number of points, N_j, in each subspace are counted. $K_j = K(N_j/n)$ seeds are randomly placed in the jth subspace.

As an improvement to the above method, in the same paper they present a method which iteratively subdivides the input domain into two disjoint volumes. In each subspace it is assumed that the points are randomly distributed and that the seeds will be placed on a regular grid. Using these assumptions they decide how many seeds should be placed on a regular grid in each subspace based on the density of points. This turns out to be the solution to a polynomial equation, which is a quadratic in the two dimensional case. Each subspace is again split and the number of seeds assigned is divided between the two smaller subspaces based on the same criterion. However, a proof of the validity of the method is not included, and they only produce results for a two dimensional database.

Thiesson et al. (1997) suggest taking the mean of the entire dataset and randomly perturbing it K times to produce the K seeds. This essentially means creating a 'cloud' of K points around the mean of the data.

Bradley and Fayyad (1998) present a technique for initialising the K-means algorithm. They begin by randomly breaking the data into 10, or so, subsets. They then perform a K-means clustering on each of the 10 subsets, all starting at the same set of initial seeds, which are chosen using Forgy's method. The result of the 10 runs is $10K$ centre points. These $10K$ points are then themselves input to the K-means algorithm and the algorithm run 10 times, each of the 10 runs initialised using the K final centroid locations from one of the 10 subset runs. The resulting K centre locations from this run are used to initialise the K-means algorithm for the entire dataset.

Likas et al. (2003) present a global K-means algorithm which aims to gradually increase the number of seeds until K are found. They compare their method with multiple runs of the K-means algorithm. They employ a variation of the *kd*-tree to initialise each run of this comparison method. They simply use the *kd*-tree to create K buckets and use the centroids of each bucket as seeds. A description of *kd*-trees is provided in Section 2.1. The variation on the *kd*-tree they use is to split each bucket along the direction of maximum variance, as defined by Principle Component Analysis.

Most recently, Khan and Ahmad (2004) describe their Cluster Centre Initialisation Method (CCIA). Their method hinges on the use of *Density-based Multi Scale Data Condensation* (DBMSDC) which was introduced by (Mitra et al., 2002). DBMSDC involves estimating the density of the data at a point, and then sorting the points according to their density. From the top of the sorted list we choose a point and prune all points within a radius inversely proportional to the density of that point. We then move on to the next point which has not been pruned from the list and repeat. This is repeated until a desired number of points remain. The authors choose their seeds by examine each of the m attributes individually to extract a list of $K' > K$ possible seed locations. Next the DBMSDC algorithm is invoked and points which are close together are merged until there are only K points remaining.

2. Methods

2.1. kd-Trees

The *kd*-tree is a top-down hierarchical scheme for partitioning data. Consider a set of n points, $(\boldsymbol{x}_1 \ldots \boldsymbol{x}_n)$ occupying an m dimensional space. Each point $\boldsymbol{x}_i$ has associated with it m co-ordinates $(x_{i1}, x_{i2}, \ldots, x_{im})$. There exists a bounding box, or bucket, which contains all data points and whose extrema are defined by the maximum and minimum co-ordinate values of the data points in each dimension. The data is then partitioned into two sub-buckets by splitting the data along the longest dimension of the parent bucket, which we denote $m_{\max}$. The value at which the split is made can be determined in various ways. One method is to split the data along the median value of the co-ordinates in that dimension, $\text{median}(x_{1m_{\max}}, x_{2m_{\max}}, \ldots, x_{nm_{\max}})$, so that the number of points in each bucket remains roughly the same. Or, we may simply split at the mean in the direction of $m_{\max}$, which requires less computation than the median. In this paper we split each dimension at the median.

This partitioning processes may then be recursively repeated on each sub-bucket until a leaf bucket (denoted L) is created, at which point no further partitioning will be performed on that bucket. L is a bucket which fulfills a certain requirement, such as, it only contains 1 data point or contains less than 10 data points. Eventually, after enough splitting of buckets, all buckets will be leaf buckets and the partitioning process will terminate. Fig. 1 shows three steps in the process of constructing a *kd*-tree for a sample dataset.

An interesting property of the *kd*-tree is that each bucket will contain roughly the same number of points. However, if the data in a bucket is more densely packed than some other bucket we would generally expect the volume of that densely packed bucket to be smaller. This will be the basis for our initialisation algorithm. We notice that this is an advantage over a simple histogram type approach to density estimation. If we were to simply divide the space into cubes (or hypercubes) by dividing each dimension into, say, 10 divisions, we would have 10^m cubes and we could estimate the density at the location of each cube by simply counting the number of points it contains. This is a valid method of density estimation at low dimensions, however at higher dimensions 10^m cubes becomes an unreasonable number. With the *kd*-tree approach at most n buckets will need to be dealt with as the space has been more intelligently divided.

2.2. kd-Density initialisation

We start by creating a *kd*-tree, stipulating that a leaf bucket is that which, arbitrarily, contains 20 points or less. We count the number of leaf buckets created as q. We calculate the volume, V_j, of each leaf bucket, L_j, and count the number of points it bounds, N_j. We then calculate the density of each leaf bucket, L_j, to be ρ_j, for $j = 1,\ldots,q$:

$$\rho_j = N_j/V_j \tag{2}$$

In the instantiation of the *kd*-tree used here, where the bounding bucket is defined by the extrema of the points it bounds, some care must be taken to ensure the volume of the bucket is not zero. This may happen if all points within a bucket have one, or more, co-ordinate values which are identical. This is simply remedied by replacing the bucket width, along the dimensions in which the width is zero, by the geometric mean of the dimensions along which the width is greater than zero.

We must associate each leaf bucket, L_j, or more exactly the density of each leaf bucket, ρ_j, with a point in the space. We choose this point to be the mean of the data points contained within the leaf bucket. We denote this point $\boldsymbol{m}_j$. So, we now have a list of q points $(\boldsymbol{m}_1,\ldots,\boldsymbol{m}_q)$ in the space and the corresponding estimates of the density of the data at each point $(\rho_1,\ldots,\rho_q)$.

To choose the initial seeds for the K-means algorithm, we wish to use this density information. We aim to choose K leaf bucket locations, $\boldsymbol{m}_j$, from q possibilities, that are separated by a reasonable distance and have a large density. We choose the first seed, $\boldsymbol{c}_1$, to be the leaf bucket with highest density: $\boldsymbol{c}_1 = \boldsymbol{m}_{\arg\max_j(\rho_j)}$. To choose the second seed we calculate for every remaining leaf bucket centroid, $\boldsymbol{m}_j$, the value g_j. g_j is the distance of $\boldsymbol{m}_j$ from the first seed location, $\boldsymbol{c}_1$ (using the Euclidean distance, $d(\cdot,\cdot)$), multiplied by the density of the leaf bucket, ρ_j, as shown in Eq. (3). The second seed is then chosen as the point, $\boldsymbol{m}_j$, with the maximum value of g

$$g_j = d(\boldsymbol{c}_1, \boldsymbol{m}_j).\rho_j \tag{3}$$

The idea is that the further away a leaf bucket is from an existing seed, and the larger its density, the more likely a candidate it is to be a seed location. This is similar to the KKZ described in Section 1.2, except we are weighting the distances by the density of each bucket. Similarly, the third seed is chosen by computing the distance of each leaf bucket centroid, $\boldsymbol{m}_j$, from its *nearest* seed location, $\boldsymbol{c}_k$, multiplied by the density of the leaf bucket itself, ρ_j, as shown in Eq. (4):

$$g_j = \left\{\min_{k=1,2}\left[d(\boldsymbol{c}_k, \boldsymbol{m}_j)\right]\right\}.\rho_j \tag{4}$$

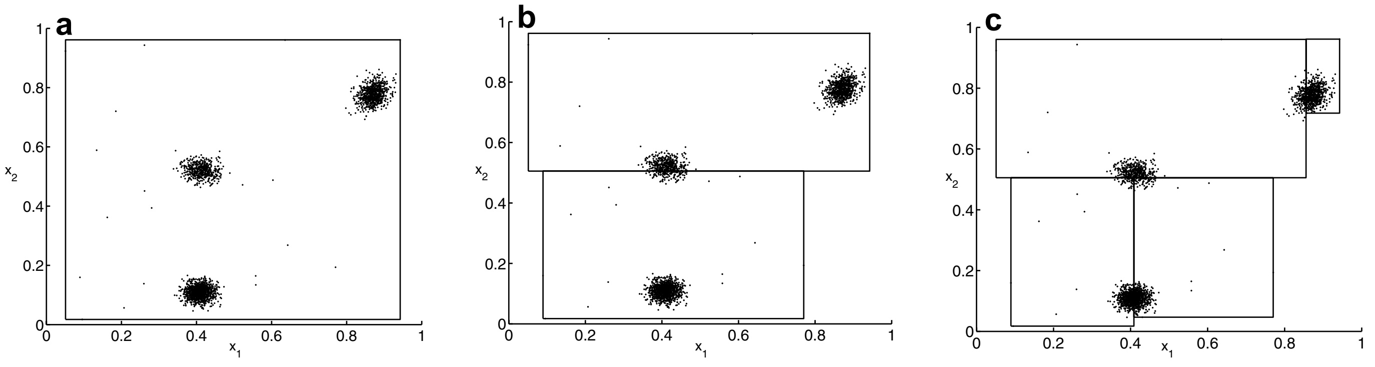

Fig. 1. Three steps in the creation of a *kd*-tree for a representative 2-D distribution: (a) the bounding bucket for all data points, (b) two buckets after splitting the bucket from (a), (c) after one further split.

In general, if t seed locations have been chosen, the next seed location, $\boldsymbol{c}_{t+1}$, is chosen as the $\boldsymbol{m}_j$ corresponding with the leaf bucket L_j, which gives the maximum value of g_j, for $j = 1,\ldots,q$, in Eq. (5). When K seeds have been chosen the algorithm terminates.

$$g_j = \left\{\min_{k=1\ldots t}[d(\boldsymbol{c}_k,\boldsymbol{m}_j)]\right\}.\rho_j \qquad (5)$$

There are two problems associated with this algorithm in its present form. The first is that computing the densities using the *kd*-tree causes the density estimates of the lower density leaf buckets to be underestimated. As a result the calculation of g in Eq. (5) is dominated by the higher density leaf buckets and the outcome is the placement of several seed locations near denser clusters, while less dense clusters are ignored. This impediment is easily overcome. We simply use the rank value of the density estimates instead of the actual estimates. For example, if there are 1024 leaf buckets each having a density estimate associated with it, we sort the density estimates and assign the leaf bucket a rank density. So, in this instance, the highest density leaf bucket will have a rank density of 1024, and the lowest density leaf bucket a rank density of 1. We will denote this new rank density estimate for each leaf bucket $\hat{\rho}$, and it will replace ρ in Eq. (5).

The second problem encountered is associated with outlying data points. Outliers will generally be contained in leaf buckets with a low density. However, if the outlier is at a significant distance from the rest of the data the large distance will give it a high value of g and the bucket will be chosen erroneously as an initial centre. To combat this issue we run the algorithm twice, creating two possible sets of seed locations. Firstly using all leaf buckets, and then a second time using, say, 80% of the densest leaf buckets. By using 80% of most dense buckets we are essentially performing an outlier removal task before running the algorithm. The algorithm is summarised in Algorithm 2.

Algorithm 2. Initialisation algorithm

```
(1) Create a kd-tree for the given data $x_i$, $i = 1,\ldots,n$.
(2) For $j = 1,\ldots,q$ calculate the rank density, $\hat{\rho}_j$, of each
    leaf bucket $L_j$, where $q$ is the number of leaf buckets in
    the kd-tree. Calculate the mean, $\boldsymbol{m}_j$, of the data points
    bounded by each leaf bucket.
(3) Choose $\boldsymbol{c}_1 = \boldsymbol{m}_z$, where $z = \arg\max_j(\hat{\rho}_j)$.
(4) For $t = 2,\ldots,K$,
      – For $j = 1,\ldots,q$ evaluate
          $g_j = \{\min_{k=1\ldots t}[d(\boldsymbol{c}_k,\boldsymbol{m}_j)]\}\cdot\hat{\rho}_j$.
      – $\boldsymbol{c}_t = \boldsymbol{m}_z$, where $z = \arg\max_j(g_j)$.
(5) Discard the 20% of leaf buckets that have the lowest
    density. Go to step 3 and calculate a second possible list
    of $K$ initial centres, $(\hat{\boldsymbol{c}}_1,\ldots,\hat{\boldsymbol{c}}_K)$.
(6) Return $(\boldsymbol{c}_1,\ldots,\boldsymbol{c}_K)$ and $(\hat{\boldsymbol{c}}_1,\ldots,\hat{\boldsymbol{c}}_K)$.
```

We note here that *kd*-trees are known to scale poorly with the dimensionality of the dataset. In Section 3 we describe the datasets used and we point out that the largest dimensionality used is $m = 20$. The proposed method has not been tested outside these bounds.

3. Experimental design

3.1. Synthetic data

In order to test our algorithm we used synthetic multivariate Gaussian data. Each dataset consists of K Gaussian clusters, hence we must choose the means and covariance matrices of each cluster, and how many points are in each cluster. To choose the mean, M_k, of the kth cluster, each co-ordinate of the mean is chosen randomly over the interval $[0,1]$ with a uniform probability. Hence the centres of all clusters are contained within a unit hypercube. The covariance matrix of the kth cluster is first chosen to be a diagonal matrix; where the value of each diagonal entry, t, is chosen to be σ_t^2 for $t = (1,\ldots,m)$. σ_t is chosen from a uniform random distribution on the interval $[0.2(s\sqrt{m}), s\sqrt{m}]$. s is an arbitrarily defined scaling factor which we can adjust to change the ratio of the (size of the cluster)/(the distance between clusters). Scaling σ_t, and hence the width of the clusters along each axis, by $\sqrt{m}$ is done so that the 'radius' of the cluster relative to the distance between clusters is kept constant as the dimension, m, increases. This covariance matrix for the kth cluster, Σ_k, essentially describes a multivariate Gaussian distribution which has been stretched by random amounts parallel to the axis of each dimension. We now wish to perform a random rotation of the distribution. This is accomplished by creating a new covariance matrix $\hat{\Sigma}_k = Q_k\Sigma_kQ_k^{-1}$, where Q_k is an $m\times m$ matrix whose columns contain vectors which are orthogonal and of unit length. We construct Q_k by choosing a random $m\times m$ matrix, say A_k, and then performing a QR decomposition to give us two matrices Q_k and R_k such that $A_k = Q_kR_k$; where R_k is an upper triangular matrix and Q_k contains unit orthogonal vectors for its columns, as required. Finally, we choose the number of points in the kth cluster, N_k, from a uniform random distribution over the interval $[50,1000]$. Note that this enables some clusters to contain up to 20 times more points than other clusters. This makes the task more difficult as it may be easy to miss the lightly weighted clusters. We now generate N_k points which have a mean M_k and a covariance matrix $\hat{\Sigma}_k$. We repeat this process for $k = (1,\ldots,K)$, hence creating K clusters. In addition, to make the problem more realistic, we add $(0.01)(n)$ random points chosen with a uniform random probability over the interval $[0,1]$, where $n = \sum_{k=1}^{K}N_k$. We construct 36 such datasets by varying K over the values $\{10,15,20\}$, the dimension, m, over the values $\{2,8,15,20\}$, and the scaling factor, s, over the values $\{0.03,0.05,0.07\}$. We also save a class variable which indicates which cluster a point belongs to, where noise points are considered a class unto them-

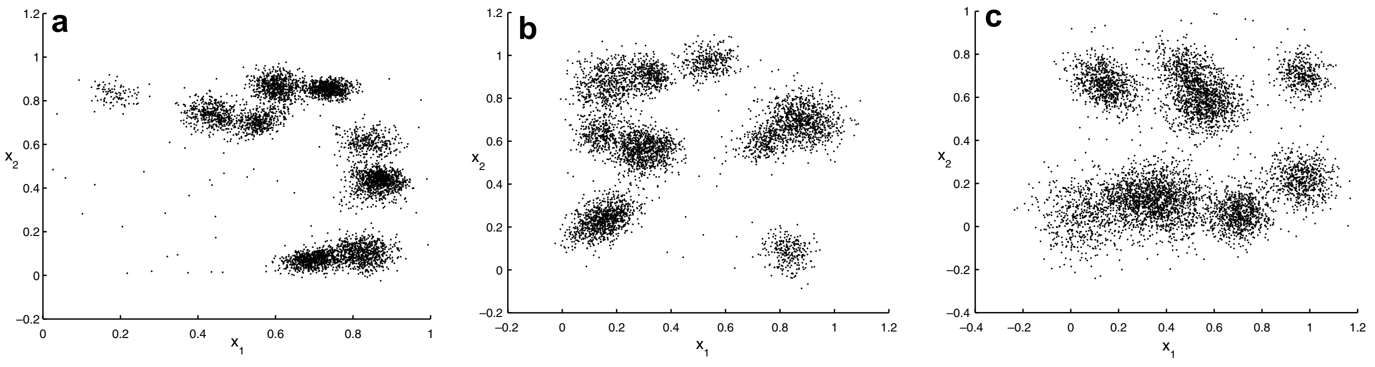

Fig. 2. Example plots of 10 clusters in two dimensions for three different scaling values, s: (a) $s = 0.03$, (b) $s = 0.05$, (c) $s = 0.07$.

selves. Hence, for K clusters there will be $K + 1$ classes including noise. We will use this class information later. For visualisation purposes, Fig. 2 shows three examples of 2 dimensional data plots containing 10 clusters each for various scale values.

3.2. Real-world datasets

We wish to test if the performance of the algorithm when applied to the synthetic datasets translates to more realistic clustering problems. To investigate the utility of our algorithm in this respect we use two real-world databases obtained from the UCI Machine Learning Repository (Newman et al., 1998). The first is the *Pen-Based Recognition of Handwritten Digits Database.* It consists of 10,992 instances, of 16 features, describing the 10 written digits $0,\ldots,9$. Hence, there are 10 classes. All ordinates take integer values between 0 and 100, inclusive.

The second database used was the *Image Segmentation Database*. It consists of 2310 instances, of 19 features, which describe aspects from 7 classes of image (brickface, sky, foliage, cement, window, path, grass).

3.3. Comparison with other initialisation methods

As a measure of performance we compare our algorithm for initialisation with some of the methods described in Section 1.2. In particular we choose to use repeated runs of Forgy's approach, Bradley and Fayyad's method, and Katsavounidis' method. We make 15 runs of the Forgy method. For the Bradley and Fayyad method we use the Forgy method to choose the seeds to be refined, and we divide the dataset into 10 subsets. In all runs of the K-means algorithm, should an empty cluster be created, we will delete it and create a new cluster containing the one data point which is furthest from its nearest centroid.

3.4. Metrics for comparison

3.4.1. Normalised information gain

An alternative measure of the quality of a clustering is that of information gain (Bradley and Fayyad, 1998). Information gain uses the concept of entropy as a measure of information. If we have n points, we let c_l denote the number of points in class l, for $l = 1,\ldots,L$. Hence, $\sum_{l=1}^{L} c_l = n$. The *Total Entropy*, (EN_{Total}), or average information per point in the dataset, measured in bits, is

$$EN_{\text{Total}} = -\sum_{l=1}^{L}\left(\frac{c_l}{n}\right)\log_2\left(\frac{c_l}{n}\right)\ \text{bits}$$

If we cluster the dataset using K clusters, such that the number of points in the kth cluster is n_k, and the number of points belonging each of the L classes is c_l^k, then the entropy of the kth cluster (EN_k) is

$$EN_k = -\sum_{l=1}^{L}\left(\frac{c_l^k}{n_k}\right)\log_2\left(\frac{c_l^k}{n_k}\right)\ \text{bits}$$

Note that if the kth cluster contains only one class, then $EN_k = 0$. We define the *Weighted Entropy* (wEN), the average information per point in each cluster, as

$$wEN = \sum_{k=1}^{K}\left(\frac{n_k}{n}\right)EN_k\ \text{bits}$$

Again, we note that if all clusters are homogeneous, then $wEN = 0$. We define the *Normalised Information Gain* (NIG) (which will take a value of $NIG = 0$ if no information is retrieved by the clustering and $NIG = 1$ if all information is retrieved) as

$$NIG = \frac{EN_{\text{Total}} - wEN}{EN_{\text{Total}}}$$

3.4.2. Summary of metrics

Below we summarise the metrics used to compare the performance of the various systems:

- the smallest value of Distortion (D_{kd}) returned by the two runs of our algorithm (before and after pruning low density buckets),
- the value of Distortion (D_{bf}) returned by the Bradley and Fayyad approach,
- the value of Distortion (D_{kkz}) returned by Katsavounidis' KKZ approach,

- the smallest value of Distortion ($Dmin_{fa}$) returned by the 15 runs of the K-means algorithm with random initialisation by the Forgy approach,
- the mean (μ_{fa}) and standard deviation (σ_{fa}) of the Distortion values returned by each of the 15 runs of the Forgy method,
- the number of times a run of the Forgy approach returns a lower ($N_{<}$), equal [within 1%] ($N_{=}$), or greater value ($N_{>}$) of Distortion than our *kd*-density algorithm. These numbers will obviously vary between 1 and 15,
- Normalised information gain (NIG) for our algorithm, (NIG_{kd}), the *maximum* NIG returned by the Forgy Approach, (NIG_{fa}), NIG for the Bradley and Fayyad approach, (NIG_{bf}), and the NIG for Katsavounidis' KKZ approach, (NIG_{kkz}).

4. Results

4.1. Synthetic data results

In Tables 1–3 we present the results of the comparison between our initialisation scheme, and the three other methods listed in Section 3.3, when applied to the synthetic data outlined in Section 3.1. A description of the metrics used in the comparison is contained in Section 3.4.2.

4.2. Real-world data results

Tables 4 and 5 show the results of the various initialisation schemes applied to the Pen-Based Recognition of Handwritten Digits dataset, and the Image Segmentation dataset from the UCI Machine Learning Repository.

5. Discussion and conclusions

We have presented a technique for initialising the K-means algorithm. We incorporate *kd*-trees to obtain density estimates of the data in regions of interest. We then sequentially select K seeds, using distance and density information to aid each selection.

Examining the results for the synthetic data in Tables 1–3 we first note that Forgy's method only produces a lower value of Distortion for 5 of the 36 datasets, and upon qualitative examination of these 5 cases the improvement does not appear to have been significant. Pooling all runs initialised by Forgy's method we see that only 6 out of 540 (15×36) runs showed an improvement over our *kd*-density

Table 1
Results for synthetic datasets, created as described in Section 3.1, with $K = 10$, dimensions $m = (2, 8, 15, 20)$ and scaling factors $s = (0.03, 0.05, 0.07)$

K	m	s	D_{kd}	D_{bf}	D_{kkz}	$Dmin_{fa}$	μ_{fa}	σ_{fa}	$N_{<}$	$N_{=}$	$N_{>}$	NIG_{kd}	NIG_{bf}	NIG_{kkz}	NIG_{fa}
10	2	0.03	14.76	13.64	17.29	13.64	30.46	13.38	1	0	14	0.91	0.93	0.86	0.93
10	2	0.05	26.25	27.17	26.35	26.35	35.53	10.50	0	1	14	0.84	0.84	0.86	0.86
10	2	0.07	47.10	47.59	47.59	46.69	57.89	8.61	0	1	14	0.78	0.78	0.78	0.78
10	8	0.03	181	234	233	181	337	134	0	1	14	0.97	0.92	0.93	0.97
10	8	0.05	547	547	641	547	703	100	0	1	14	0.97	0.97	0.91	0.97
10	8	0.07	1225	1225	1225	1225	1439	191	0	2	13	0.95	0.95	0.95	0.95
10	15	0.03	647	888	828	647	1136	365	0	1	14	0.98	0.90	0.94	0.98
10	15	0.05	2113	2335	2113	2113	2525	410	0	3	12	0.98	0.94	0.98	0.98
10	15	0.07	2963	3047	3121	3026	3191	142	0	0	15	0.97	0.94	0.92	0.95
10	20	0.03	1473	1473	2790	1473	2173	472	0	3	12	0.98	0.98	0.86	0.98
10	20	0.05	3532	3532	3532	3532	4034	266	0	1	14	0.98	0.98	0.98	0.98
10	20	0.07	6454	6454	7158	6454	6793	194	0	1	14	0.97	0.97	0.89	0.97

The performance metrics displayed are described in Section 3.4.

Table 2
Results for synthetic datasets, created as described in Section 3.1, with $K = 15$, dimensions $m = (2, 8, 15, 20)$ and scaling factors $s = (0.03, 0.05, 0.07)$

K	m	s	D_{kd}	D_{bf}	D_{kkz}	$Dmin_{fa}$	μ_{fa}	σ_{fa}	$N_{<}$	$N_{=}$	$N_{>}$	NIG_{kd}	NIG_{bf}	NIG_{kkz}	NIG_{fa}
15	2	0.03	16.02	15.60	16.74	15.52	20.24	4.70	1	3	11	0.84	0.83	0.82	0.84
15	2	0.05	43.05	44.41	42.92	42.98	48.40	5.39	0	1	14	0.86	0.86	0.86	0.86
15	2	0.07	62.28	61.96	64.24	62.04	67.88	5.14	0	2	13	0.76	0.76	0.75	0.76
15	8	0.03	285	325	356	295	487	124	0	0	15	0.98	0.94	0.96	0.97
15	8	0.05	815	815	831	841	944	104	0	0	15	0.96	0.96	0.96	0.95
15	8	0.07	1802	1890	1876	1802	1958	105	0	1	14	0.93	0.88	0.89	0.93
15	15	0.03	901	976	1146	1071	1431	241	0	0	15	0.98	0.97	0.96	0.96
15	15	0.05	2561	2599	2588	2588	3029	355	0	0	15	0.98	0.97	0.97	0.97
15	15	0.07	3988	4044	4054	4023	4173	133	0	1	14	0.95	0.93	0.93	0.94
15	20	0.03	1583	1854	1632	1649	2305	549	0	0	15	0.98	0.96	0.98	0.97
15	20	0.05	4642	4691	4689	4701	5090	310	0	0	15	0.98	0.98	0.97	0.97
15	20	0.07	7689	7771	7689	7689	8021	232	0	1	14	0.96	0.95	0.96	0.96

The performance metrics displayed are described in Section 3.4.

Table 3
Results for synthetic datasets, created as described in Section 3.1, with $K = 20$, dimensions $m = (2, 8, 15, 20)$ and scaling factors $s = (0.03, 0.05, 0.07)$

K	m	s	D_{kd}	D_{bf}	D_{kkz}	$Dmin_{fa}$	μ_{fa}	σ_{fa}	$N_<$	$N_=$	$N_>$	NIG_{kd}	NIG_{bf}	NIG_{kkz}	NIG_{fa}
20	2	0.03	23.05	22.63	26.02	22.25	24.98	1.62	2	0	13	0.88	0.88	0.85	0.89
20	2	0.05	41.77	42.99	42.62	41.28	46.69	6.67	1	2	12	0.76	0.75	0.75	0.76
20	2	0.07	52.36	53.07	51.70	51.64	53.74	2.50	1	5	9	0.68	0.68	0.68	0.67
20	8	0.03	414	554	699	488	692	170	0	0	15	0.98	0.95	0.91	0.96
20	8	0.05	1082	1138	1106	1085	1217	91	0	1	14	0.96	0.94	0.95	0.96
20	8	0.07	1669	1698	1665	1667	1711	30	0	4	11	0.87	0.86	0.88	0.87
20	15	0.03	1483	2007	1887	1651	2605	464	0	0	15	0.98	0.95	0.96	0.97
20	15	0.07	7382	7383	7721	7694	7860	120	0	0	15	0.94	0.94	0.91	0.90
20	20	0.03	2704	3203	2949	3555	4407	444	0	0	15	0.98	0.98	0.97	0.94
20	20	0.05	7232	7444	7231	7444	8173	548	0	0	15	0.98	0.97	0.98	0.97
20	20	0.07	1220	1229	1230	1229	1256	187	0	1	14	0.96	0.94	0.95	0.94

The performance metrics displayed are described in Section 3.4.

Table 4
Results for the pen-based recognition of handwritten digits datasets discussed in Section 3.2

K	m	D_{kd}	D_{bf}	D_{kkz}	$Dmin_{fa}$	μ_{fa}	σ_{fa}	$N_<$	$N_=$	$N_>$	NIG_{kd}	NIG_{bf}	NIG_{kkz}	NIG_{fa}
10	16	4.93×10^7	4.93x10^7	5.15×10^7	4.93×10^7	5.08×10^7	9.07×10^5	0	1	14	0.67	0.67	0.64	0.67

The performance metrics displayed are described in Section 3.4.

Table 5
Results for the image segmentation dataset discussed in Section 3.2

K	m	D_{kd}	D_{bf}	D_{kkz}	$Dmin_{fa}$	M_{fa}	σ_{fa}	$N_<$	$N_=$	$N_>$	NIG_{kd}	NIG_{bf}	NIG_{kkz}	NIG_{fa}
7	19	1.40×10^7	1.46×10^7	2.40×10^7	1.39×10^7	1.51×107	2.40×10^6	0	6	9	0.48	0.37	0.21	0.54

The performance metrics displayed are described in Section 3.4.

initialisation technique. In fact 495 of the Forgy runs returned a worse value of minimum Distortion. Of the 36 datasets the Bradley and Fayyad method returned a lower minimum Distortion than our *kd*-density method four times, whereas it returned a larger value (greater than 1%) for 20 datasets. The KKZ approach did not return any lower values of minimum Distortion and returned a larger value for 24 of the 36 datasets. For these synthetic datasets the NIG measure was not very instructive with all methods performing within similar ranges.

All techniques, except the KKZ method, performed equally well on the Pen-Based Recognition of Handwritten Digits dataset. The KKZ was the only method not to find the minimum Distortion value of 4.93×10^7. Our *kd*-density method and the Bradley–Fayyad method found this clustering in one run. However, only 1 of the 15 Forgy initialisations resulted in this clustering.

The Image Segmentation dataset was the more interesting of the two real-world datasets. The Bradley–Fayyad method returned a minimum Distortion value 4% larger, and the KKZ method a minimum Distortion 71% larger, than our *kd*-density method. Accordingly, the NIG measures reflect the larger Distortion values ($NIG_{kd} = 0.48$, $NIG_{bf} = 0.37$, and $NIG_{KKZ} = 0.21$). However, the 15 runs of the Forgy method found 6 solutions with a Distortion value equal (within 1%) to the *kd*-density value. Of those 6 solutions, one happened to have a Normalised Information Gain of 0.54. This is illustrative of the fact that while our method attempts to minimise Distortion, as does the *K*-means algorithm, it is ignorant of the class labels used in the calculation of the NIG measure. The advantage of repeated random initialisation is that it allows the search of solution spaces which may return a better clustering as defined by the NIG measure but not necessarily minimise the Distortion value for the dataset.

We draw the readers attention to the total runtime for the experiments. We use the 36 synthetic datasets to illustrate our point. All algorithms were implemented in Matlab 6.5 and executed on a Pentium M processor. We made no effort to optimise algorithm speed. Repeated runs of Forgy's method took 3276 s (54 min, 39 s), while our *kd*-method took 188 s (3 min, 8 s), the Bradley–Fayyad method took 169 s (2 min, 49 s) and the KKZ method took 138 s (2 min, 18 s). We may wish to take advantage of the stochastic element of repeated runs of Forgy's method to try and maximise information gain rather than minimise Distortion, however, as a consequence the runtime is considerably lengthened.

We do not provide an exposition of the computational complexity of our initialisation algorithm since we do not claim that it is computationally more efficient than the Bradley and Fayyad's, or Katsavounidis' initialisation methods. In fact, our method performs slower than these two methods (at 188 s). However, for a very modest increase in runtime our algorithm provides improved performance.

Again we draw the readers attention to the fact that *kd*-trees are known to scale poorly to high dimensions. The reader should note that the largest dimensionality of any dataset used in this paper is $m = 20$.

We conclude that our method of seed selection for the *K*-means algorithm provides a fast and reliable method of initialisation. We have compared our method to three other initialisation techniques. While the method occasionally failed to provide the lowest value of Distortion, in the cases were it did not, it did achieve a competitive result.

Acknowledgements

The authors would like to thank the Irish Research Council for Science, Engineering and Technology (IRCSET) for supporting this work.

References

Anderberg, M.R., 1973. Cluster Analysis for Applications. Academic Press, New York.

Babu, G.P., Murty, M.N., 1993. A near-optimal initial seed value selection in *k*-means algorithm using a genetic algorithm. Pattern Recognition Lett. 14 (10), 763–769.

Bradley, P.S., Fayyad, U.M., 1998. Refining initial points for *k*-means clustering. In: Proc. 15th Internat. Conf. on Machine Learning. Morgan Kaufmann, San Francisco, CA, pp. 91–99. Available from: <citeseer.ist.psu.edu/bradley98refining.html>.

Daoud, M.B.A., Roberts, S.A., 1996. New methods for the initialisation of clusters. Pattern Recognition Lett. 17 (5), 451–455.

He, J., Lan, M., Tan, C.-L., Sung, S.-Y., Low, H.-B., July 2004. Initialization of cluster refinement algorithms: A review and comparative study. In: Proc. of Internat. Joint Conf. on Neural Networks (IJCNN), Budapest, Hungary, vol. 1, pp. 297–302.

Huang, C., Harris, R., 1993. A comparison of several codebook generation approaches. IEEE Trans. Image Process. 2 (1), 108–112.

Jain, A.K., Murty, M.N., Flynn, P.J., 1999. Data clustering: A review. ACM Comput. Surveys 31 (3), 264–323. Available from: <citeseer.ist.psu.edu/jain99data.html>.

Katsavounidis, I., Kuo, C.C.J., Zhen, Z., 1994. A new initialization technique for generalized lloyd iteration. Signal Process. Lett. IEEE 1 (10), 144–146.

Kaufman, L., Rousseeuw, P.J., 1990. Finding Groups in Data. An Introduction to Cluster Analysis. Wiley, Canada.

Khan, S.S., Ahmad, A., 2004. Cluster center initialization algorithm for *k*-means clustering. Pattern Recognition Lett. 25 (11), 1293–1302.

Likas, A., Vlassis, N., Verbeek, J.J., 2003. The global *k*-means clustering algorithm. Pattern Recognition 36, 451–461.

Linde, Y., Buzo, A., Gray, R.M., 1980. An algorithm for vector quantizer design. IEEE Trans. Commun. 28, 84–95.

MacQueen, J.B., 1967. Some methods for classification and analysis of multivariate observation. In: Le Cam, L.M., Neyman, J. (Eds.), Berkeley Symposium on Mathematical Statistics and Probability. University of California Press, pp. 281–297.

Mitra, P., Murthy, C.A., Pal, S.K., 2002. Density-based multiscale data condensation. IEEE Trans. Pattern Anal. Machine Intell. 24 (6), 734–747.

Newman, D.J., Hettich, S., Blake, C., Merz, C.J., 1998. UCI repository of machine learning databases. Available from: <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

Pena, J.M., Lozano, J.A., Larranaga, P., 1999. An empirical comparison of four initialization methods for the *k*-means algorithm. Pattern Recognition Lett. 20 (10), 1027–1040.

Thiesson, B., Meck, B., Chickering, C., Heckerman, D., 1997. Learning mixtures of bayesian networks. Microsoft Technical Report TR-97-30, Redmond, WA.

Tou, J., Gonzales, R., 1974. Pattern Recognition Principles. Addison-Wesley, Reading, MA.