TAMPERE UNIVERSITY OF TECHNOLOGY

Sindhuja Ranganathan

Improvements to *k*-means clustering

Master's Thesis

Examiner: Professor Tapio Elomaa, TUT
Teemu Heinimaki, TUT
Examiner and topic approved by Faculty Council of the Faculty of Computing and Electrical Engineering
On 14 August 2013

Abstract

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Program in Information Technology
Ranganathan, Sindhuja: Improvements to *k*-means clustering
Master's thesis, 42
November 2013
Major Subject: Software Systems
Examiner(s): Professor Tapio Elomaa
Keywords: Algorithm, Computational Complexity, Data mining, *k*-means Clustering, Machine Learning.

Working with huge amount of data and learning from it by extracting useful information is one of the prime challenges of the Internet era. Machine learning algorithms; provide an automatic and easy way to accomplish such tasks. These algorithms are classified into supervised, unsupervised, semi-supervised algorithms. Some of the most used algorithms belong to the class of unsupervised learning as training becomes a challenge for many practical applications.

Machine learning algorithms for which the classes of input are unknown are called unsupervised algorithms. The *k*-means algorithm is one of the simplest and predominantly used algorithms for unsupervised learning procedure clustering. The *k*-means algorithm works grouping similar data based on some measure. The *k* in *k*-means dethe number of such groups available. This study starts from the standard *k*-means algorithm and goes through some of the algorithmic improvements suggested in the machine learning and data mining literature. Traditional *k*-means algorithm is an iterative refinement algorithm with an assignment step and an update step. The distances from each of the clusters centroids are calculated and iteratively refined. The computational complexity of *k* –means algorithm mainly arises from the distance calculation or the so nearest –neighbor query.

Certain formulation of the *k*-means calculations are NP hard. For a dataset with d dimension and N values the computational complexity is $O(kdN)$ for a single iteration. Some of the *k*-means clustering procedures take several iterations and still fail to produce an acceptable clustering error. Several authors have studied different ways of reducing the complexity of *k*-means algorithm in the literature. Particularly, this study focuses mainly on the algorithmic improvements suggested in the related works of Hamerly and Elkan. These algorithmic improvements are chosen to be studied as they generic in nature and could be applied for many available datasets. The improved algorithms due to Elkan and Hamerly are applied to various available datasets and their computation performances are presented.

Preface

This thesis is submitted in partial fulfillment of the requirements for Master's degree in Software Systems. First I would like to thank my professor and supervisor Prof. Tapio Elomaa for this opportunity to do this thesis as part of my studies at Tampere University of Technology. I would also like to thank Teemu Heinimaki for his guidance in the process of writing the thesis. Writing this thesis has been hard but in the phase of writing I feel I have learned a lot about clustering. I would like to express my gratitude to my husband, my parents, relatives, friends and almighty for providing such a wonderful opportunity to study and also for being supportive throughout. I would like to thank my son Anirudh Gopalan for cooperating with me to complete my degree. Finally I would like thank my friends in Tampere for their support and cooperation during my studies.

Date: Signature:

Contents

1. Introduction

Analyzing huge amounts of data generated in the social media, Internet and various walks of life and extracting useful information is a nontrivial task and some parts of machine learning and data mining are used to automate these tasks.

Machine learning can be classified broadly into two different types namely the inductive learning and deductive learning. In deductive learning theory, learning happens with existing knowledge and deduces new knowledge from the preexisting knowledge. Deductive reasoning uses arguments to move from premises which are given and assumed to be true to conclusions which must be true if the premise is true. In inductive reasoning, rules and patterns are extracted out of sub class of large data sets which can then be generalized. Inductive reasoning takes examples and generalizes rather than starting with existing knowledge. In inductive reasoning the premise of an argument is believed to support the conclusion but do not ensure that it’s true. In general, inductive logic takes examples of a concept and tries to build the general description of the concept. Mostly, the examples are described by using attribute- value pair. The other way of representing objects in inductive learning is by using relational techniques. The most common algorithms that are used for Inductive reasoning are the ID3, FOIL and ILP.

In a broader sense, Inductive methods are characterized as search methods over some hypothesis space constraint by some biases. The hypothesis space are many at times big and to effectively search them they are limited by relational learners introduce partial order between the hypotheses and then this ordered space is searched according to the bias of each inductive method.

Traditional Machine learning approaches use both the inductive and deductive steps. In the inductive step the model is learnt from the unprocessed data and in the deductive step the model is applied to predict the behavior of new data.

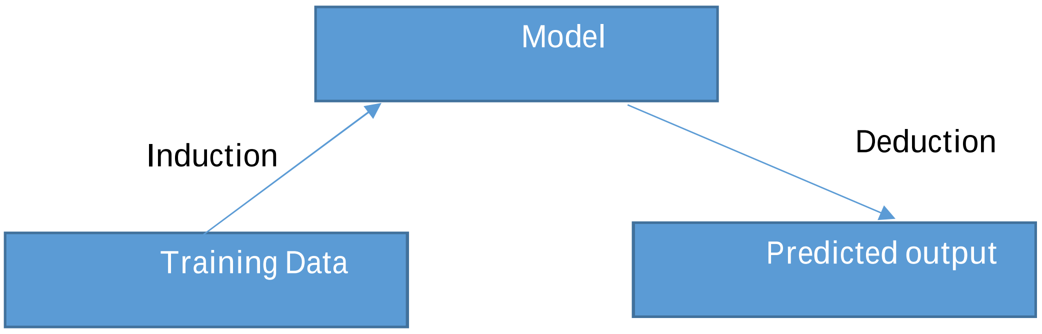


Figure 1: Inductive and Deductive Learning

Due to the random nature and non-availability of a prior information about such data unsupervised techniques become a natural choice for these data. One of the important methods in unsupervised learning is clustering. Clustering is defined as partitioning the set of data points into subsets such that a specific criterion is optimized. Each data point is assigned to a cluster and the criterion which is to be minimized is the average squared distance between the data point and its corresponding cluster center.

Clustering is used in mining data, pattern recognition; applications like marketing for example use it to find customers with similar behavior, biological applications use it to classify plants and animal features, insurance to identify frauds, earth quake studies to identify dangerous zones by clustering observed epicenters. A general approach to clustering is to view it as density estimation problem. In density estimation based clustering probability density function is estimated for the given dataset to search for the regions that are densely populated. There are several algorithms to solve the problem. Some of the widely used algorithms are EM algorithm and *k*-means clustering algorithm.

The *k*-means algorithms can be viewed as a special case of EM algorithm. The *k*-means algorithm is explained elaborately later next chapter. The *k*-means algorithm minimizes a distortion function in two steps, first by finding an optimal encoder which assigns index of the cluster which can be viewed as the expectation step. Then the cluster centers are optimized which can be seen as the maximization step. The generic EM algorithm finds cluster for the dataset by mixture of Gaussian distributions. The mixture of Gaussian distribution can be seen as a probabilistic interpretation of the *k*-means clustering. As explained earlier, the EM algorithm alternates between two steps. In Expectation step, probability of each data point associated with the cluster is determined. In Maximization step, parameters of the cluster are altered to maximize the probabilities.

Several other clustering algorithms which are related to the *k*-means are the geometric *k*-center, Euclidean *k*-medians and other such location based algorithms, and most of them do not have efficient solution as they are NP-hard. But asymptotically efficient approximation of such algorithms exists [1] [2]. Large scale factors make these algorithms impractical. Given such complexity issues, still the attractiveness of the *k*-means algorithm stems from the simplicity, convergence properties and its flexibility. Some of the major drawbacks of the *k*-means algorithm are its slow speed, its scales poorly for larger datasets and the solution may converge to a local minimum.

There have been many approaches to work around these issues mentioned above. Pelleg and Moore in [3] propose a method of geometric reasoning for accelerating *k*-means clustering. Pelleg and Moore in [4], Har-Peled in [5] and Kanungo in [6] have proposed methods to efficiently implement *k*-means algorithm. Hamerly in [7], Hamerly and Elkan in [8] have proposed algorithm to deal with the '*k* 'in *k*-means and about learning the number of clusters needed in *k*-means algorithm. Elkan in [9] discuss about the speed and limitations related to the speed of the *k*-means algorithm. Stochastic global optimization methods like simulated annealing, genetic algorithms have been developed to solve the problem of initializing methods of *k*-means clustering. But these techniques have not been widely accepted. In this thesis, we study the algorithmic improvements to *k*-means clustering algorithm particularly the improvements made by Hamerly [7] and Elkan [9]. The rest of this thesis is organized as follows: In Chapter 2 we review *k*-means clustering algorithm in general and its variants. Chapter 3 deals with the methods to optimize *k*-means. In particular we focus the work of Charles Elkan, Greg Hamerly. In Chapter 4 we present an empirical evaluation of the algorithms in Chapter 3.In Chapter 5 we discuss on the speculation about further improvements to *k*-means clustering.

2. *k*-means and its variants

This chapter discusses the principle of clustering and various ways of clustering and about different *k*-means algorithm and its variants.

2.1 Clustering: Principle and methods

Clustering aims at grouping data points that are close or similar to each other and to identify such clusters in an unsupervised manner. Figure 2 illustrates to identify four clusters and its centers into which the input data is divided.

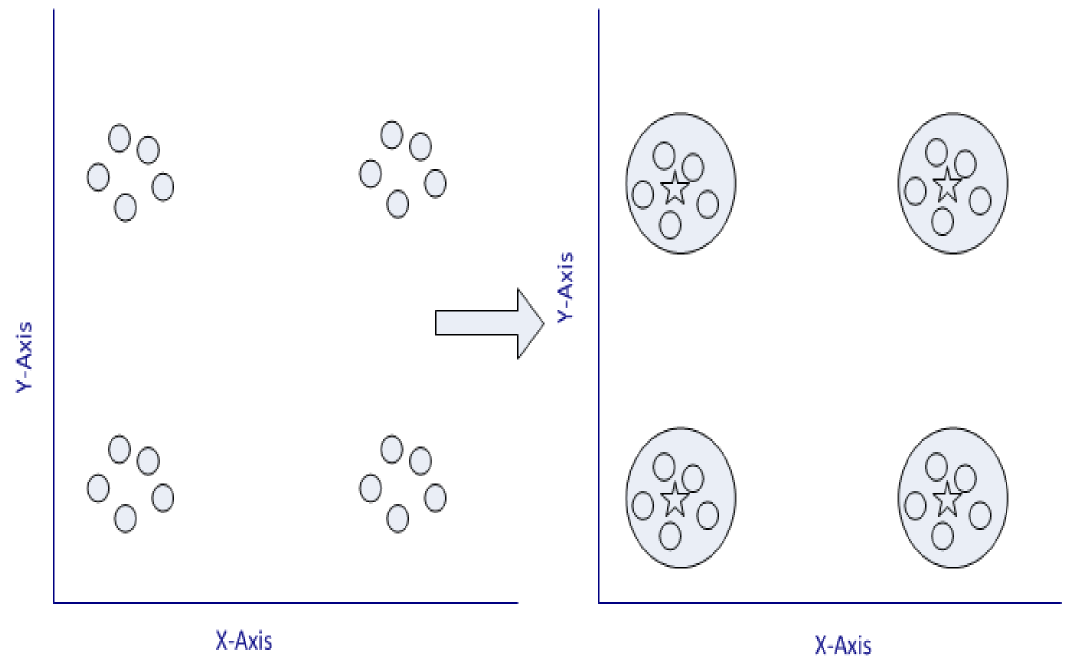


Figure 2: Clustering of data

Two well-known methods of clustering are hierarchical clustering and the partitioned clustering. Figure 3 gives the overall view of classifications in clustering methods. Partitional clustering algorithms find all the clusters simultaneously as a partition of the data and do not impose any sort of hierarchy. In many practical scenarios, there is an inherent hierarchy. The clusters have sub classes within them, and the subclasses might in turn have their own subclasses. Such classifications are hierarchical and they can be partitioned properly by hierarchical clustering. For example, automobiles form a class and passenger automobile is a subclass and cars are a subclass of the passenger automobile and various manufacturer of this car can be a subclass of cars. Figure 4 shows such an example.

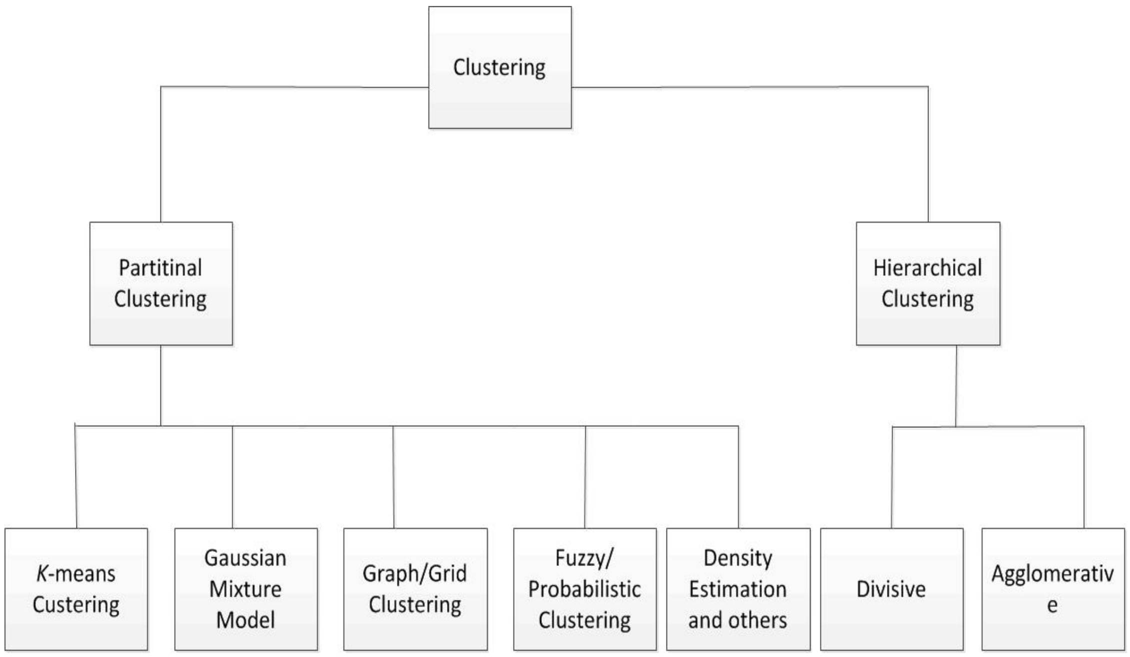


Figure 3: Classifications of clustering

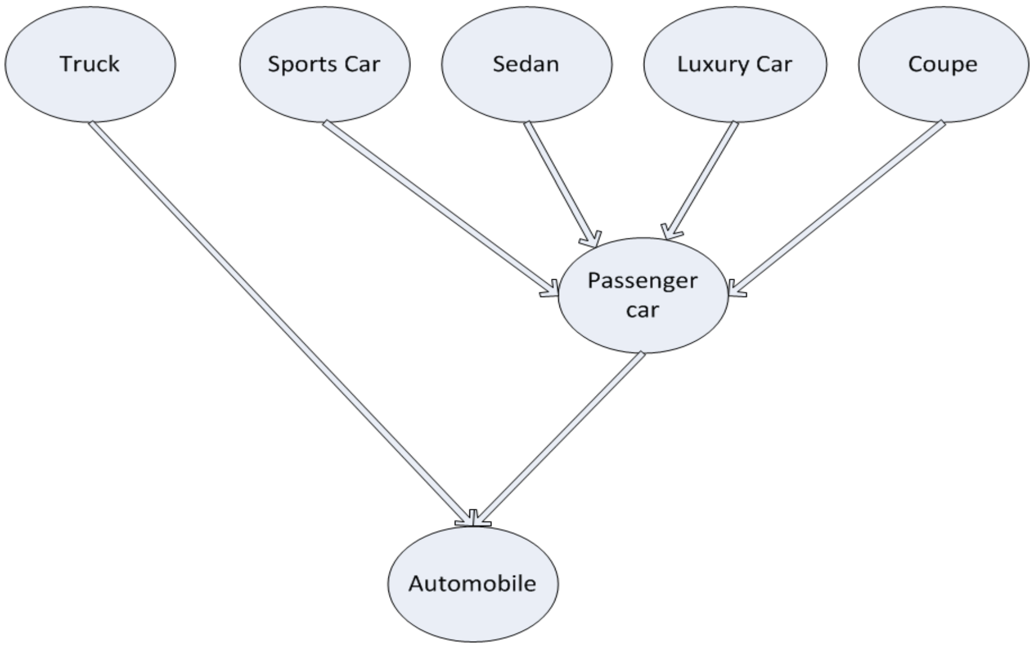


Figure 4: Example data for hierarchical clustering

In hierarchical clustering, each data point is assigned a cluster and there is a hierarchy of clusters which is built at the end of the clustering process. There are two approaches of hierarchical clustering; agglomerative clustering and divisive clustering. In agglomerative clustering, each data point is a cluster and pair of clusters is merged as it moves up in hierarchy in bottom-up fashion. In contrast, divisive clustering works in top-down fashion where it starts with one cluster and it is split recursive as it moves towards bottom. The clusters closer to the leaf are recursively combined to form bigger clusters they then are further combined recursively to form one final cluster at the top. Some of the well-known hierarchical methods are single-link method and complete-link method. For both single link and complete-link method of hierarchical clustering, in the beginning, each data point is a cluster of its own. Figure 5 depicts the single-linkage hierarchical clustering example. Figure 6 depicts the complete-linkage hierarchical clustering example. The clusters are then sequentially combined into larger clusters, until all data point end up in one cluster. The single-link method is different from the complete-link method from the perspective of which two elements gets combined. For the single-link method the nearest-neighbors gets combine and for the complete-link method the farthest-neighbors gets combined. In partition clustering, the dataset is divided into clusters, such that each cluster has at least one data point and each data point has one cluster. The well-known partitional algorithm is *k*-means. There are many variants of *k*-means. So to avoid confusion, we refer to the original *k*-means algorithm as naïve *k*-means.

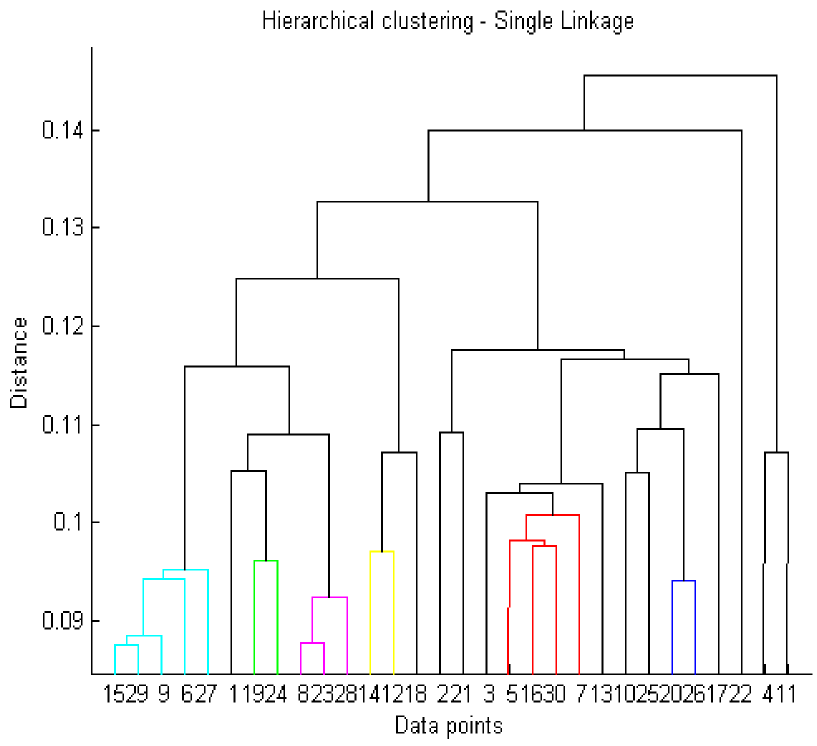


Figure 5: Dendrogram obtained for Single linkage hierarchical clustering of a random dataset with 3 dimensions

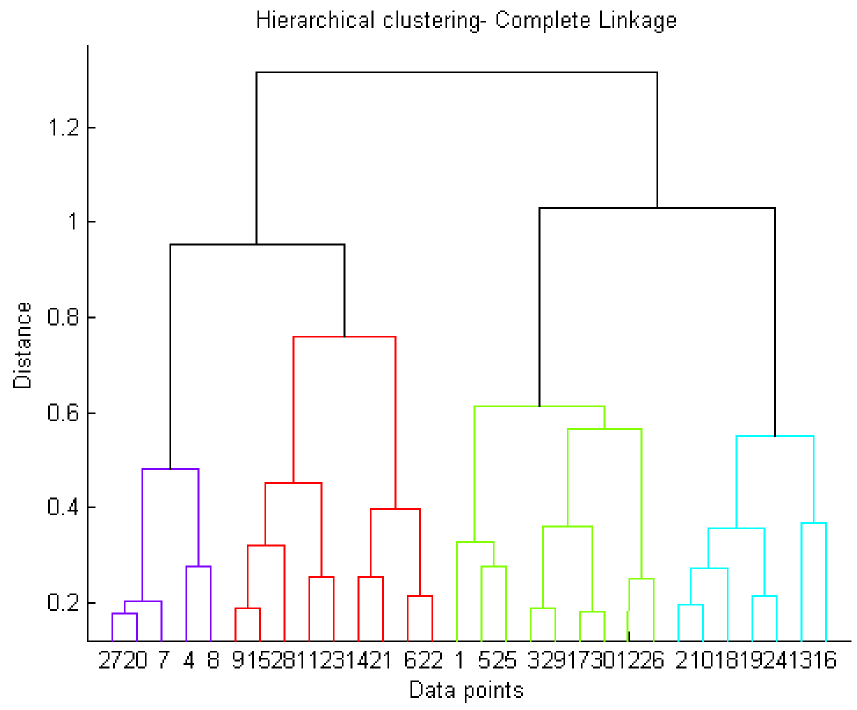


Figure 6: Dendrogram obtained for complete linkage hierarchical clustering of a random dataset with 3 dimensions

2.2 Other approaches to clustering

Even though, *k*-means and extensions to *k*-means are some of the most popular algorithms for clustering, several other methods of clustering exist. Here in this section they are briefly stated and relevant publications are listed for deeper and more thorough discussions of these methods.

The other well-known clustering algorithms can be broadly classified under three themes they are probabilistic or density based methods, graph theoretic methods and the information theoretic methods [10]. The Density based methods are attractive due to their inherent ability to handle arbitrarily shaped clusters. The performance of the density based algorithms is poorer while handling high-dimensional data. When the data is high-dimensional the feature space is sparse and the differentiation between high density regions from low-density regions becomes difficult.

The graph theoretic clustering also known as the spectral clustering, uses weighted graphs to represent data points as nodes. The edges are weighted by their pair-wise similarity. The main idea is to partition the nodes into two subsets *A* and *B* such that the cut size is minimized. The algorithm used initially to solve this problem is the minimum cut algorithm. Using this algorithm results in imbalanced cluster sizes. The ratio cut and the normalized cut algorithms improve upon the minimum cut algorithm. More information on this can be found in [10].

The information theoretic method like the entropy method assumes that the data is generated by using a mixture model and each cluster is modeled using a semi parametric probability density. In these methods, the expected entropy of the partitions over the observed data is minimized. The other popular method is the information bottleneck method which is a generalization of the rate-distortion theory. This method adopts a lossy data compression view. When a joint distribution over two random variables is given, this algorithm compresses one of the variables while retaining the maximum amount of mutual information with respect to the other variable. For example, let us assume we have a document and let the words be the two variables, the words are clustered first such that the mutual information with respect to documents is maximally retained, and using the clustered words, the documents are clustered such that the mutual information between the clustered words and the clustered documents is maximally retained [10].

2.3 Naïve *k*-means

Given a set $S = \{x_1, x_2, \dots, x_N\}$ of N instances and each instance is a vector of d attribute values $X_i = \langle a_{i1}, a_{i2}, \dots, a_{id} \rangle$, naïve *k*-means groups them in to k clusters $C = \{c_k, k = 1, \dots., k\}$. For simplicity, we assume all the attributes have a numerical value i.e. $X_{i,j} \in \mathbb{R}\ \forall i, j, 1 \leq j \leq d, 1 \leq i \leq N$. If we want to find some structure among the instances in S, then one of the unsupervised learning approaches that can be used is to cluster the instances by ordering. Several approaches exist to perform clustering but predominantly used algorithm is the naïve *k*-means algorithm [11].

The algorithm partitions the dataset so that the empirical mean of squared errors between the data points and the cluster is reduced. This method of minimizing has been proved to be NP hard. There are three parameters on which the algorithm depends on. They are cluster initialization, number of clusters, *k* and a distance metric. Initialization of clusters is an important parameter because the stability and quality of the final clustering depends on the initialization of the clusters. Improper initialization methods may lead to the problem of *k*-means converging to local minimum. Consider a dataset having k clusters and *k*-means is initialized with k' clusters. If there is at least one center in each cluster here and if $k' = k$ then the initial centers tend to remain in the same cluster where it has to be placed. If $k' > k$ then this leads to different clustering which results in instability. If $k' < k$ then there are some clusters without centers and centers does not move between clusters. There have been several methods for cluster initialization. Most commonly, the algorithm initializes the clusters by randomly choosing data points from the dataset before updating and randomly partitioning the dataset. The other commonly used method is random restart. It is an empirical way where the algorithm is executed with different number of clusters and the resulting sum of squares is measured. Based on the measurement the clustering with minimum squared error is selected to avoid the problem getting stuck with local minimum. One of the recent methods for cluster initialization is *k*-means++. A detailed summary of this algorithm is given in Algorithm 1.

Choosing a right value for *k* had been a critical issue. Wrong choice of *k* may result to incorrect clustering that is why *k*-means algorithm is run with different value of *k* and the partitions which are most relevant to the domain are selected. Finally, the choice of metric plays an important role in naive *k*-means clustering. The distance metric that is usually used is Euclidean distance metric to find the distance between the data point and the cluster centers. Using any other metric other than Euclidean metric may prohibit the algorithm from converging. Generally metric space is defined as set that has the notion of distance between the elements in the set. Let $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ be vectors in $\mathbb{R}^n$ then, Euclidean distance is defined as, $d(X, Y) = \|X - Y\| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. Some of the well-known modifications of *k*-means clustering obtained using various other distance metrics are spherical *k*-means and *k*-medoid. **Error! Reference source not found.** describes the *k*-means algorithm. The main steps in the algorithms described in plain English are

1. Select an initial partition with *k* clusters
2. Generate a new partition by assigning each pattern to its closest cluster center.
3. Compute new cluster centers.
4. Continue to do steps 2 and 3 until memberships finalize.

```
Input: k (Number of Clusters), X = $\{x_1, \dots, x_N\}$(N data points in d dimension), the initial locations of the centers C= $\{c_j\}$.

Output: $cluster\ centers\ \ C$

    $kmeans(X \in \mathbb{R}, N \times d, k, C)$

1.  while any $c_j$ change location do
2.      $for\ \ i \in \{1 \dots N\}$
3.          $class(x_i) \leftarrow argmin_J \|x_i - c_j\|$
4.      end for
5.      $for\ \ j \in \{1 \dots k\}\ do$
6.          $c_j \leftarrow \sum_i I((class(x_i) = j)(x_i)) / \sum_i I(class(x_i) = j)$, ( $I$ is the indi-
function)
7.      end for
8.  end while
9.  return C
```

Algorithm 1: Naïve k-means

Example: Consider the random data set *X* of 100 two-dimensional instances and *k* =2. On applying *k*-means for the dataset, we obtain 2 clusters illustrated in Figure 7 with their corresponding centers marked with a black “X” which is obtained after few iterations until which the centers does not move.

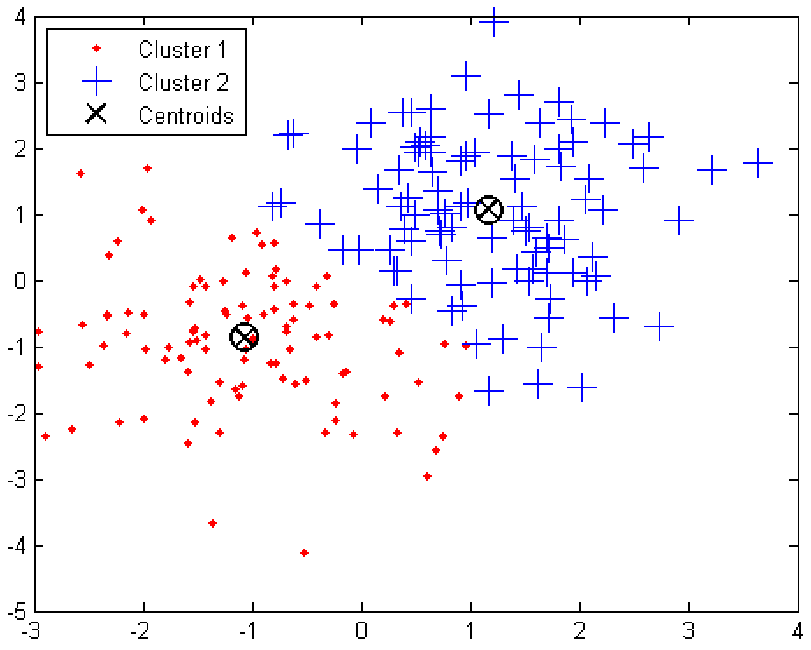


Figure 7: Dataset after k-means clustering

Figure 8, shows how an improper choice of *k* in *k*-means would cluster the data points. The quality of the result obtained in Figure 8 can be improved by using a cluster initializing method *k*-means++ to initialize the data cluster and then apply *k*-means.

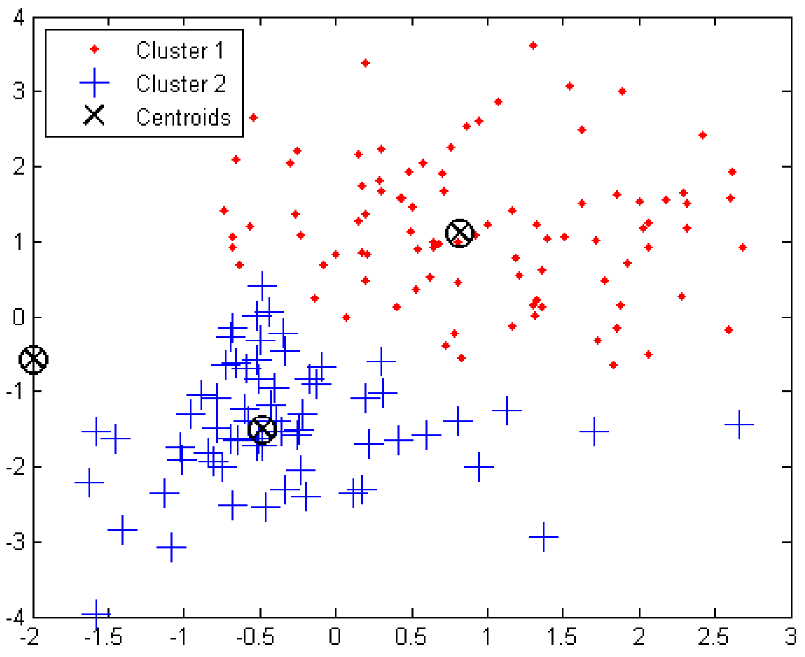


Figure 8: Result of wrong clustering due to incorrect value of k

The *k*-means algorithm is simple and efficient because only point-center distances are calculated and not point-point distances. The running time of a single iteration is $O(kdN)$.

2.4 Parameters for *k*-means algorithm

The *k*-means algorithm needs three different parameters which are defined by the user namely, the number of clusters *k* cluster initialization and the distance metric. A correct choice of *k* results in better clustering with a smaller number of iterations for the centroids to converge. But a bad choice of *k* increases the number of iterations for the centroids to converge and reduces the performance. Therefore it is not efficient to run the algorithm for large datasets and with bad choice of *k* since it requires several iterations. Sometimes in last few iteration the centroids move little. Since continuing such expensive iterations reduces the performance drastically, there must be some convergence criteria so that the iteration stops when the criteria is met.

k-means algorithm typically uses Euclidean distance metric for computing the distance between data points and the cluster centers. This results in *k*-means finding spherical or ball-shaped clusters in data. *k*-means might also use several other distance measures such as the L1 distance, Itakura-Saito distance or Bergman distance or Mahalanobis distance. In case of using Mahalonobis distance metric the clusters become hyper-ellipsoidal.

2.5 *k*-means++

k-means++ is a method to initialize the number of cluster *k* which is given as an input to the *k*-means algorithm. Since choosing the right value for *k* in prior is difficult, this algorithm provides a method to find the value for *k* before proceeding to cluster the data. More on this algorithm can be found in [12]. The algorithm is presented as Algorithm 2.

```
Inputs: Dataset X(N number of data points in d dimension)

Step 1. Choose one center uniformly at random from the dataset X.

Step 2. Choose another center $c_j$, $c_j = x \in X$ with probability $\frac{v(x)^2}{\sum_{x \in X} v(x)^2}$ where $v(x)$ is the shortest distance between the data point x and its closest center which is chosen.

Step 3. Repeat 2 until k centers are chosen.

Step 4. Perform k-means clustering using the k centers as the initial centers.
```

Algorithm 2: k-means++

2.6 EM Algorithm

It is well known that the *k*-means algorithm is an instance of Expectation Maximization (EM) algorithm which is a general algorithm of density estimation. This algorithm is based on distance. EM algorithm is model based iterative algorithm for solving the clustering problem where the data is incomplete or considered incomplete. EM algorithm is an optimization algorithm for constructing statistical models of the data. In this algorithm each and every data instance belongs to each and every cluster with a certain probability. EM algorithm starts with initial estimates and iterates to find the maximum likelihood estimates for the parameters. Closely following reference [13] EM algorithm is shown here in this section.

Given a dataset $\{x_i\}_{i=1}^{N}$ the task of assigning a cluster for each instance in the dataset, is the goal that we aspire for. Let there be N data points in the dataset and let us assume that the number of clusters is *k*. Let the index of the cluster be modeled as a random variable $z = j$ and let its probability be given by a multinomial distribution satisfying

$\sum \pi_j = 1$, Such that

$$\pi_j = p(z = j), \forall j, j = 1, \cdots k$$

It is assumed that $p(x|z = j) \sim N(\mu_j, \sigma_j I_j)$ is a Gaussian distribution. I_j denotes the identity matrix of order j. The unknown parameters of the model namely the mean μ_j, variance $\Sigma_j = \text{diag}\{\sigma_1, \sigma_2 \cdots \sigma_j\}$ and the distribution function π_j are estimated.

$$\theta = \{\mu_j, \Sigma_j, \pi_j\}_{j=1}^{k}$$

$$p(x|\theta) = \sum_{z=1}^{k} p(x|z,\theta) p(z|\theta) = \sum_{j=1}^{k} p(x|z = j,\theta)\pi_j ,$$

where z is an unknown hidden variable. The total log likelihood of all data is given by

$$l(\theta, D) = \log \prod_{i=1}^{N} \sum_{j=1}^{k} \pi_j \exp\left\{-\frac{\|x_i - \mu_j\|^2}{2\sigma_i^2}\right\}$$

The parameter values that maximize the likelihood function $l(\theta, D)$ are the ones that are chosen. Here D denotes the data. This optimization is complicated and to solve this some of the unknowns are assumed to be known, while estimating the others and vice versa [13].

For each class, the conditional expectation of $z = j$ given the data and the parameters

$$\omega_j = p(z = j | x, \theta) = \frac{p(x|z = j,\theta) p(z = j|\pi_j)}{p(x|\theta)} = \frac{\pi_j N(x|\mu_j, \Sigma_j)}{\sum_{i=1}^{k} \pi_i N(x|\mu_i, \Sigma_i)}$$

Since each point x contributes to ω_j in some proportion, for particular x_i we have

$$\omega_{ij} = \frac{\pi_j N(x_i|\mu_j, \Sigma_j)}{\sum_{i=1}^{k} \pi_i N(x_i|\mu_i, \Sigma_i)}.$$

The optimization algorithm is called EM and has the following steps: Assume we have some random initial estimates of the means and variances of the el $\{\mu_j^{(0)}, \Sigma_j^{(0)}, \pi_j^{(0)}\}$. Algorithm 3, describes the EM algorithm.

EM Algorithm

Initialize: means and variances of the model $\left\{\mu_j^{(0)}, \Sigma_j^{(0)}, \pi_j^{(0)}\right\}$

Step 1. Expectation: Using the estimates of parameters $\theta^{(t)} = \left\{\mu_j^{(t)}, \Sigma_j^{(t)}, \pi_j^{(t)}\right\}$, compute the estimate of ω_{ij}

$$\omega^{(t)}_{ij} = p(z = j \mid x_i, \theta^{(t)}) = \frac{\pi_j^t p(x_i \mid z_i = j, \theta^{(t)})}{\sum_{m=1}^{k} \pi_m^t p(x_m \mid z_m = m, \theta^{(t)})}$$

Step 2. Maximization: Using estimates of $\omega_{ij}^{(t)}$, update the estimates of the model parameters

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{N} \omega^{(t)}_{ij} x_i}{\sum_{i=1}^{N} \omega^{(t)}_{ij}}$$

$$\sigma_j^{(t+1)} = \frac{\sum_{i=1}^{N} \omega^{(t)}_{ij} \left\| x_i - \mu_i \right\|^2}{\sum_{i=1}^{N} \omega^{(t)}_{ij}}$$

$$\pi_i^{(t+1)} = \frac{1}{N} \sum_{i=1}^{N} \omega^{(t)}_{ij}$$

- **Step 3.** Repeat steps expectation and maximization until the parameter change gets small enough.

Algorithm 3: EM algorithm

2.7 *k*-medoids

The *k*-medoids is also a partition algorithm like *k*-means. The *k*-medoids algorithm is briefly described here. For a detailed discussion of the *k*-medoids algorithm please see [14] and the references quoted in there. This algorithm clusters data points based on closest center like the *k*-means algorithm. Unlike *k*-means algorithm, which replaces each center by the mean of points in the cluster, *k*-medoids replaces each center by the medoids of points in the cluster. Medoids are defined usually as the centrally located data point of the cluster. Therefore in *k*-medoids, the cluster centers are found within the data points themselves. *k*-medoids is less affected by noise and other outliers as compared to *k*-means clustering [14]. This can be attributed to the cluster formation based on medoids.

The *k*-medoids algorithm has the same properties as *k*-means.

- The algorithm always converges and
- It converges at local minimum.

The *k*-medoids algorithm is shown in Algorithm 4.

Algorithm: *k*-medoids

Inputs: *k*, number of cluster; N, number of data points.

Output: Set of k clusters which minimizes the dissimilarity measure of all the data points to their nearest medoid.

Step 1. Randomly select *k* data points as initial medoids *m*.

Step 2. Assign remaining data points to the cluster with closest medoid.

Step 3. Determine new medoid

- For each medoid *m* and data points *x* associated to that medoid swap *m* and *x* and find the distance between the remaining data points and data points associated to the medoid. Select that data point with minimum distance as new medoid.

Step 4. Repeat steps 2 and 3 until there is no change in the assignment.

Algorithm 4: k-medoids

Some of the key disadvantages of *k*-medoids algorithm are,

- The value of *k* is required in advance,
- *k*-medoids is not efficient.

2.8 Fuzzy *k*-means clustering

The traditional *k*-means clustering algorithm suffers from serious drawbacks like difficulty in finding the correct method for the cluster initialization, making a correct choice of number of clusters (*k*). Moreover *k*-means is not efficient for overlapped data set. There have been many methods and techniques proposed to address these drawbacks of *k*-means. Fuzzy *k*-means is one of the algorithms which provide better result than *k*-means for overlapped dataset.

Fuzzy *k*-means was introduced by Bezdek [15].The fuzzy *k*-means algorithm is also called fuzzy c-means. Unlike naive *k*-means which assigns each data point completely belonging to one cluster, in fuzzy c-means each data point has the probability of belonging to a cluster. This allows data point of data set *X* to be a part of all centers of set *C*. For example, points on the edges of the clusters might belong to a cluster with lesser degree than those data points belonging to the same cluster at its center. This algorithm is mainly used for datasets in which the data points are between the centers. The algorithm works on the objective to minimize the following function,
$F(X, C) = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij}^{m} \left|\left| x_i - c_j \right|\right|^2$.

Here m is any real number greater than 1.u_{ij}, is the degree of membership of data point x_i to the cluster center c_j with the limitation that $u_{ij} \geq 0$ and $\sum_{j=1}^{k} u_{ij} = 1 \ \forall i$. Iterative procedure of optimizing the objective function $F(X, C)$ by updating the degree of membership of the data point x_i to the center c_j and the cluster center c_j results in the clustering of the data.

$$u_{ij} = \frac{\sum_{j=1}^{k} \left|\left| x_i - c_j \right|\right|^{\frac{-2}{m-1}}}{\sum_{j=1}^{k} \left(\left| x_i - c_j \right| \right)^{\frac{-2}{m-1}}}$$

$$c_j = \frac{\sum_{i=1}^{n} u_{ij}^{m} . x_i}{\sum_{i=1}^{n} u_{ij}^{m}}$$

As the value of m increases the algorithm becomes fuzzy. At m around 1 the sharing centers among data points becomes less and it behaves like standard *k*-means [15]. For example consider a one dimensional dataset as depicted in Figure 9.

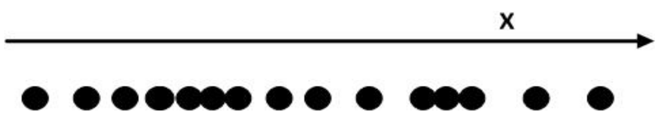


Figure 9: Input mono-dimensional data

We could find two clusters A and B based on the data points associations. On applying *k*-means to the above dataset, each data point is associated to the centroid close to it as depicted in Figure 10.

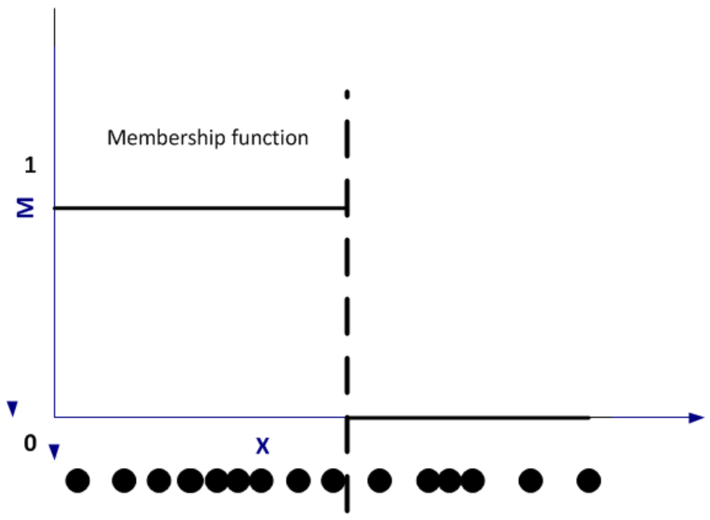


Figure 10: Clustered using k-means

If fuzzy *k*-means clustering approach is used on the dataset, the data point does not exclusively belong to a cluster instead it is in the middle way. There is a smoother line to indicate that every data point may belong to more than one cluster as in Figure 11. More information on this example can be found in [16].

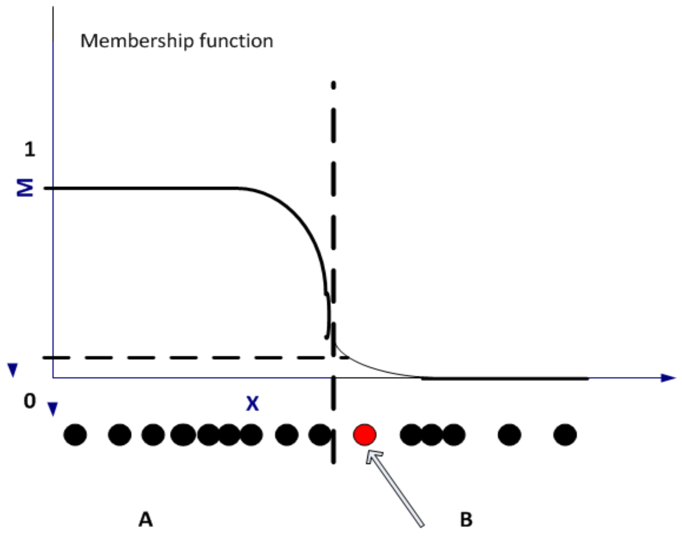


Figure 11: Clustered using fuzzy k-means

3. Improvements to *k*-means clustering

3.1 Challenges of *k*-means clustering

Naïve *k*-means algorithm suffers from the problems that were discussed in the previous section. One of the main disadvantages is the high time complexity. In any distance based clustering approaches like *k*-means, all data points or existing cluster are inspected in the same manner irrespective of their distance in order to make a decision over clustering. This scanning of all data points or the clusters does not have linear time scalability and it fails for big datasets. The *k*-means is inherently slow because *k*-means clustering algorithm takes $O(kdN)$ for single iteration. Thus algorithm becomes impossible to be used for large datasets as it would take several iterations. Distance calculations are one of the reasons which make the algorithm slow. Determining the number of clusters in advance has been a challenge in *k*-means. Increasing the value of *k* reduces error resulting in clustering. The error is the sum of the squared Euclidean distances from data points to the cluster centers of the partitions to which data points belong. In extreme case there is a possibility of zero error if clustering is performed by considering each data point as its own cluster. Usually value of *k* is chosen by mostly assumptions, prior knowledge.

There have been several methods proposed to determine the value of *k* or overcome the disadvantages related to the choice of *k* value. Bischof [17] tried to solve the selection of the *k* parameter by formulating the problem as a model selection problem using the minimum description length (MDL) framework. The algorithm starts from a large value of *k* and removes centers whenever that choice reduces the description length. Pelleg and Moore [3] used information theoretic criterion like the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) to choose the value of *k*. The algorithm searches over many values of *k* and scores each clustering model using the BIC. In [18] Steinbach and other authors propose combining agglomerative clustering with *k*-means to get the best of both worlds. This method is called the bisecting *k*-means technique. In this method, initially there is a single cluster which is assumed to be encompassing all the data points and then this cluster is split until there is a cluster for each data point. The process of split can also be stopped at any time by defining a termination criterion for the bisection process.

Several researchers have been looking the *k*-means clustering problem and there have been many approaches to accelerate *k*-means. As a result several elegant and simple methods have been introduced towards scalability and reducing time complexity of *k*-means algorithm. Some of the notable works that this thesis is based on are the algorithms proposed by Pelleg and Moore in [4], Hamerly in [7] and Elkan in [9]. Pelleg and

Moore [3] have proposed a method called *X*-means. The idea of this method is to split the some centroids into two when a fit to the data is achieved. The decision over the split is based on Bayesian Information Criterion (BIC). To accelerate *k*-means with large date sets *X*-means caches information which remains unchanged over iterations. *X*-means algorithm has been proved to be more efficient than *k*-means in [3].This method is not without any disadvantage. BIC based on which the split is made chooses too many centroids when data is not strictly spherical.

Gaussian-means algorithm proposed by Hamerly and Elkan in [8] aims to determine the number of clusters. The idea behind this algorithm is to split the clusters into two. The decision to split the center is based on a statistical test. If not the center is split. Experimental results have demonstrated this algorithm to be more efficient than *X*-means by determining correct number of clusters and also the location of its center. Gaussian-means does not require any other parameter as input except the significance level of the statistical test. In this thesis we are not dealing with the intricate details of determining the number of clusters in *k*-means. More information on this can be found in reference [8].

Pelleg and Moore [4] used geometric reasoning to accelerate *k*-means clustering. In this work they use the *mrkd*-tree data structure which is a special type of *kd*-tree to store the centers of the clusters and hence the savings achieved will be a function of the clusters and not the number of points or data in the dataset. Kanungo in [6], use *kd*-tree based ‘filtering’ where the points which are further away from the centers are ‘filtered out’ and only the nearest neighbor are clustered together. Elkan in [7] propose an algorithm to avoid distance calculations by using one lower bound thereby avoiding inner loop of *k*-means. In [9] the author uses triangle inequality to avoid the distance calculations between the point and the center. In Chapter 3 we concentrate in the work contributed by Elkan, Hamerly, Pelleg and Moore to accelerate *k*-means.

3.2 The Blacklisting Algorithm

Several methods have been proposed to find the nearest center of the data point in *k*-means. One of the methods is to organize the data points into trees. This approach is to update the cluster in groups or bulk instead of updating it point by point. The data points in the group or bulk belong to the hyper-rectangle in the *kd*-tree by nature. To make sure of the correctness of those groupings, we need to make sure that all the points in a given hyper-rectangle belong to a specific center before the sufficient statistics are added to the *kd*-node. According to Pelleg and Moore [4], this gives rise to the idea of owner. In this algorithm, we find those centers that will never be the owner of the hyper plane *h*. If *c* has been identified that it will never become the owner then it eliminates the need to check *c* for the descendants of the hyper-rectangle *h*. The hyper-rectangles are the decision regions for cluster. Hence it is named blacklisting algorithm.

Each and every node of the *kd*-tree stores a set of statistics and "any point associated to the node should have *c* (center) as its neighbor" as stated in [4]. This eliminates the need to perform certain arithmetic operations to update the centroids of the cluster, since they are pruned. This increases the speed of the algorithm drastically. Before proceeding to the algorithm in detail, we need to know some theorems, definitions and basics of *kd*-tree for better understanding which is detailed in [4].

3.2.1 *kd*-tree

The *kd*-tree is a data structure to store finite set of data points from a *d*-dimensional space. It has following characteristics: It is a binary tree. The node is split along a plane into two parts, left subtree and right subtree. The sub trees are split recursively until a leaf is obtained. There are many variants of *kd*-tree based on the splitting plane. The type of *kd*-tree used in blacklisting algorithm is *mrkd*-tree where, the region of node is rectangles or hyper-rectangles with two vectors h_{min}, h_{max}. The root node represented by hyper-rectangle contains all the data points in the set. Each node contains information about all the other nodes in the hyper-rectangle *h*. The statistics that are stored are number of other nodes in the hyper-rectangle, center of the mass and sum of Euclidean norms of the points in the hyper-rectangle. See [19] for more information on *kd*-tree.

Definition 1: ***Given a set of centers C and a hyper-rectangle h, we define by*** $\boldsymbol{owner_c(h)}$ ***a center*** $\boldsymbol{c \in C}$ ***such that any point in h is closer to c than to any other center in C, if such a center exists*** [4]**.**

Theorem 1: ***Let C be a set of centers, and h a hyper-rectangle. Let*** $\boldsymbol{c \in C}$ ***be*** $\boldsymbol{owner_c(h)}$***. Then,*** $\boldsymbol{d(c, h) = min(d(c', h))}$***. Here,*** $\boldsymbol{d(c, h)}$ ***is the distance between the hyper-rectangle and the center c*** [4]**.**

According to this theorem the center which influences the hyper-rectangle is closest to the hyper-rectangle. If there is more than one center in the hyper-rectangle then the one with shortest distance to the hyper-rectangle h is the owner. If both the centers share minimal distance to h then no unique owner exits. So this results in splitting of nodes to find unique owner among the descendants. The following definition defines the owner in case of two centers in a hyper-rectangle.

Definition 2: ***Given a hyper-rectangle*** $\boldsymbol{h}$***, and two centers*** $\boldsymbol{c_1}$ ***and*** $\boldsymbol{c_2}$ ***such that*** $\boldsymbol{d(c_1, h) < d(c_2, h)}$***, we say that*** $\boldsymbol{c_1}$ ***dominates*** $\boldsymbol{c_2}$ ***with respect to h if every point in h is closer to*** $\boldsymbol{c_1}$ ***than it is to*** $\boldsymbol{c_2}$ [4]**.**

Lemma 1: ***Given two centers c_1, c_2 and a hyper-rectangle h such that $d(c_1, h) < d(c_2, h)$, the decision problem "does c_1 dominate c_2 with respect to h?" can be answered in $O(d)$ time, where d represents the dimensionality of the dataset*** [4]**.**

Step 1. All the centers are assigned to the root node.

Step 2. Find the center c which is closest to the hyper-rectangle h. Using Theorem1 and Lemma 1, center c of the node is checked if it influences the hyper-rectangle h.

Step 3. Using Lemma 1, verify if the center c dominates all other centers.

Step 4. If step 3 is satisfied, assign all the data points in that hyper-rectangle to the center c. Update the statistics of the node.

Step 5. If step 3 is not satisfied then, the center c, centers dominated by c and the auxiliary centers are moved down the tree to the child nodes.

Step 6. Repeat step 2 to step 5 for the child nodes until there is only one center which is the owner. Update the statistics of the node.

Algorithm 5: Blacklisting algorithm

Algorithm 5 has been proved to be efficient if the dimensions of the data are within a specific range [4]. If the dimensions increase after a particular value then this method becomes slow. Threshold dimension for blacklisting algorithm is reported as 8.The reason behind that is, as number of dimensions increase distances between data points and centers increase. Therefore, there is little filtering of centers. There are many other disadvantages like constructing a *kd*-tree non-linear data structure increase the memory cost and updating the data set to reflect the changes in data set is expensive.

3.3 Accelerating *k*-means using triangle inequality

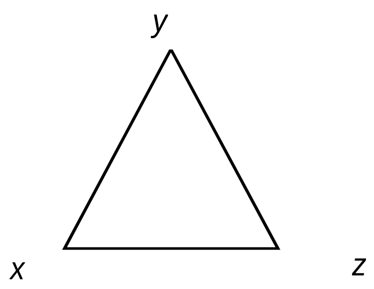
The inefficiency of traditional *k*-means algorithm stems from the fact that *k*-means do not pass information from an iteration of the algorithm to another. As stated by Kanungo in [6] the centers of the clusters do not change for most of the clusters. Using this information coherently would improve the efficiency of the algorithm. Elkan's [9] proposed an algorithm, which uses these findings to avoid unnecessary distance calculations between data point and those cluster centers, which are far away from each other in the *k*-means algorithm, using triangle inequality. The decrease in computations is achieved by avoiding redundant distance calculations. The reduction in distance calculation is achieved by avoiding distance calculation to the points which are further away from the center of the cluster. Conversely, if a point is closer to the center than any other point then calculating the distance can be avoided. The intuitive idea behind the savings in calculation is explained with triangle inequality. The proposed algorithm satisfies the following properties

- It can be used with any initialization methods,
- Result of this algorithm is same as the standard *k*–means algorithm with improved efficiency and it can be used with any distance metric.

Before explaining the algorithm, the concept of triangle inequality which is used in this version of the *k*-means algorithm due to Elkan [9] is briefly shown below.

3.3.1 Triangle Inequality

According to the triangle inequality, for any 3 points x, y, z. $d(x, z) \leq d(x, y) + d(y, z)$ where, *d(.,.)* is the distance metric between any two points $x, y, z \in \mathbb{R}^n$.



3.3.2 Application to the algorithm

Consider *x* to be a data point. Let *y* and *z* be the centers. Data point *x* can be assigned to any of these centers based on its distance calculation. Consider $d(x, z) \geq d(x, y)$, then distance calculation between the point *x* and center *z* can be avoided since it is far away or almost the same as the distance between *x* and *y*.

Following two lemmas are needed to derive lower bounds from the triangle inequality to avoid unnecessary distance calculations as triangle inequality gives only upper bounds.

Lemma 2

If x is a data point and y, z are the centers and $d(y,z) \geq 2 * d(x,y)$ then,

$$d(x,z) \geq d(x,y)$$

Proof

According to triangle inequality

$$d(y,z) \leq d(y,x) + d(x,z).$$

Therefore, $d(y,z) - d(x,y) \leq d(x,z)$

$$d(y,z) - d(x,y) \geq 2 * d(x,y) - d(x,y) = d(x,y)$$

Hence, $d(x,y) \leq d(x,z)$. Figure 12 explains Lemma 2.

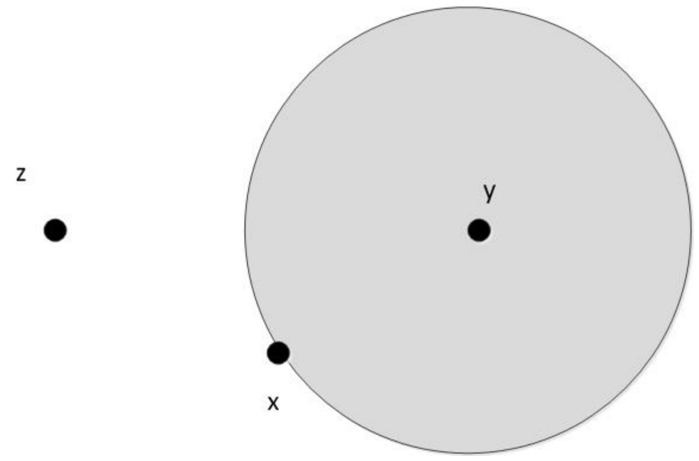


Figure 12: Pictorial representation of Lemma 2

Lemma 3

Consider x is a data point. y, z are the centers then, $d(x,z) \geq max\{0, d(x,y) - d(y,z)\}$

Proof

$$d(x,y) \leq d(x,z) + d(y,z).$$

So,

$$d(x,z) \geq d(x,y) - d(y,z).$$

Moreover, $d(x,z) \geq 0$. Figure 13 explains Lemma 3.

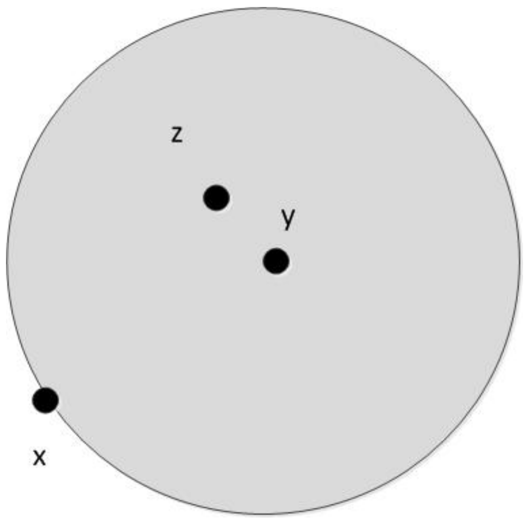


Figure 13: Pictorial representation of Lemma 3

3.3.3 Use of Lemmas in the algorithm

Use of Lemma 2: Let x be a data point, and z is the center to which it is assigned to. Let z' be the other center. According to the Lemma 1,$d(z, z') \geq 2 * d(x, z)$ then,

$d(x, z') \geq d(x, z)$. Hence, the distance $d(x, z')$ can be avoided.

Use of Lemma 3:

Let x be the data point, y be some center and y' be the previous version of the same center. If in the previous iteration, we know a lower bound lb' such that distance $d(x, y') \geq lb'$ from lemma 3 the lower bound for the current iteration is, $d(x, y) \geq max\{0, d(x, y') - d(y, y')\}$ which means,

$$d(x, y) \geq max\{0, lb' - d(y, y')\} = 1$$

Step 1. Initialize all center distances and find the minimum distance calculation per center in order to apply Lemma 2 which means for all centers c and c'. Find the distance $d(c, c')$ and for each center c, find $b(c) = 0.5\left(min_{c' \neq c} d(c, c')\right)$.

Step 2. Remove those points which cannot change their center according to Lemma 2 which means that those data points whose distance between it and to its closest center(upper bound value) is less or equal to $b\{c(x)\}$ (distance from center of x to its other closest center).

Step 3. The remaining data points x are candidates for cluster reassignment.

- Take into account only centers c into which x is not currently assigned to i.e. $c \neq c(x)$.
- Upper bound $ub(x)$ of distance to own center $c(x)$ is strictly larger than the lower bound on the distance between x and c.i.e. $ub(x) > lb(x, c)$.
- The condition derived from Lemma 2 holds for x and c such that $ub(x) > \frac{1}{2} d(c(x), c)$.

Step 3a. Let $r(x)$ be a Boolean indicator indicating whether $ub(x)$ (upper bound of x) is out-of-date. If it is true, then the distance between x and its own center needs to be recomputed. Otherwise the distance is simply set to its upper bound value.

Step 3b. If $c \neq c(x)$ or $ub(x) > lb(x, c)$ holds, we compute the distance $d(x, c)$ between x and c. Reassign x to center c if it closer to x than its previous center $c(x)$.

Step 4. Compute the mean $m(c)$of points assigned to center c.

Step 5.Update lower bounds according to Lemma 3.

Step 6. Update the upper bound $ub(x)$ to reflect the change in the mean $m(c)$value. Now set $r(x)$ to be true to indicate that $ub(x)$ has changed and needs to be recomputed.

Step 7. Finally, replace c by the mean $m(c)$computed in step 4.

Algorithm 6: Elkan's Algorithm on k-means using triangle inequality

Algorithm 6 is made efficient from the start by making the upper and lower bound tight which means that the whenever a center c is found close to a data point x, then it is assigned to that center. This assignment is repeated until the upper and lower bound is exact for each and every data point. This assignment requires many distance calculations. There can also be another method of upper and lower bound assignment by starting from an arbitrary center and proceed with the algorithm but in the later stages it leads to the many distance calculations.

3.4 Making k-means even faster

This algorithm was proposed by Hamerly. He had proposed an algorithm [7] which produces the same result as k-means which is much faster than k-means. Like the previously discussed Elkan's algorithm [9], it avoids unnecessary distance calculations by using triangle inequality and distance bounds. This algorithm is simpler and faster than the Elkan's algorithm. Hamerly's algorithm avoids the inner loop which is the loop that iterates over the k centers and it does not maintain the distance between the data point and second closest center which is called as lower bound as maintained in Elkan's algorithm. This reduces the time complexity of the algorithm thereby making the algorithm fast. This algorithm is fast for the data sets with low dimensions.

The parameters used in the algorithm are described below,
$c(j)$- *cluster center* j *(where* $1 \leq j \leq k$*)*
$x(i)$- *data point i*
b(j)- Distance from $c(j)$ *to its closest other center*
$c'(j)$- *vector sum of all points in cluster* j
$ub(i)$- *distance between* $x(i)$ *and its assigned center* $c(a(i))$

$lb(i)$- *distance between* $x(i)$ *and its second closest center*
$a(i)$- *index of the center to which* $x(i)$ *is assigned*

Step 4 of the Algorithm 7 is the major difference between Elkan's algorithm and Hamerly's algorithm. The inner loop of this algorithm finds the distance between the point and center except for the assigned center. It does not perform the check against lower bounds for each center to find if the distance calculation between the point and center is necessary or not like Elkan’s algorithm. The inner loop is avoided by making sure that the condition in step 3 is greater than $ub(i)$ by performing first bound test and second bound test. These two test checks for the same condition twice. So, this algorithm has more possibility to avoid inner loops. This makes the algorithm fast.

Step 1. Initialize c, x, q, c', ub, lb, a .

Step 2. Update b: For each center find the minimum of the distance between itself and the closest other center.

Step 3. For each data point, find the t, which is the maximum of its lower bound and the half (b). $t \leftarrow \max(b(a(i))/2, lb(i))$

Step 4. First bound test: If the t value is less than upper bound of the data point then tighten the upper bound with that center that is, $ub(i) \leftarrow d(x(i), c(a(i))$and also compute the inner loop by finding the distance between the data point $x(i)$ and all the centers to find the correct one. If the value of t is greater than the upper bound he inner loop is avoided

Step 5. Second bound test: Again check if $t < ub(i)$. If true then find distance between the data point $x(i)$ and all the centers.

Step 6. Based on the number of data points assigned the center and the distance the center has moved, move the center.

Step 7. Update the lower and upper bound as the centers move.

Algorithm 7: Hamerly's algorithm

4. Experiment Results

This chapter discusses the results generated by running accelerating algorithms as described in the Chapter 3. The experiments are performed to compare the time consumption of the following algorithms: naïve k-means, k-means based on *kd*-tree, Elkan's algorithm based on triangle inequality [9]and Hamerly's algorithm [7] by varying various parameters. The parameters include the number of clusters k, dimensions d. The experiments are conducted on Linux machine at Lintula laboratory. The programs used to implement the algorithms are in C++. The results were collected and plotted in Matlab.

4.1 Datasets

There are two types of datasets used in this experiment: synthetic data which is generated using Matlab and some known datasets like KDD Cup, Wine taken from the standard machine learning repositories like the University of California, Irvine, and Machine learning repository. The wine dataset is a multivariate dataset with integer and real values. The dataset are the results of chemical analysis of wine grown in different regions of Italy. The synthetic data is generated using the Matlab's rand function which generates uniform data in the unit interval. The synthetic dataset gives the worst scenario compared to the other datasets available. It is due to the fact that the dataset is not structured properly. In synthetic data sets, most of the data points are at the edge of the cluster than being close to the cluster centers. Results for Pelleg & Moore's algorithm on *kd*-tree are obtained for low dimensional uniform datasets, Wine dataset and KDD Cup dataset. The algorithms are tested against 6 different datasets listed in Table 1.

Table 1: Dataset used for the experiment

DATASET	CARDINALITY	DIMENSIONALITY
Wine	178	14
Random	100000	2
Random	100000	8
Random	100000	32
Random	100000	64
KDD Cup	95413	56

4.2 Experiment

The algorithms discussed in Chapter 3 are implemented using C++. The results for all the algorithms are obtained for six different values of *k* the initial number of clusters for every dataset. Each and every algorithm in this experiment is run 10 times for a combination of *k* and a dataset. The mean value of the 10 trials is tabulated in Table 2. The **Error! Reference source not found.** is used to determine the time consumption of the accelerating *k*-means algorithms, and standard *k*-means algorithm. The results of these experiments are then compared and analyzed from their performance point of view. The time listed in the Table 2 represents the time to calculate the number of point to center distances for the respective algorithms in CPU seconds. One of the primary goal of the algorithms discussed in this thesis, especially the ones by Elkan [9] and by Hamerly [7] is to speed up the traditional naive *k*-means algorithm. It can be easily inferred that when we have more number of distance calculations the time needed will be greater and lesser number of calculation shall result in lesser time consumed. This can be explicitly observed in the tabulation. The dimensions used for the experimentation are 2, 8, and 32 for the uniform dataset and for the Wine & KDD Cup datasets the dimensions are 14 and 56 respectively.

Table 2: Experimental results obtained by running standard, kd-tree, Elkan's and Hamerly's algorithm are under the heading standard, kd-tree, Elkan's and Hamerly's for both synthetic datasets and 2 common datasets and for different k values.

		Time in CPU seconds					
Dataset	**Algorithm**	***k* =3**	***k* = 10**	***k* = 30**	***k* = 50**	***k* = 70**	***k* =100**
KDD Cup n =95413 d =56	Standard	86.7	387.3	2235.1	3896.7	4708.5	7848.9
	Elkan's	36.0	**106.1**	**414.8**	**426.2**	**727.4**	**904.2**
	Hamerly's	**34.2**	108.3	429.8	428.2	794.3	1119.3
	kd-tree	98.1	507.7	3747.4	5543.1	7496.1	10682.3
Uniform random n= 100000 d = 2	Standard	9.8	39.3	252.1	459.0	617.2	1137.4
	Elkan's	13.5	53.4	302.9	564.7	749.2	1339.6
	Hamerly's	7.8	15.6	44.3	**66.7**	**80.7**	**121.8**
	kd-tree	**2.0**	**6.6**	**37.1**	72.9	104.0	201.7
Uniform random n= 100000 d = 8	Standard	20.8	418.8	626.7	1793.0	1946.5	2480.2
	Elkan's	16.7	324.7	489.5	1461.9	1630.3	1062.0
	Hamerly's	**11.2**	**93.3**	**104.2**	**327.9**	**420.7**	**321.7**
	kd-tree	90.7	1118.9	3101.6	4884.1	5280.2	6697.8
uniform random n= 100000 d = 32	Standard	121.8	2475.6	3217.6	9305.4	11191.5	10678.5
	Elkan's	58.7	781.6	976.7	2770.9	3365.6	629.9
	Hamerly's	**53.3**	**479.1**	**568.2**	1781.1	**2445.1**	**1424.3**
	kd-tree	478.3	3338.44	18160.2	17080.1		
Wine n=178 d= 14	Standard	0.0128	0.0461	0.0656	0.1802	0.2244	0.2800
	Elkan's	0.0127	0.0410	**0.0570**	**0.1605**	0.2171	0.1530
	Hamerly's	**0.0124**	**0.0392**	0.0622	0.1666	**0.2102**	**0.1469**
	kd- tree	0.0248	0.0822	0.2134	0.3086	0.3410	0.3798
uniform random n= 100000 d = 64	Standard	578.0	3393.4	6555.6	8530.0	10927.7	11296
	Elkan's	124.1	610.9	1133.7	1467.3	1834.3	1834.3
	Hamerly's	126.2	528.8	1149.0	1709.8	2312.7	2312.7
	kd- tree	1216.17	4161.56	11630.9	13738.6		

4.2.1 Effect of dimensionality

The effect of varying the dimensions on the time complexity of the algorithms is evaluated. Figure 14 is plotted for the datasets with 100000 data points and number of clusters *k*=50. The total time taken by algorithms for dimension d=2, d=8, d=32, d=64 are plotted in Figure 14. The inference made from the result is that naïve *k*-means has the high time consumption compared to other algorithms that are used in the experiment. Generally, as the dimensions increases the time taken by naïve *k*-means also increases. There is a negligible decrease in the time for 64 dimensional data which could be attributed to the statistical averaging of the values obtained by 10 trials. In case of *kd*-tree as the dimension is small it consumes less time but still poorer than Hamerly's algorithm as in Figure 14. As the dimension increases it performs the worst among naïve, Elkan's and Hamerly's algorithm. Similar to naïve *k*-means there is a decrease in time consumption for *kd*-tree for 64 dimensional data which could be attributed to the statistical averaging of the values obtained by 10 trials. According to Figure 14, Elkan's algorithm initially performs poorly compared to Hamerly's as the time consumption increases but in the later as the number of dimensions increases Elkan's algorithm becomes efficient whereas for Hamerly's algorithm it is the reverse. Initially it consumes less time when dimensions are small. After a particular value of dimension Hamerly's algorithm performs poorly when compared to Elkan's. As per our experimental result, when dimensions are between 30 and 35 the performance of Hamerly's algorithm starts decreasing. Therefore, Elkan's algorithm can be used for high dimensional data and Hamerly's algorithm for low dimensional data.

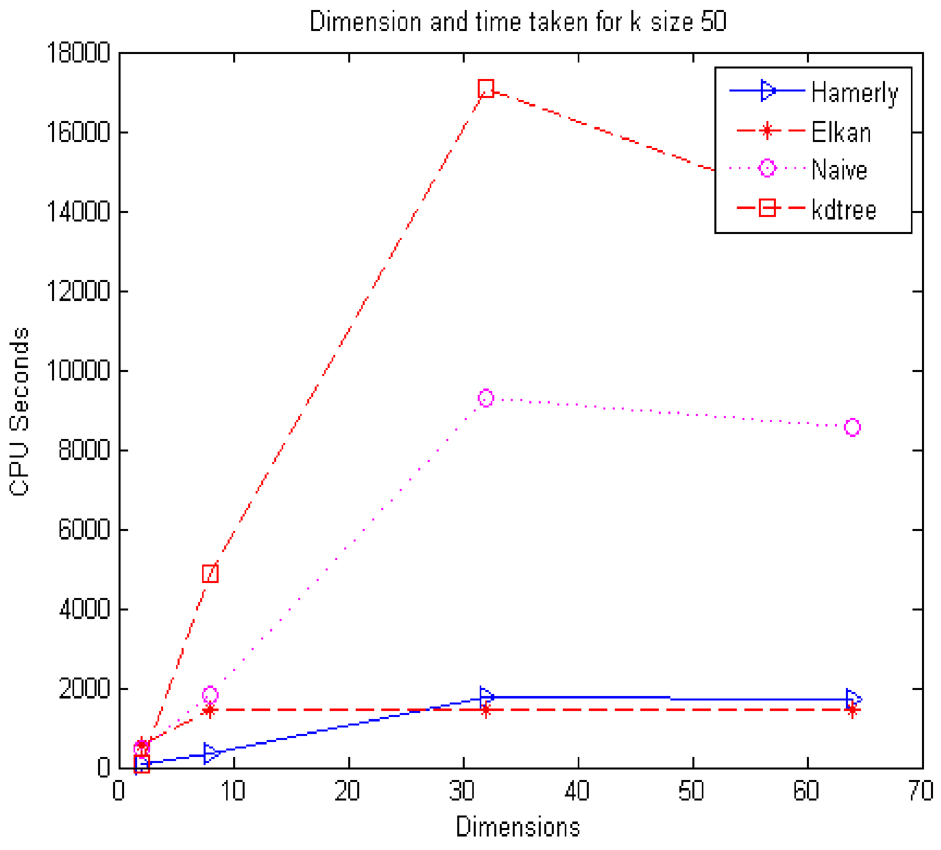


Figure 14: effect of dimensionality for uniform random dataset with 2, 8, 32 and 64 dimensions

4.2.2. Effect of change in number of clusters

The effect of variation of the number of centers on the time complexity of the algorithm is seen in this experiment. The result obtained in Figure 15 is for uniform random dataset with dimension 2 and 100000 data points. As we could observe in Figure 15, as the number of clusters increases the time consumption also increases in case of naïve *k*-means. This is due to the increase in number of distance calculations. Even in case of *kd*-tree the situation is better to naïve *k*-means. In case of Elkan's algorithm, the performance is poor than naïve *k*-means. This due to the effect of dimensionality, that it performs poor for low dimensional dataset. Hamerly's algorithm performs the best among other three algorithms with least time consumption.

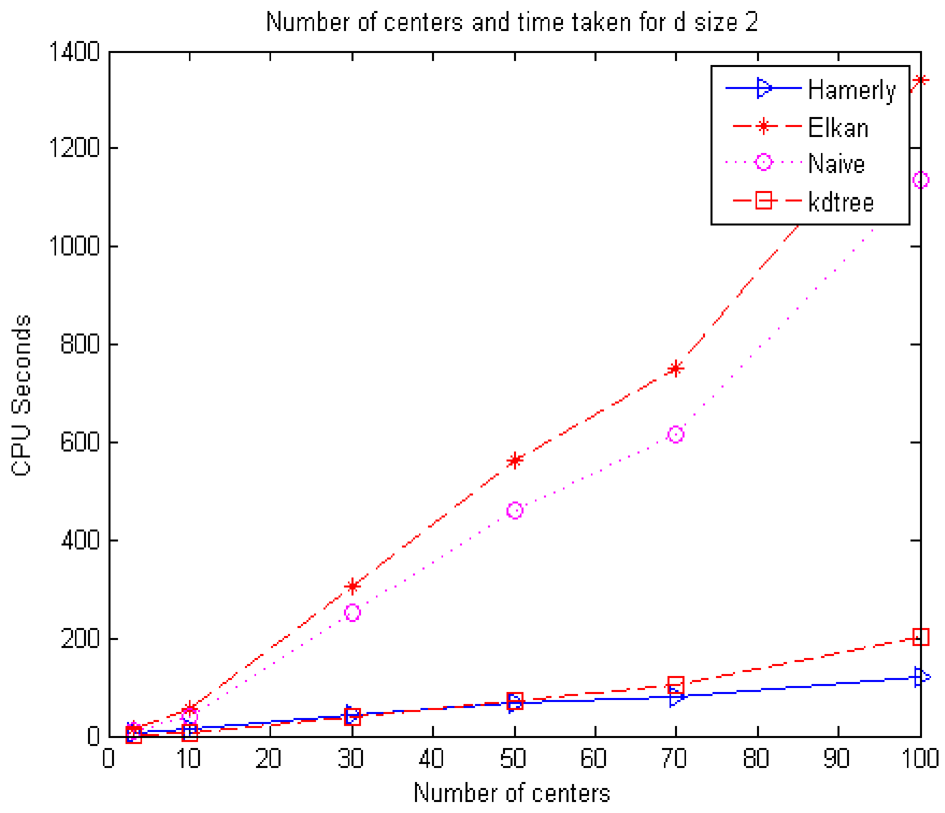


Figure 15: Effect of number of clusters for uniform random data with 2 dimensions

The result obtained in Figure 16 is for uniform random dataset with dimension 8 and 100000 data points. As we could observe in Figure 16, as the number of clusters increases *kd*-tree performs poorly. Naïve *k*-means performs better than *kd*-tree but still poor compared to Elkan's and Hamerly's algorithm. Elkan's algorithm performs better compared to naïve *k*-means and *kd*-tree. Hamerly performs the best compared to Elkan's algorithm, naïve *k*-means and *kd*-tree.

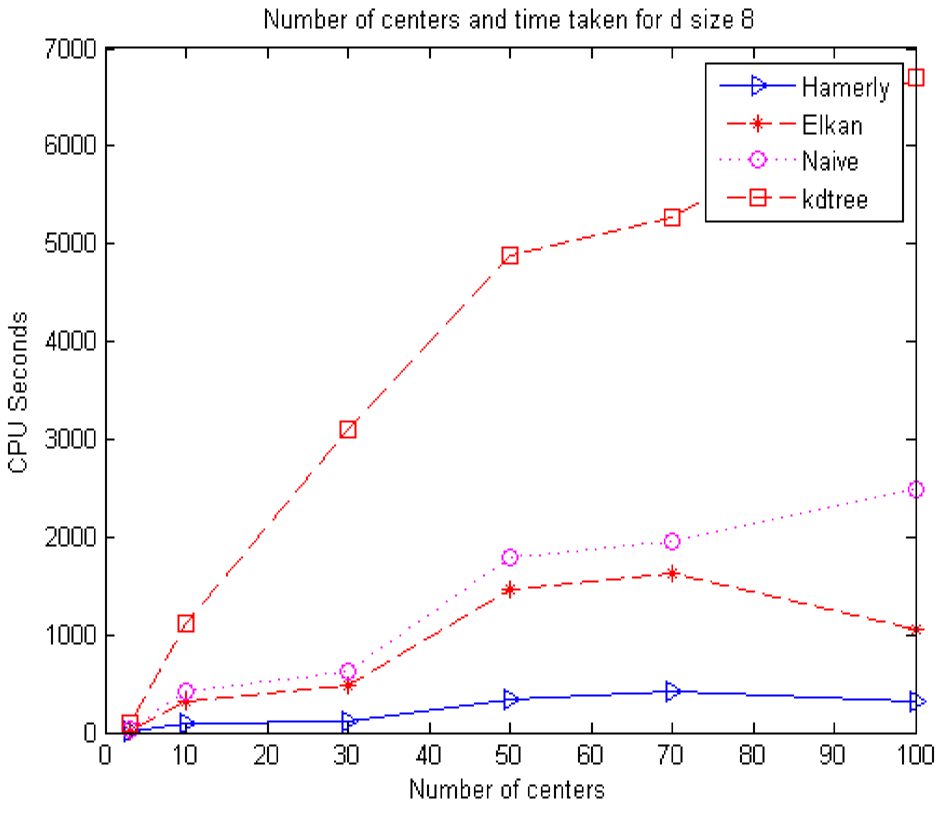


Figure 16: Effect of number of clusters for uniform random data with 8 dimensions

The result obtained in Figure 17 is for uniform random dataset with dimension 32 and 100000 data points. As we could observe in Figure 17, naïve *k*-means performs the worst with highest time consumption with to the increase in the number of clusters and dimensions. In case of Elkan's algorithm, as the number of clusters increase the performance becomes better which is evident when *k* value is greater than 70. This is because as the number of centers increase, the data points get clustered to its closest center. So, the data points do not move a lot in comparison to their previous *k* value which are less than 70. This avoids unnecessary distance calculations and reduction in time consumption. In case of Hamerly's algorithm, like Elkan's algorithm the time consumption decreases as number of clusters increases but it performs better than Elkan's algorithm because the Hamerly's algorithm performs better for low dimensional data than Elkan's algorithm.

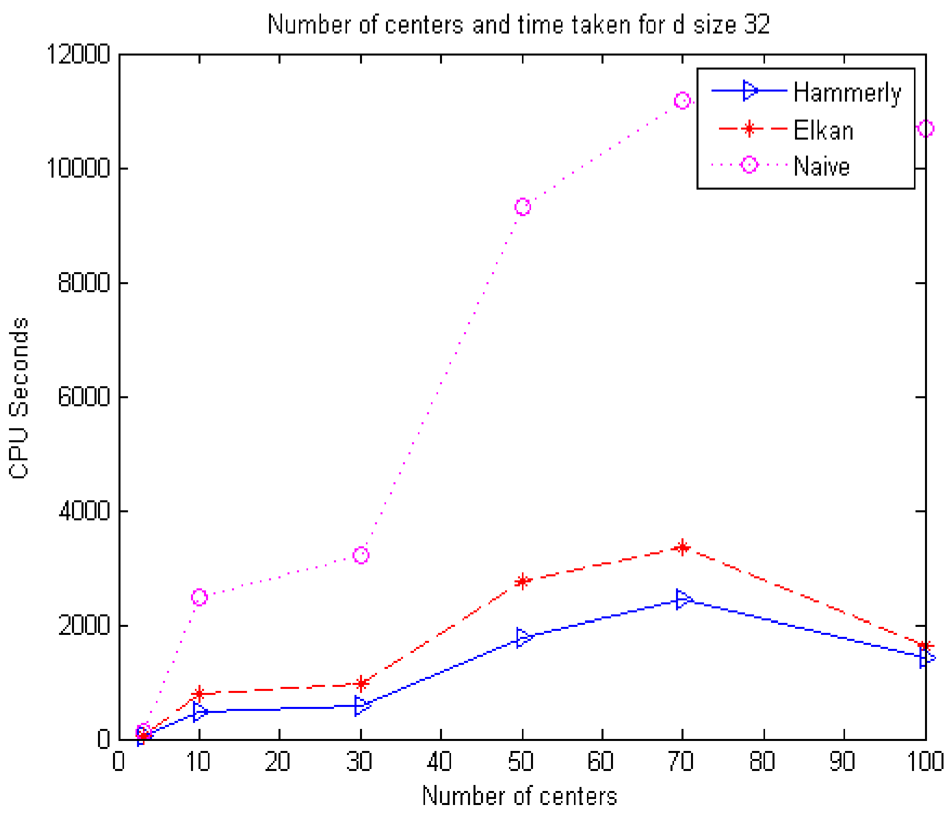


Figure 17: Effect of number of clusters for uniform random data with 32 dimensions

Figure 18 shows the result obtained for the uniform random dataset with dimension 64. From the results, we observe that naïve *k*-means performs poorly compared to Elkan's algorithm and Hamerly's algorithm. Hamerly's algorithm and Elkan's algorithm performs similarly until the number of clusters *k* is around 30. But as *k* increases we observe that Elkan's algorithm out performs Hamerly's algorithm by consuming less time. When *k* is 70 for both Hamerly's algorithm and Elkan's algorithm time consumption starts to reach saturation.

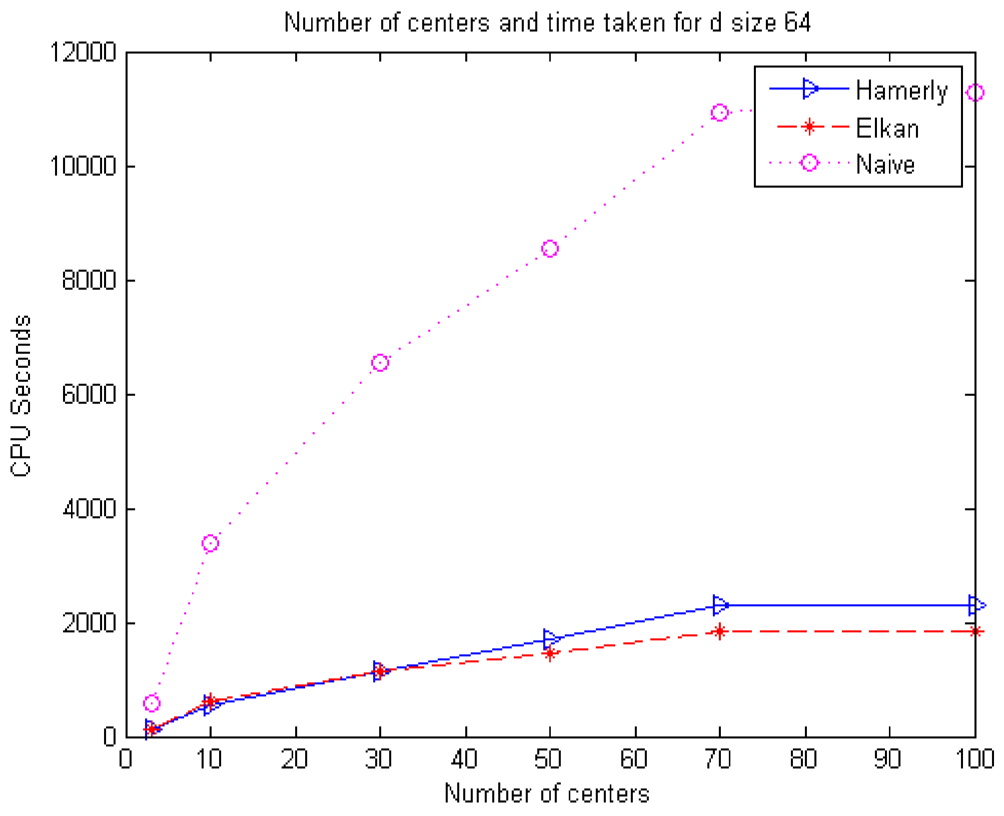


Figure 18: Effect of number of clusters for uniform random data with 64 dimensions

The result obtained in Figure 19 is for KDD Cup dataset with dimension 56 and 95413 data points. As we could observe in Figure 19, *kd*-tree performs the worst with highest time consumption with to the increase in the number of clusters. Naïve *k*-means performs better than *kd*-tree but still poorer than Elkan's and Hamerly's algorithm. In case of Hamerly's algorithm, it performs best when *k* is small, and as the *k* increases, it is comparable to Elkan's algorithm. In case of Elkan's algorithm, it performs the best of all the other three algorithms. This is due to the fact that Elkan's performs best for those data which are structured and are with more number of clusters.

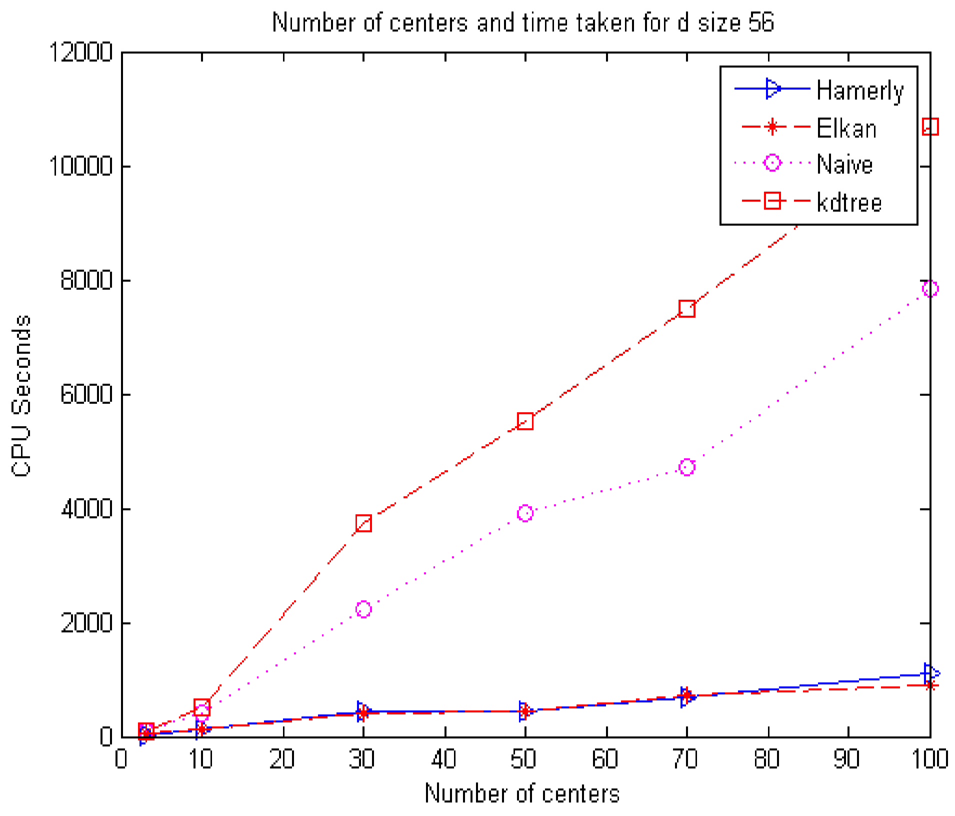


Figure 19: Effect of number of clusters in KDD Cup dataset

In Figure 20, effect of increasing the number of clusters for the algorithms on Wine dataset is illustrated. In case of *kd*-tree, the time consumption is the highest. Naïve *k*-means consumes more time as the *k* value increases. Elkan's and Hamerly's algorithm seems to have very small difference in their time consumption. Comparatively Hamerly's algorithm does best to Elkan's algorithm and naïve *k*-means. Elkan's algorithm performs similar to naïve *k*-means but at the end it performs better than naïve *k*-means with a very small difference.

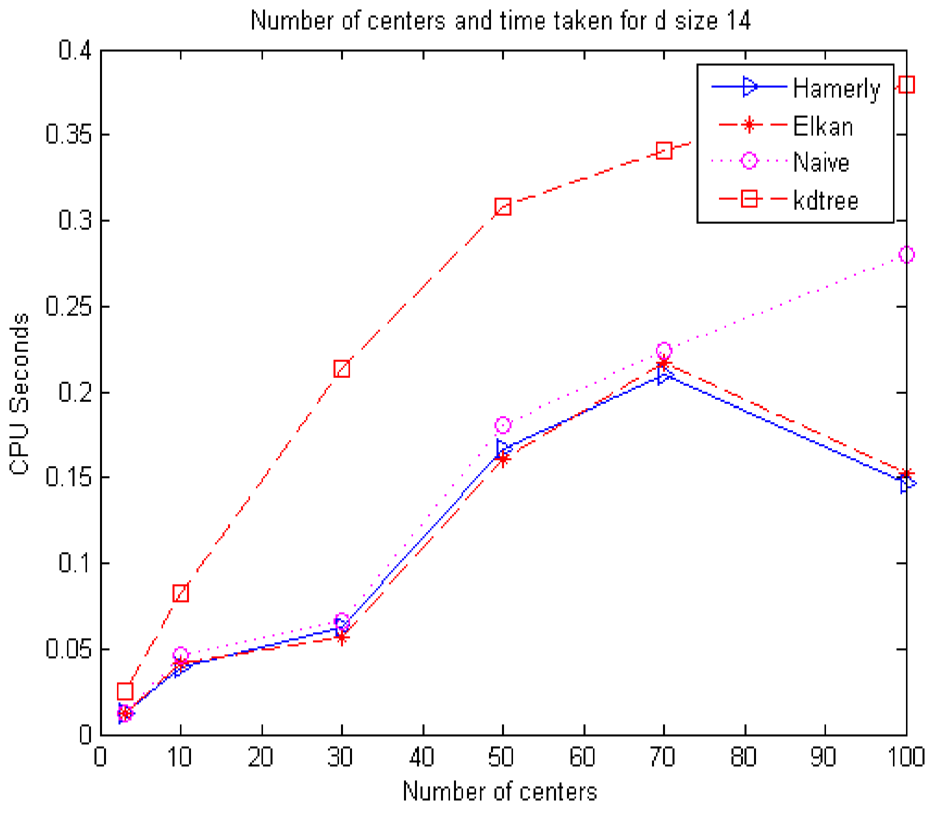


Figure 20: Effect of number of clusters in Wine dataset

4.3. Discussion on Experiment Results

From the experimental results obtained, we observe that Hamerly's algorithm is faster than naïve *k*-means. As a matter of fact, Hamerly's algorithm is the fastest of all the algorithms in the experiment, conducted for 8 dimensional and 32 dimensional uniform datasets. For 2-dimensional uniform dataset Hamerly's algorithm is fast but *kd*-tree performs better and it is the fastest for this case. The *kd*-tree's performance starts to deteriorate fast as the dimensions increase. For KDD Cup dataset, Hamerly's algorithm outperforms all the other algorithm when the *k* value is small. As the value of *k* increases, the performance of Hamerly's algorithm is comparable to the Elkan's algorithm. In summary, Hamerly's algorithm performs better compared to Elkan's algorithm as long as the dimension is less than or equal to 32. It can be seen clearly that Hamerly's algorithm suits the best for low dimensional datasets.

Observation from the results on Elkan's algorithm suggests that, it is efficient for data with increasing number of clusters *k*. This is because as the number of clusters increase there is decrease in the movement of data points every time and hence the number of distance calculations decreases. Since redundant distance calculations are reduced, the algorithm becomes faster. Another striking observation is that the algorithm performs better for datasets with higher dimensions. From the results obtained and shown in Table 2, Elkan's algorithm performs better for high dimensional data such as the KDD Cup and uniform data with dimension 64. The performance clearly improves with the number of clusters. The performance of Elkan's algorithm for two dimensional uniform dataset is very inefficient. Elkan's algorithm and Hamerly's algorithm complement each other as each of them are efficient for data sets with different dimensions.

In case of *kd*-tree, it works at its best for the datasets within a particular set of dimension value. As per our results it works best only for 2 dimensional dataset. As the dimensions of data increases, its performance decreases. It can be easily observed that *kd*-tree is even slower than naïve *k*-means for most of the cases.

5. Conclusion

In this thesis we have studied the improvements to *k*-means clustering. *k*-means clustering algorithm is a simple algorithm but, it is time consuming . The *k*- means algorithm suffers from problem of converging to local minimum. There have been various algorithms which aims to improve the *k*-means algorithm and to work around the limitations of the *k*-means algorithm. Some of such algorithms are discussed in this thesis. The focus of the thesis is mainly on the algorithms which decrease the time consumption of the *k*-means algorithm. Apart from this, the initialization of the cluster centers also plays an important role in the convergence of the algorithm. Incorrect initialization leads to incorrect results. Earlier researches have been directed at solving this issue and a huge amount of results are available in the literatures. Some of the results and methods are discussed in this thesis. All the algorithms, which have been compared in our study, are sensitive to initialization of the centers. Usually, in the *k*- means algorithm is run with different initialization and the one with the minimum mean squared distance is selected.

It is also seen that as the datasets size increases, naïve *k*-means and *kd*-tree based algorithm fail to scale and deliver the same performance and efficiency as it did for the lower size datasets. The simulations and the experiments are performed on naïve *k*-means and other accelerating algorithms of *k*-means proposed by Pelleg and Moore, Elkan and Hamerly. The improvements made to the *k*-means by Elkan and Hamerly reduces the number of required distance calculations using the triangle inequality. The algorithm by Pelleg and Moore uses geometric reasoning and this approach is also studied in this thesis. The time complexities of those algorithms are of primary concern in this thesis. It is seen that Elkan's algorithm is better for high dimensional data. Hamerly's algorithm performs well for low dimensional data.

The open questions that need further investigation from this thesis point of view are, can there be a bound for the minimum number of distance calculation for *k*-means clustering. There are also the questions of how to perform better clustering and how to find the local optima. The other things that can be investigated is, are there any other inequalities by which the clustering process speed-up.

6. References

[1] S. Arora, P. Raghavan and S. Rao, "Approximation schemes for Euclidean *k*-median and related problems," in *Thirtieth Annual ACM symposium on Theory of Computing,* Dallas, 1998.

[2] T. Kanungo, D. Mount, S. Netanyahu N, C. Piatko, R. Silverman and A. Wu, "A Local Search Approximation Algorithm for *k*-Means Clustering," in *Elsevier special Issue on the 18th Annual Symposium on Computational Geometry,* 2004.

[3] D. Pelleg and A. Moore, "X-means: Extending *k*-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning,* Palo Alto, CA, July 2000.

[4] D. Pelleg and A. Moore, "Accelerating exact *k*-means algorithms with geometric reasoning (Technical report CMU-CS-00105)," Carnegie Mellon University, Pittsburgh,PA.

[5] S. I Har-Peled and B. Sadri, "How fast is the *k*-means Method," in *ACM-SIAM Symposium on Discrete Algorithms,* Vancouver, 2005.

[6] T. Kanungo, D. Mount, S. Netanyahu N, C. Piatko, R. Silverman and A. Wu, "An Efficient *k*-means clustering algorithm: analysis and implementation," *In IEEE Transactions On Pattern Analysis And Machine Intelligence,* no. 7, pp. 881-892, July 2002.

[7] G. Hamerly, "Making *k*-means even faster," in *proceedings of the 2010 SIAM international conference on data mining (SDM 2010),* 2010.

[8] G. Hamerly and C. Elkan, "Learning the *k* in *k*-Means," in *Neural Information Processing Systems,* MIT Press, 2003.

[9] C. Elkan, "Using the Triangle Inequality to Accelerate *k*-means," in *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003),* Washington D.C, 2003.

[10] K. Anil, "Data Clustering: 50 years beyond *k*-means," *Journal on Pattern recognition Letters,* vol. 31 , no. 8, pp. 651-666, June,2010.

[11] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *proceedings of fifth Berkeley Symposium on Mathematics, Statistics and Probaility,* Berkeley,CA, 1967.

[12] D. Arthur and S. Vassilvitskii, "*k*-means++: The Advantages of Careful Seeding," in *Society for Industrial and Applied Mathematics* , Philadelphia, 2007.

[13] J. Kosecka, "EM Algorithm Statistical Interpretation," George Mason University, Virginia, 2005.

[14] L. Kaufman and P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, NewJersey: Weily, 1990.

[15] J. C. Bezdek, Pattern Recognition with fuzzy objective function algorithms, Newyork: Plenum, 1981.

[16] M. Matteucci, "A tutorial on clustering algorithms," [Online]. Available: http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html. [Accessed 24 10 2013].

[17] B. Horst, A. Leonardis and A. Selb, "MDL principle for robust vector quantization," *Pattern analysis and applications,* vol. 2, pp. 59-72, 1999.

[18] M. Steinbach, G. Karypis and V. Kumar, "A Comparison of document Clustering methods," in *KDD conference,* 2000.

[19] A.Moore and M.S.Lee, "Cached Sufficient Statistics for Efficient Machine Learning with large datasets," *Journal of Artificial Intelligence Research,* vol. 8, pp. 67-91, 1998.