KTH Computer Science
and Communication

Efficient K-means clustering and the importance of seeding

PHILIP ELIASSON, NIKLAS ROSÉN

Bachelor of Science Thesis
Supervisor: Per Austrin
Examiner: Mårten Björkman

Abstract

Data clustering is the process of grouping data elements based on some aspect of similarity between the elements in the group. Clustering has many applications such as data compression, data mining, pattern recognition and machine learning and there are many different clustering methods. This paper examines the k-means method of clustering and how the choice of initial seeding affects the result. Lloyd's algorithm is used as a base line and it is compared to an improved algorithm utilizing kd-trees. Two different methods of seeding are compared, random seeding and partial clustering seeding.

Referat

Effektiv K-means klustring och vikten av startvärden

Klustring av data innebär att man grupperar dataelement baserat på någon typ a likhet mellan de grupperade elementen. Klustring har många olika användningsråden såsom datakompression, datautvinning, mönsterigenkänning, och maskininlärning och det finns många olika klustringsmetoder. Den här uppsatsen undersöker klustringsmetoden k-means och hur valet av startvärden för metoden påverkar resultatet. Lloyds algorithm används som utgångspunkt och den jämförs med en förbättrad algorithm som använder sig av kd-träd. Två olika metoder att välja startvärden jämförs, slumpmässigt val av startvärde och delklustring.

Contents

Chapter 1

Introduction

Data clustering is the process of grouping data elements in a way that makes the elements in a given group similar to each other in some aspect. Clustering has many applications such as data mining, statistical data analysis and bioinformatics[1]. It is also used for classifying large amount of data, which in turn is useful when analyzing data generated from search engine queries, articles and texts, images etc.

Clustering has no exact definition and there are several ideas of how a cluster can be defined. Each idea result in different types of algorithms that each is useful for different types of data. An example of such an idea is *centroid based* clustering. The main idea of centroid based clustering is that each data points belongs to the cluster which center is within closest *distance* of that data point. Again, how distance is measured varies depending on application and clustering algorithm. For each clustering method the resulting clusters should match the intuitive idea of how the elements are partitioned. This definition is far from precise as it depends on ones intuition, and it does not apply to data of higher dimensions as it is extremely hard, if not impossible, to grasp the natural partitioning of data with more than three dimensions.

1.1 Problem Statement

The k-means method is a popular[2] approach to clustering, the method is simple and heuristics allow for relatively efficient implementation that still produce good results. Lloyd's algorithm is a well known and very simple heuristic for performing k-means clustering but it suffers from performance problems. The speed and results of Lloyd type algorithms depend on the initial positions of cluster centers and the algorithm used to update the centers. This paper will use Lloyds algorithm as a baseline and examine how the update algorithm can be improved using kd-trees as well as the impact the initial cluster centers have on results and running speed.

1.2 Document Overview

In chapter 2 the k-means method will be described in detail and in sections 2.1 and 2.2 the algorithms considered in this paper are explained and briefly discussed. In section 2.4 two methods of choosing seeds will be examined. Chapter 3 contains the practical experiments performed and their results. In section 3.1 the two clustering algorithms are compared and in section 3.2 two methods of choosing initial seeds are tested against each other. Lastly in chapter 4 the conclusions of this paper are presented.

Chapter 2

The k-means method

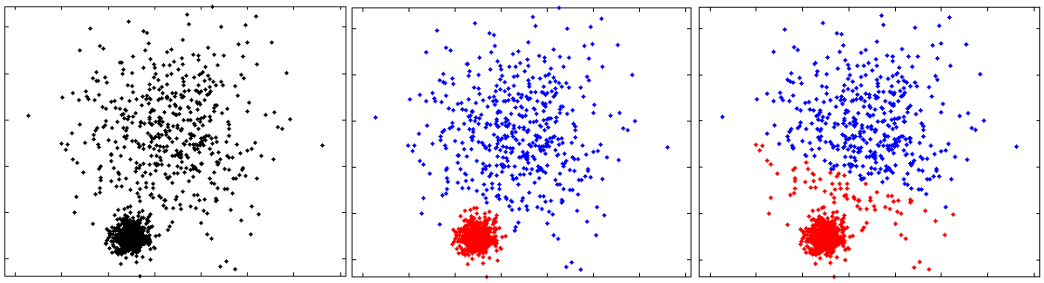
The clustering method known as k-means is a method that describes the "best" partitioning of a data set containing k number of clusters. The method is defined by its objective function which aims to minimize the sum of all squared distances within a cluster, for all clusters. The objective function is defined as:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \left(\sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2 \right)$$

where $\mathbf{x}_j$ is a data point in the dataset, S_i is a cluster (set of data points) and $\boldsymbol{\mu}_i$ is the cluster mean (the center point of cluster S_i).

One aspect of k-means that makes it different from many other clustering methods is that the number of clusters is fixed when clustering occurs. This can be considered as both a weakness and a strength. One positive property of a fixed number of clusters is that the k-means method does not introduce new clusters in case of an anomaly data point, instead it sorts the anomaly data point to its closest cluster. The drawback of using a fixed number of clusters is that it might not be clear how many clusters a dataset might contain. Using an unsuitable k may cause the k-means method produce poor results, possibly to the point of becoming unusable.

As with any clustering method k-means is not suitable for all types of data. Even when individual clusters have suitable properties for k-means clustering the density and position of the cluster can cause it to produce counterintuitive results.



Raw input data | Intuitive clustering | K-means clustering

Figure 2.1: K-means clustering unfit to cluster irregular data clusters

Lastly the computational difficulty of finding a cluster configuration that minimize the objective function has been proven to be NP-hard[1] with a worst case time complexity of $2^{\Omega(n)}$[3] where n is the number of data points. Therefore heuristic algorithms are used in order to minimize the k-means function.

2.1 Lloyd's algorithm

The first heuristic algorithm to perform k-means clustering was proposed in 1957 and is known as Lloyd's algorithm [1]. This algorithm is often referred to as "the k-means algorithm" which might cause confusion, as it does not define k-means clustering.

The algorithm works by iteratively improving the position of the cluster centroids. How the initial centroid positions, known as *seeds*, are decided is not part of the algorithm and is something that has to be provided as an argument in order for the algorithm to operate. How seeds can be estimated is discussed in section 2.4.

The algorithm improves centroid positions by iterating the following two steps:

$$\textbf{step one:} \qquad S_i = \{\mathbf{x}_j : \|\mathbf{x}_j - \boldsymbol{\mu}_i\| \leq \|\mathbf{x}_j - \boldsymbol{\mu}_c\| \; \forall \; 1 \leq c \leq k\}$$

$$\textbf{step two:} \qquad \boldsymbol{\mu}_i = \frac{1}{|S_i|} \left(\sum_{\mathbf{x}_j \in S_i} \mathbf{x}_j \right)$$

In **step one** all data points, $\mathbf{x}_1 ... \mathbf{x}_n$, are assigned to one of its closest centroid, $\boldsymbol{\mu}_i$. The process of finding a closest neighbors is called nearest neighbor search.
In **step two** all centroids, $\boldsymbol{\mu}_1 ... \boldsymbol{\mu}_k$, are updated by calculating the mean of all data points in the cluster.

These steps are performed until some type of criteria is met, usually that there were no change in the cluster configuration or that the change in the target function was below a certain threshold.

Note that Lloyd's algorithm does not specify how distance is measured. The most common measure of distance is the euclidian distance, although other distance functions is used when clustering certain kind of data. An example is the Jaccard distance used for natural language processing[4].

The time complexity of one iteration of Lloyd's algorithm is $O(ndk)$[5] where n is the number of data points, d the dimension of the data and k the number of cluster centroids.

2.2 Kd-tree

One approach to improve Lloyd's algorithm examined by Pelleg and Moore[5] is to reduce the times it take to assign each data element to its closest cluster. This is done with a space partitioning data structure called a kd-tree. Simply put a kd-tree is a binary tree where every node contains a hyperplane defined by a dimension and a point. This plane splits space into two, represented by the two children of the node. The data points are stored in the leaves of the tree. The splitting dimension is usually chosen according to a fixed sequence but may also be randomized. The kd-tree algorithms discussed below are both computationally the same as Lloyds and will generate the same cluster configuration for the same input data.

2.2.1 Kd-center

A kd-tree can be used in several different ways in order to improve performance. In Lloyd's algorithm a nearest neighbor search is performed on every data point with respect to every cluster centroid to determine which cluster the data point should be assigned to. This takes nk time, but with the help of a kd-tree it is possible to reduce this factor by building a kd-tree containing all cluster centroids. Since it takes $dk \log^2(k)$ time to build a kd-tree and $\log(k)$ time to find the nearest neighbor in a kd-tree containing k elements, the time it takes to assign all data points has now been reduced to $dk \log^2(k) + n \log(k)$.

2.2.2 Blacklisting algorithm

An alternative way to use a kd-tree is to reduce the number of nearest neighbor searches performed. This kd-tree optimization is based on the idea that several data points often share a common nearest cluster center. Instead of building a kd-tree containing cluster centroids the data points are used instead. When a nearest neighbor search is performed the kd-tree is traversed until a subspace with only

one closest centroid is found. All data points in this subspace can then be assigned to that cluster center. If a leaf is reached then all data points within that leaf is assigned to its closest cluster similar to Lloyd's algorithm, the difference being that each data point only has to be compared to the closest centroids for that leaf.

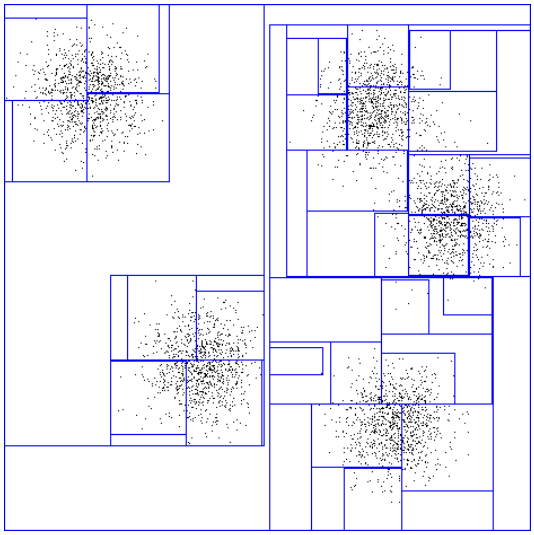


Figure 2.2: Subspaces in a kd-tree, partitioning the data points

When a kd-tree is used in this fashion it is also possible to improve step two of Lloyd's algorithm. Since the data in the kd-tree does not change between iterations the kd-tree need to be built only once. This enables certain data to be calculated and stored in the nodes of the kd-tree, which can be used to improve the speed of calculating new cluster means. In Lloyd's algorithm the time it takes to update $\boldsymbol{\mu}_i$ is linear with $|S_i|$. By storing the number of data points and mean of all data points in the subspace for every node in the kd-tree the calculation will then become linear with the number of subspaces assigned to the cluster instead of the number of data points.

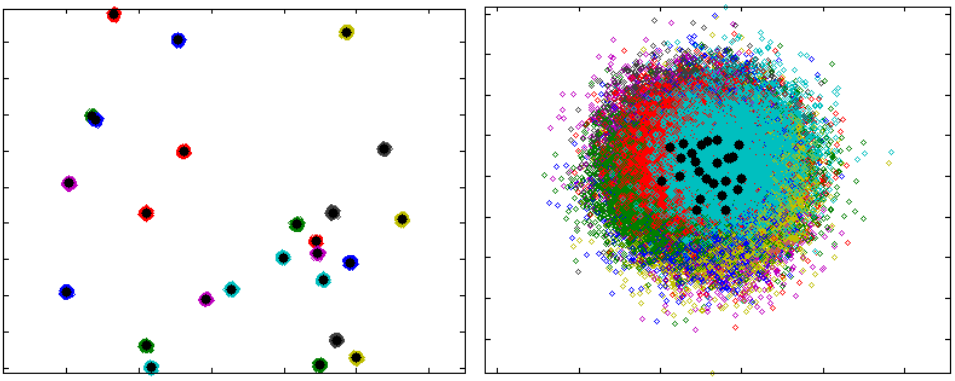
Note the algorithm explaind in this section is a slightly simplified version of the true blacklisting algorithm detailed in [5].

2.3 Weaknesses of Lloyd's and the kd-tree algorithms

As both Lloyd's algorithm and the kd-tree algorithms are heuristic algorithms there are no guaranties of how well they will perform. This in regard of both clustering results and speed.

If for example "bad" data were to be clustered using k-means it can greatly af-

fect running time of the algorithms. An example of "bad" data are clusters that are in close proximity of each other. This will cause the cluster configurations to converge very slowly, causing many iteration to occur.



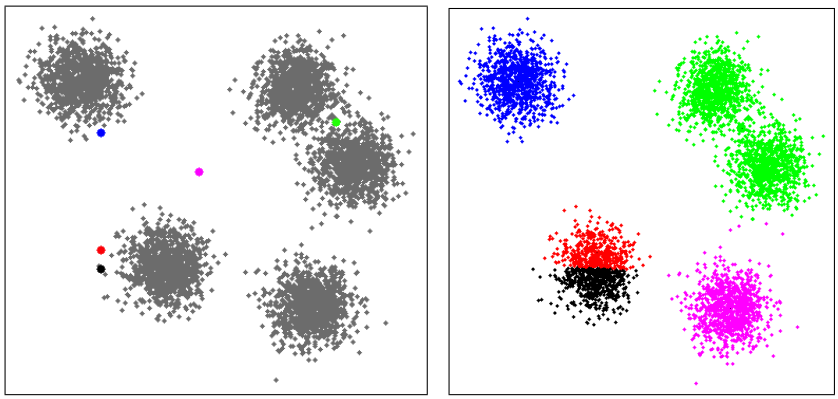
Data with clear structure
cpu time: 21.5s
iterations: 5

Data with no clear structure
cpu time: 149.3s
iterations: 70

Figure 2.3: Differance in running time using Lloyd's algorithm on data sets where $d = 2$, $k = 25$ and $n = 100000$.

The "bad" data used in figure 2.3 is of course not appropriate nor meaningful to cluster using k-means, and is only used for explanatory purposes. There is however meaningful data that impact the kd-trees algorithms especially. Data of higher dimensions reduce the effectiveness of the kd-tree structure since it has to traverse through many nodes when performing a nearest neighbor search. This can cause the overhead to negate the positive properties of the kd-tree.

Another problem regarding clustering results is that clusters can converge to their local minimum instead of the global minimum. This is caused by unlucky or poorly chosen seeds.



Initial cluster centroids (seeds) Resulting k-means clustering

Figure 2.4: Seeds causing k-means clustering to converge to local minimum.

2.4 Seeds

As descibed in section 2.1 the k-means algorithms need seeds in order operate. There are several methods to find seeds, but only two will be mentioned here.

2.4.1 Random

Random seeding is the most simple form of seeding. k number of data points are selected randomly from the data set and those points are used as initial cluster positions. This might seem as a bad strategy but it works due to the fact that most data points are in the proximity of a cluster center, given that the data set is appropriate for k-means clustering. When using random seeds the common practice is to perform clustering several times with new random seeds each time in order to minimize the risk of poor results[2].

2.4.2 Partial clustering

The basic idea of partial clustering seeding is to perform clustering on a subset of the data and use the resulting cluster centroids as seeds when clustering the complete data set. This method is potentially more powerful than using random seeds but the time needed to cluster the subset must be taken into account. Using partial clustering requires a tradeoff between cluster quality and clustering time.

Chapter 3

Experiments

In this section the blacklisting kd-tree algorithm is compared to Lloyd's algorithm in respect to running speed in order to substantiate the theory discussed earlier. The effects of seeding will be examined with respect to the result and running time of Lloyd's algorithm. The data used for the clustering experiments is generated to be normally distributed around the cluster centers. The positions of the centers are randomized with a uniform distribution and scaled to produce well separated clusters. All data generated is made to be well suited for k-means clustering, as we were not interested in examining worst case scenarios.

3.1 Running time of Lloyd's and K-means

3.1.1 Dimensions

The purpose of the following test is to evaluate how well the different algorithms perform as the number of dimensions increases. In order to keep the tests meaningful the dimensions span from 2 to 10, as the kd-tree algorithm is know to perform bad at higher dimensions.

Test data generated for this test has 20000 data points and for every combination of n, k and d, 3 instances was generated resulting in a total of 60 data sets. Lastly, the average for each combination of d and k were calculated in order to produce the final result.

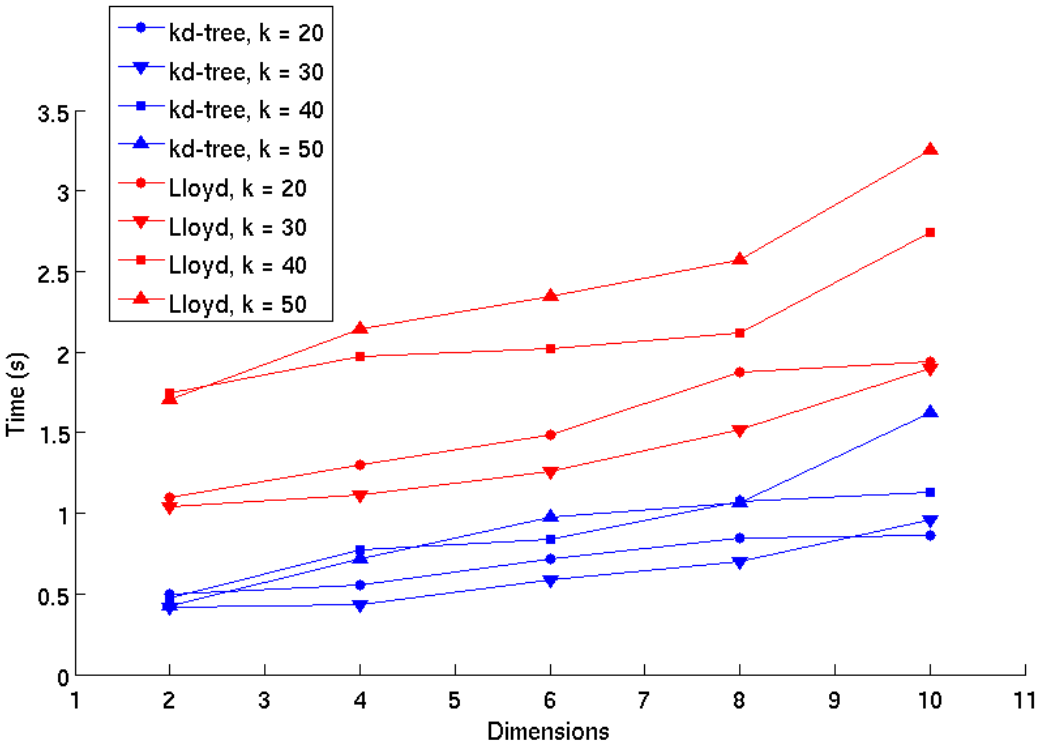


Figure 3.1: CPU time for clustering data sets of varying number of centroids and dimensions containing 20000 data points.

As seen in figure 3.1 the kd-tree algorithm performs very well compared to Lloyd's algorithm. Lloyd's algorithm scales as expected and performs at best 98% worse than the kd-tree ($k = 20$, $d = 6$). It also shows how well the kd-tree scales as the number of clusters increase.

From this test it is not clear how the kd-tree algoritm scales in regard to dimensions. For the data sets where $k = 20$ and $k = 50$ it looks as if the kd-tree algorithm might scale sub-linear with dimensions. In order to clarify a more thorough test is performed.

For the following test a total of 50 data sets is used. For every dimension 10 data sets were generated. For this test $k = 20$ and $n = 20000$. The final result is the average of all 50 clusterings performed.

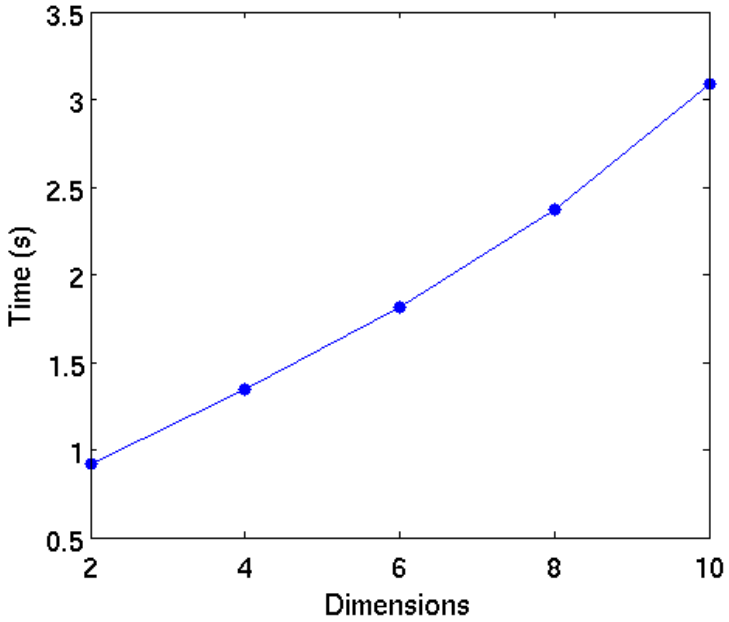


Figure 3.2: CPU time for clustering data sets of varying dimensions using the kd-tree algorithm.

In figure 3.2 it becomes clear that the kd-tree algorithm scales super-linear with the number of dimensions.

3.1.2 Data points

The purpose of the following test is to see how well Lloyd's algorithm and the kd-tree algorithm scales as the number of data points increase. A total of 45 data sets is used, 3 data sets for every combination of n and k. The dimension of the data is 3. The results was averaged in order to produce the final results.

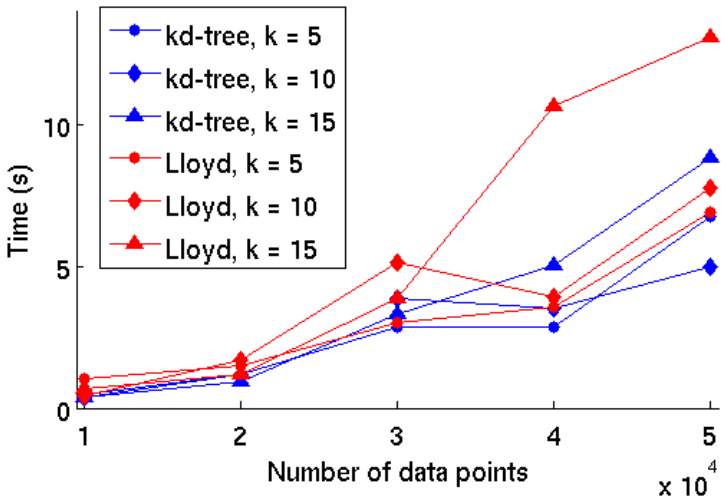


Figure 3.3: CPU time spent clustering data sets containing a varying amount of data points and clusters of dimension 3.

The results displayed in figure 3.3 show that Lloyd's algorithm can compete with more advanced algorithms such as kd-tree for data sets of smaller size. The difference in time becomes more noteable as the the number of data points increase. The reason for Lloyds algorithm performing so well is most likely due to its simplicity causing only a minor overhead. Note that Lloyd's never performed better than the kd-tree for any given k.

3.1.3 Cluster centers

The following experiments are performed in order to evaluate how the kd-tree algorithm and Lloyd's algorithm perform as the number of centroids increase. The data used contains 20000 data points of dimension 3.

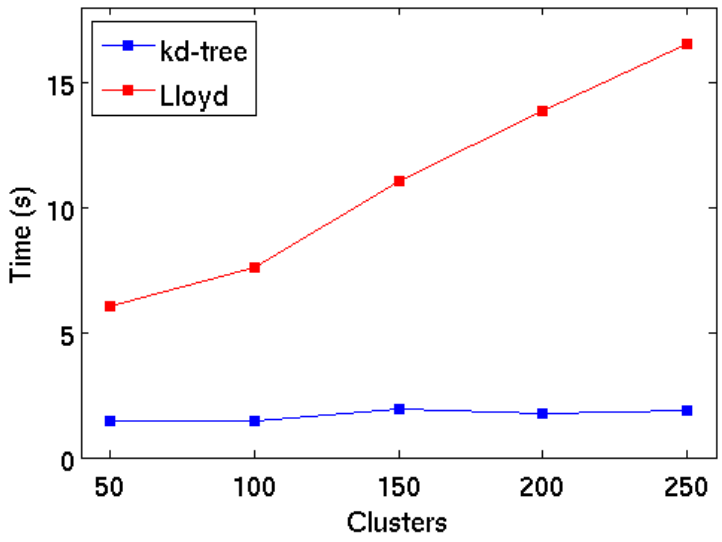


Figure 3.4: CPU time when clustering data sets with varying cluster count

The results displayed in figure 3.4 shows how well the kd-tree algorithm scales as the number of clusters increase. Even though it was expected for the kd-tree to severely outperform Lloyd's algorithm in this kind of test the results was still better than expected. The difference in time between 50 and 250 centroids is only 0.29 seconds for the kd-tree algorithm. For Lloyd's algorithm the difference is 10.22 seconds.

The reason the kd-tree algorithm perform so well for this test is most likely due to the fact that as the number of clusters increased, the number of data points per cluster decreased. This results in the kd-tree structure having to create an equal amount of subspaces in order to cover all data points regardless of how many clusters is used. This in turn makes the kd-tree equally deep and equally fast to search in.

3.2 Effects of seeding

In order to examine the effects of seeding, Lloyds algorithm is used to cluster the same data with different initial seeds. The quality of the clusters and the running time of the clusterings are then compared between the two ways of seeding. The data generated for this experiment has 10000 data points, 20 clusters and 3 dimensions. A total of 100 data sets is used.

In order to perform partial clustering $\sqrt{n} = \sqrt{10000} = 100$ points are chosen at random and clustered. These clusters will then be used as seeds when clustering the full data set.

For the random seed simply $k = 20$ data points are selected at random and these will become the seed when clustering the full data set.

In table 3.1 the parameter q_{avg} denotes the average clustering quality. This is a unitless scale where lower results equal better quality. The parameter i_{avg} denotes the average number of iterations performed when clustering. The parameter t_{avg} denotes the average running time. Note that for partial clustering the time used to perform the partial clustering is **not** included due to limitations in the software used to perform clustering. Only the time used to cluster the full data set is available. The column "Best time" describes how many times the seeding method resulting in faster running time and "Best quality" describes how many times the seeding method resulted in better quality.

	t_{avg}	q_{avg}	i_{avg}	Best time	Best qualtiy
Pre clustered	1.6055	$3.8533 * 10^5$	22.4000	65	67
Random	1.8370	$5.0430 * 10^5$	24.9300	34	33

Table 3.1: Results of the random seeding method and the partial clustering seeding method.

Looking at the results in table 3.1 shows the difference in performance regarding both time and quality. Unfortunately the true cost of the partial clustering method is still unknown as the cost of finding the seed is not included. However, it is clear that partial clustering can notable affect both quality and running time when clustering a complete data set, showing us that carefully chosen seeds can improve algorithms such as Lloyd's and kd-tree.

Chapter 4

Conclusions

The problem statement asks how Lloyd's algorithm can be improved in order to deal with its performance issues. Analyzing the results of our experiments we can conclude that there are several ways of improving Lloyd's algorithm. Although the kd-tree structure provided a significant speed increase for certain types of data it does not always perform better in terms of speed. The kd-tree algorithm still perform bad when used with high dimensional data [5], which is a common type of data in certain applications. We have also shown that carefully chosen seeds can improve both speed and quality of Lloyd's algorithm.

In this paper we have examined two different ways of seeding and two different ways of iteratively improving cluster centroids. There is of course others and more complex methods that can be examined in order to further improve the task of k-means clustering.

Bibliography

[1] Anil K. Jain, *Data Clustering: 50 Years Beyond K-Means*. Michigan State University, Michigan

[2] P.S Bradley, Usama M. Fayyad, *Refining Initial Points for K-Means Clustering*. University of Wisconsin, Wisconsin

[3] Andrea Vattani, *k-means Requires Exponentially Many Iterations Even in the Plane*. University of California, San Diego

[4] Sophia Katrenko, Pieter Adriaans, *A Local Alignment Kernel in the Context of NLP*. University of Amsterdam, Netherlands

[5] Dan Pelleg, Andrew Moore, *Accelerating Exact k-means Algorithms with Geometric Reasoning*. Carnegie Mellon University, Pittsburgh