

Comparison of Clustering Algorithms for Searching in High-Dimensional Data

Daniel Dunning

Yu Zhuang

Abstract—Searching data is an area of research that is always open to improvement. The problem of searching one-dimensional data has already been solved, but with the increase of Big Data, as well as an increase in the need for computational sciences, higher-dimensional data has been on the rise. It is known that searching sorted data is exceptionally faster than unsorted, but the problem has become finding efficient methods of sorting data which has more than one dimension. The purpose of this paper is not to create a new method for sorting high-dimensional data, rather, comparing previous methods and comparing their efficiency. In addition, we will investigate the searching methods performed on each the clustered data, and study whether this should have any impact on the use of the clustering algorithms.

Index Terms—Clustering, High-dimension data, Searching, Efficiency

I. INTRODUCTION

THERE has constantly been a need for searching data in all fields of research. This task can be either searching for a specific point in the data set, or searching for a point which is closest to a given query point. No scientific or technological research area can avoid the need to compare new data with already gathered data, so the desire to do this efficiently is high. The techniques for actually searching data that is sorted are mostly known; the most famous example being a binary search of a list. Initially, the question must be asked, whether it is computationally worth it to sort data before it is searched. The search time of a brute force search, searching through all n elements of a data set, is $O(n)$. Going back to the example, the best case for search time for a binary search is: the time taken to sort the data (at best $O(n \log(n))$) plus the binary search time ($\log(n)$). This total time is more than a simple brute force search, so why bother with sorting the data? The reason lies in the fact that sorting allows us to search the same data set efficiently for multiple queries. If a particular data is only ever search once and then discarded, it would be a waste to perform a sorting technique to it before search, because it would take more time. However, this is rarely the case, and in most cases data sets need to be searched for thousands of queries, making sorting the efficient solution.

Sorting techniques have been researched extensively for one-dimensional data, and the concept of sorting, for example, a list of numbers is simple: the numbers are arranged in ascending (or descending) order. However, as we increase dimensions, the task of grouping the data in terms of *least* and *most* becomes harder. Even using a two-dimensional example, it is easy to say that on a coordinate plane a point with small x and small y values is "smaller" than a number with large x and large y values. However, how do we categorize when

x is small and y is large, or x is large and y is small? These distinctions are harder to make, so instead we *cluster* the data by how similar it is to those points around it. We can then perform groupings within the data, and can use these groups or clusters to search data points that are similar.

The remainder the the paper will follow as such. Section 2 will discuss the problem in more detail and give an explanation for each of the data sets used. Section 3 will give more detail to explain the clustering algorithms and the search procedure used within them. Section 4 will discuss the procedure of the research and the experiments that followed. Finally, Section 5 will discuss the results of the tests and Section 6 will discuss the contributions of this research and future research.

II. PROBLEM AND DATA SETS

The main goal of this research was to compare various methods of clustering high-dimensional data, and study the efficiency of them, both for their speed in clustering, but also in the speed of the searching for the closest data point in each cluster using the resulting clusters. In addition to having a dimensionality of at least two, the data sets were also expected to have a sufficiently large amount data points attributed to them. Four data sets were used for the tests. In the tests performed, up to 10,000 query points were randomly generated (with the corresponding dimensions based on their data set counterpart). The seed for these random query points was kept the same through all algorithms and their subsequent tests, so that the results were comparable. The first data set was randomly generated and not used for testing the efficiency of the algorithms, as it was relatively small, with only 10,000 data points of two dimensions, so there was little time distinction between the algorithms using this set. This data set was generated by having mostly random points, but with the addition of several points per query that were close in distance to each query point. This is done as mentioned in REFERENCE HERE because in most real-world scenarios there are a couple of points which lie very close the the query, and this data set was created to mimic those scenarios. In addition, the data set was used to check for algorithm accuracy compared to a brute force search, and also to show a two-dimensional representation of how the clustering algorithms worked. The second data set is a list a the longitude and latitude of all major cities in the world. The third set consists of the results from an experiment studying the chemical reactions between ethylene gas and carbon-dioxide gas in a chamber, with the results coming from several sensors over the course of many time steps. The last data set comes from several

studies of cancerous cells, which contains the gene expression of all possible genes taken from cancer cells. These data sets were chosen because of the unique characteristics each one brings; The world coordinates contain millions of data points with low dimensionality, the chemical reaction set has millions of points while also have fairly high dimensionality, and the cancer set has a small number of points with extremely high dimensionality. The data sets are listed in Table 1 with their respective attributes. INSERT TABLE HERE WITH THE 4 DATA SETS

III. ALGORITHMS

The clustering algorithms that were used for the research are KD-Tree, KMeans, and Locality Sensitive Hashing (LSH). These are three of the primary clustering algorithms used for data mining, and they all cluster the data using different techniques.

A. KD-Tree

KD-Tree is a recursive algorithm used to cluster data in rectangle-shaped groups. This is done by splitting the data set in half along the dimension with the highest variance. GIVE EQUATION FOR VARIANCE. This splitting is then performed on each subsequent group until the desired number of clusters is reached. As a matter of principle, it is efficient to use a number of clusters equal to a power of two. Over the course of the clustering, it is important to keep track of each cluster's minimum and maximum value for each dimension, as this will be store at the end as the boundaries of the cluster. This boundary is used for searching the clusters. For each query, a euclidian distance will be calculated between the closest point of the cluster and the query point, which will be 0 if the point lies within the boundaries of the cluster. At this point, all the data points in the closest cluster are exhaustively searched to find the minimum distance to the query point. At this point, it is important to check whether there are any clusters which are closer to the query point than the minimum distance found. This is because it is possible for a point to have its nearest neighbor lie in a cluster different than the one the query point falls into. For every cluster that is closer than the initially-found distance, we exhaustively check every point in those clusters, comparing to the minimum distance. If the particular application in fact only needed an approximate nearest neighbor, and not an exact nearest neighbor, the last step in the search can be excluded. One would only need to search for the closest data point in the closest cluster, and that would be sufficient. An example of the clustering done using KD-Tree is shown in Figure 1. INSERT FIGURE

B. KMeans

KMeans was one of the first algorithms used for clustering multi-dimensional data. The process involves initializing centroids equal to the number of clusters needed. Each data point is then assigned to the centroid it is closest with, and a new centroid is calculated using the average value in each dimension of the points assigned to it. This process continues,

assigning points to a centroid and calculating new centroids, until there is no change in centroid from one iteration to the next, or the assignment of data points does not change. The clusters that are formed take on a circular shape, and for this reason the cluster boundary uses the radius from each cluster centroid, which is simply the distance to the farthest point in each cluster. Searching can then be done by finding which cluster is the closest, and searching points in it. The closest distance will be the distance to the centroid, minus the cluster radius. If this value is negative, it means the query point falls inside of that cluster. As with KD-Tree, once the initial minimum distance to a point within a cluster is found, it is important to check points in other clusters where the distance to that cluster is less than the minimum distance to a point already found.

KMeans is a very powerful algorithm, and will of course always yield the correct results, however it is extremely computation-heavy. Each iteration requires each point finding the different lengths to each centroid, which is very time-consuming when there are a large number of centroids and the data has many dimensions. In addition, the number of iterations can be high, which leads to more computations. The amount of time that the algorithm takes is very dependent on the initial centroids. The simple way to generate them, apart from random points, is to choose a data point as the starting centroid. Next, choose the data point which is farthest away from that point as the next centroid. For the future centroids, obtain maximum distances to each centroid for each data point, and choose the minimum of those maximums. This technique, however, tends to not yield much better efficiency results than simply choosing random centroids. The technique used to calculate the initial centroids for this research is a variation of KMeans called Bisecting-KMeans. The initial centroid is chosen the same way, choosing a random data point. After the second point is chosen (the original way), however, the KMeans algorithm is then performed on those two points, assigning data to them, and calculating new centroids, with the same stopping conditions. Once the two centroids are "stable" i.e. unchanging, Then the next centroid is added, using the minimum maximum point method, and the three centroids and then put through the algorithm. It is obvious that the initialization of the centroids takes longer than the simple ways, because it must go through the KMeans algorithm at each step. However, hope of doing this is that each additional centroid will not take long to compute if the other ones are already "good" centroids, and the goal of this technique is that the actual KMeans algorithm will run more efficiently when the initial centroids start off close to their final spot. Figure 2 shows an example of clustering done by KMeans. INSERT FIGURE

C. LSH

The LSH algorithm REFERENCE HERE BOTH PAPERS is much more recent than either KD-Tree or KMeans, and it clusters data in a different way than its those algorithms. The simple goal of LSH is to obtain several hash values for each data point, which together make up its overall hash value.

The idea is that data points which are near each other will share the same or similar (differing by one number) hash values. Getting the hash values involves using this formula $H = \text{INSERT FORMULA}$ where H is the hash value and I is the vector. Apart from calculating the dot product, LSH benefits from being unaffected by the dimensionality of the data, because it mainly works with hash values for each data point, rather than the point itself. As one might expect, data points that have the same hash value are grouped in the same cluster. Another difference between LSH and the former algorithms is that the number of clusters is undefined. The number of clusters is dependent on how many vectors are used, the values of the vectors themselves, and how much the clusters are "cut" with the w value. Searching is done by following the same hashing steps on the query point, and searching data points in the cluster with the matching hash value. As described in the papers, the searching is only an approximate search, and no information is given for an exact search after LSH has been applied. This is one novel part of this research and our contribution, where we give a solution for an exact search. It is more complicated to find the appropriate additional clusters to search because it depends heavily on the vectors used for hashing. In two dimensions the clusters tend to look like parallelograms, and the closer the vectors are to being parallel, the thinner the clusters become, meaning clusters farther away of potential of containing the closest point. The technique simply follows a combination of the previous approaches for the other algorithms. Boundaries are found in each dimension, and a centroid and radius is found, for each cluster. For each query point, we find which of these boundaries is the furthest, and associate that value with each cluster. The larger of the two distances is used when trying to find the minimum distance because we cannot afford to exclude a cluster, since the boundaries offer different cluster shapes. This results in slightly more computation, but guaranteed correctness. Once the maximum boundary distance for each cluster is found, the searching becomes the with previous algorithms. If that distance is less than the distance to a data point found, that cluster's points must also be searched. It should be noted that this process of finding boundaries and radii, along with the exact search adds a great amount of time to the algorithm, though it is necessary for proper comparison. An example of clustering done with $m = 3$ and $w = 500$ is shown in Figure 3.

IV. TEST PROCEDURE

Testing of the algorithms was straightforward. Each algorithm was tested on each set of data, with different numbers of clusters in each test, so that the most efficient combination could be achieved for each algorithm. The time taken to cluster the data was measured independent from the time taken to search the data. For each algorithm the parameters that minimized the total clustering plus search time was used as a final comparison between the algorithms for each data set. It is important to measure the time for clustering and searching independently because one can gather a greater breadth of knowledge about the algorithm by investigating the

efficiency with which it performs each task. Each algorithm had a specific set of requirements imposed by the researchers for how the number of clusters should be chosen. The number of clusters (k) for KD-Tree was set to a power of 2. This is for simplicity and efficiency, as the algorithm works by splitting each previous cluster in half, so very little time is saved from not doubling the number of clusters at each step. For instance, instead of choosing 512 clusters, if one chooses 500 clusters, the clustering time is relatively unchanged and, except for particular data sets, the search time would be increased. The value for k when running KMeans was unable to be increased above 50 in most cases. The reasoning for this is explained further, however the simple explanation is even when increasing the efficiency by combining it with Bisecting-kmeans, the sheer amount of computation time with either large data sets or extremely high-dimensionality causes the algorithm to run longer than a reasonable amount (10+ hours) just for clustering. LSH is a different type of algorithm than the other two, in that the number of clusters is not specified by the developer. Instead, it is a bi-product of the number of vectors used and the way each vector is partitioned. Instead of choosing a specific number of clusters when running LSH, the researchers had to choose values of m and w such that an informative and reasonable number of clusters resulted. This involved some initial guess and check, partnered with more educated guess based on early tests.

V. RESULTS

The graphs from the respective tests are shown in Figures 4 through SOMETHING. A similar trend for all data sets appears from the graphs. KMeans was the slowest algorithm for both clustering and searching. The comparison of searching is rather unfair for this algorithm, because with large data sets, when the data is only clustered into at most 50 clusters, there is still a high number of points to exhaustively search within that cluster, so not as much time is saved in the clustering. In fact, the searching techniques for both KD-Tree and KMeans is quite similar, so the KMeans algorithm could, in theory, perform similarly in searching. However, it falls drastically behind in its clustering efficiency, so much so that the searching cannot even be compared on the same level. Both KD-Tree and LSH had efficient clustering of all data sets. This due to the fact that, unlike KMeans, neither algorithm requires performing distance calculations for all the points, and comparing these distances, until the searching phase, which is a commonality for all algorithms. Instead, each algorithm only performs a single (in reality it is one calculation for each dimension) calculation for each cluster, and compares across that calculation as oppose to the calculations needed for each point in KMeans. The main calculation needed in KD-Tree is to compute the variance for each dimension in each cluster so that it can split the cluster along the correct dimension. For LSH, the computation comes from computing the dot product with each data point and the vectors used for the hashing.

In most cases, KD-Tree had faster search times, while LSH presented drastically better clustering times. This is to be expected, as the characteristics needed to do a search on the

clusters for both of the algorithms involves the boundaries of the clusters. KD-Tree keeps track of these values in each phase of its clustering. However, LSH does not typically need that information unless it intends to perform an exact search, so these values must be calculated during the search time for LSH. There needed to be a distinct time for calculating these borders needed for an exact search in LSH. In the other algorithms this time is part of the clustering, however, in LSH it was added to the search time. The reasoning behind this choice is the fact that the other algorithms need to keep track of the boundary values in clustering, so adding them to their appropriate lists or arrays is not time consuming, in fact, it is the exact same time as if it was done after clustering. For LSH, on the other hand, the boundary values are not added until after the clustering is completed because it is not needed for clustering. In fact, unlike the other two algorithms, those cluster boundaries are not even needed for an approximate search of the single closest cluster. It is only needed for the exact search in which other clusters need to be checked. Figure SOMETHING+1 shows each algorithms best time for each data set.



Michael Shell Biography text here.

VI. DISCUSSION

The following section will discuss some contributions that were made as a result of this research, as well as some considerations for the future.

John Doe Biography text here.

A. Contributions

B. Future Work

VII. CONCLUSION

The conclusion goes here.

APPENDIX A

PROOF OF THE FIRST ZONKLAR EQUATION

Appendix one text goes here.

APPENDIX B

Appendix two text goes here.

ACKNOWLEDGMENT

The authors would like to thank...

Jane Doe Biography text here.

REFERENCES

- [1] H. Kopka and P. W. Daly, *A Guide to L^AT_EX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.