K-Means clustering variants and improvements

Stefan Sebastian, 242

November 16, 2018

Abstract

TODO make abstract

Contents

1 Introduction

1.1 Motivation

In recent times there has been an explosive growth of data. This new, digital resource has become so important that a recent article from The Economist magazine has called it 'the new oil'[1]. Some important sources of data, gathered by Bernard Marr[2], a business consultant in big data and AI technologies, are: Social Media (every minute 456000 tweets are sent on Twitter, 46740 photos are uploaded on Instagram, 4146600 YouTube videos are being watched), Communication (16 million text messages and 156 million emails are sent every minute), Services(45788 trips are made through Uber per minute), Internet of Things(8 million people use voice control every month). The amount of data in the world is predicted to reach 163 zettabytes by 2025[3], where one zettabyte represents a trillion gigabytes.

The main reason for choosing the K-means method for this paper is because I believe that the development of a computationally efficient unsupervised learning method will be necessary to keep up with future's data, which will continue to grow and sometimes lack meaningful labels.

1.2 Clustering

Clustering is a field of data analysis whose purpose is to organize points, or objects into natural clusters. In other words given N objects described by a set of features, find a number of groups, such that objects in the same cluster are similar to each other and different from objects in other clusters[4]. Given that there is no rigorous definition of similarity and that clusters in real data can overlap there are multiple algorithms and difference measures proposed in research.

1.3 K-means

The K-means algorithm was discovered independently in multiple scientific fields around the same time and its name comes from a paper published by MacQueen in 1967[4]. Despite being so old, this method is still widely used in clustering tasks because of its efficiency, simplicity and proven success.

The main goal of this algorithm is to minimize the distance between cluster centers and points assigned to them, which is a NP-hard problem. For this reason K-means is a greedy algorithm and has a chance to get stuck in local minimum instead of the best solution[4].

A basic version of the K-means algorithm is[5]:

1. Select an initial partition with K clusters and repeat 2 and 3 until the point membership no longer changes
2. Assign each point to the closest center
3. Update cluster centers based on the new assignment

2 Improvements

This section presents some recent research results from two of the most researched aspects of K-means, computation time optimization and initial seed selection. The first is done with the aim of scaling the algorithm to huge datasets while the other has the goal of fine tuning the accuracy of this algorithm.

2.1 Execution time optimizations

As illustrated in the introductory section, data sets are growing larger and more varied. Consequently there is a need for algorithms that scale well and obtain good results without having huge computational costs. K-means is such an algorithm, since it is characterized by simplicity and has been proven to work empirically with a multitude of datasets[4]. This success has driven researchers to scale the algorithm for use on larger datasets. Next, I will present some of these execution time optimizations for the algorithm.

2.1.1 Enhanced K-means

A simple improvement for the basic K-means has been proposed by Fahim et al.[6] under the name of "enhanced K-means". The main idea is that some time can be saved during the computation of distances between points and cluster centers in the update phase of the original algorithm. This distance is calculated at every iteration of the algorithm but the information from previous iterations could be used, by employing an additional data structure that memorizes the previous distance between a point and its cluster's center. At every iteration the distance to the new cluster mean is calculated and if it's less or equal to the previous distance stored for that point then it is left in its cluster and there's no need to calculate the other k-1 distances.

The complexity of this algorithm can be approximated to O(nk) compared to the complexity of the classic algorithm which is O(nki), where n represents the number of points, k the number of clusters and i the maximum number of iterations. In brief, the time complexity when updating the position of a point is O(1) if it stays in its cluster and O(k) otherwise. Since the algorithm converges to a local minimum then the number of points updated in each iteration decreases, meaning the expected complexity is $nk\sum_{j=1}^{i} 1\,/\,j$.

The performance of the algorithm was tested on 3 different datasets, with the following number of records and features: (20000, 16), (4177, 7), (6574, 15) and was compared with the classic K-means and the CLARA algorithm. The enhanced K-means obtains similar results on smaller number of clusters (approximatively less than 50) but outperforms the other methods on larger values being about twice as fast as CLARA and 5 times as fast as the classic variant on more than 100 clusters. In conclusion this algorithm can be a useful optimization for working with datasets in which we expect to find a large number of clusters.

2.1.2 K-means clustering using MapReduce

MapReduce[7] is a framework for processing large datasets over a cluster of computing nodes. It offers a programming model based on map and reduce functions and it has gained popularity in the big data community due to its simplicity, fault tolerance and scalability. This framework is a good starting point for clustering implementations that can work at large scale.

Zhao et al.[8] provide a detailed description of a simple parallel K-means implementation using MapReduce. The algorithm, called 'PKMeans' uses only one MapReduce job: a map function that assigns each sample to the closest center and a reduce function which updates the cluster centers. The input dataset is stored on a distributed file system as a sequence of (key, value) pairs, where the key is the position of record in the data file, and the value is a string content of the record. Center values are stored on a global variable, so map functions can calculate distances independently. After each map task a combine function is applied which computes partial sums of the points assigned to the same cluster and their count. The reduce function takes as input the result of the combine task and uses it to compute the new means. The algorithm was tested with a number of computers varying between 1 and 4 and datasets of 1 up to 8GB and reported a linear speedup in computation time, meaning a system with m systems runs the algorithm m times faster. However, this speedup should decrease on larger systems due to increases in communication costs.

2.1.3 K-means algorithm on the GPU

A Graphical Processing Unit[9] is a special circuit designed for computer graphics and image processing. In contrast to CPUs, they have over a hundred cores that can handle thousands of threads at the same time, meaning GPUs are optimized for applying the same operation over a large batch of data. For this reason, a lot of data analysis algorithms have been implemented with this technology.

A sample implementation of K-means on the GPU has been published by Li et al.[10]. The algorithm can use two strategies, depending on the dimension of the data.

If the features of a data point can be stored on the GPU registers then, for the distance calculation step, each of the n points is dispatched to a thread. In the other case, a flow similar to matrix multiplication is used with the following matrices: data[n][d], centroid[d][k] and result[n][k]. To optimize latency, the matrices are processed in tiles, instead of individual elements. After this operation, in order to find the closest centroid, the task of analyzing a row in 'result' to find the max value is distributed across n threads.

For the step of updating the values of the means a divide and conquer strategy is used. The data is first divided into $n' = n/M$ groups, where M is the number of multiprocessors on the GPU and each group is reduced to get temporary centroids. The obtained data is then split again into M groups and the same process loops until the resulting number of loops is lower than M.

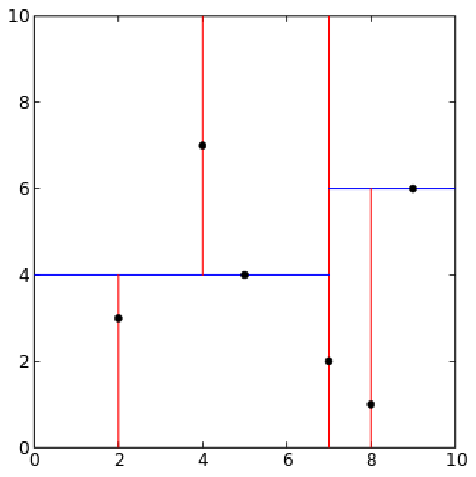
The first strategy is tested on datasets with up to 4 million points, 400 clusters and 8 feature dimension, finishing the computation in just 2.2 seconds. An impressive result for the second strategy is processing 494.080 points, with a 100 clusters and 34 features in just over a second.

2.2 Seed selection

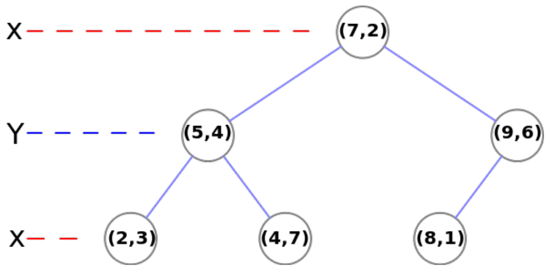
The K-means algorithm converges to a local optima[5], meaning it can be strongly influenced by the seed selection process. The most common approach is random sampling. However, if the initial seeds are near a local optima then the algorithm is unlikely to find the best cluster centers. For this reason, various strategies have been proposed for efficient seed initialization.

2.2.1 Sort and split

A conceptually simple heuristic for choosing the K-means seeds was proposed by Yedla et al.[11]. The main idea of the method is to sort all points in the initial dataset by



(a) An example of a 2d k-d tree[13]



(b) The data structure[14]

Figure 1: K-d tree graphic representation

distance from origin. In case that there are points with negative feature values then they are transformed, by subtracting the minimum attribute value out of the feature value for each point. After sorting, the data is partitioned in k equal sets, where k is the number of desired seeds. From each set, the middle point is chosen as a seed. This method was compared to the classic algorithm that uses the random seed initialization and has been shown to obtain better accuracy on all testing sets.

2.2.2 K-d trees

A k-d tree, introduced by Jon L. Bentley[12], meaning k-dimensional tree is a data structure used to organize points in k-dimensional space, which is useful for partitioning data and optimizing multidimensional queries. The algorithm for building such a tree is recursive: select a point in the current dataset, choose a dimension, create a node of the selected point with a left subtree made out of the points with a lower value on the chosen dimension and a right subtree with the others. The space complexity for this structure is O(n) and all operations have an average time complexity of O(log n) and O(n) in the worst case.

An implementation of seed selection based on the k-d structure is proposed by Redmond and Heneghan[15]. The reason for using this data structure is to gain information about the density of the dataset. In order to suit the problem some decisions have been made while building the tree. At each step the dataset is split among the longest dimension of the bounding box. This box can be computed by taking the minimal and maximal values for each feature in the dataset, and building two new points. The longest dimension is simply the feature for which there is the greatest difference between the two points calculated in the previous step. The chosen condition for stopping and making a leaf node is when the size of the considered dataset is $\frac{n}{10K}$ or lower, which allows approximatively 10 leaf buckets for each cluster.

After the k-d tree is built some density estimations are made using the obtained leaf buckets. For each bucket the value of the density is calculated as N/V, where N is the number of points it contains and V is the volume of the bounding box. The volume is computed as the product of the differences between each feature of the two bounding points, or the geometric mean in case of two features being equal.

The heuristic for choosing the seeds is to get points that have a large density and are separated by a reasonable distance. Thus, the first seed is equal to the mean of the bucket with the highest density. The other k-1 seeds are chosen to be the ones that maximize the value g, where g is the product between the density of the bucket mean and the minimum distance to other bucket means. An optimization to avoid outliers is to compute the seeds, remove 20% of the least dense buckets, and then compute another pair of seeds.

The algorithm was compared to another 3 initialization methods (Forgy, Bradley and Fayyad, Katsavounidis), on 36 datasets and obtained better Distortion values on over 30 compared to each of the other methods. Distortion is the sum of the distances between each point and its cluster center.

2.2.3 ROBIN

ROBIN stands for Robust Initialization and is a method designed by Hasan et al.[16], with the main goal to avoid choosing outliers as seeds for the K-means algorithm. For this reason the measure of LOF(Local Outlier Factor) is used. To compute this for a point x, we first need the neighborhood $N(x, mp)$ on the threshold mp(mininum points). This is a set of all the points closer or equally close to x than the mpth nearest neighbor. The $density(x, mp)$ value is calculated as the fraction of the size of the neighborhood and the sum of distances between x and all points in N. The average relative density of x, $ard(x, mp)$, is then computed as the ratio of $density(x, mp)$ and the average density of its nearest neighbors. The LOF score is then calculated as the inverse of the ard value.

The algorithm stands from a reference point r, which can be any point in the dataset. It then sorts the points in decreasing order of distances from r and chooses the first point that has a LOF value around 1. In the following iterations, instead of r, the points are sorted in decreasing order of the minimum distances from the points that have been chosen as centers. The worst case computational complexity is $O(n \cdot logn \cdot k \cdot d)$, due to the sorting method.

The method was evaluated on a set of synthetic datasets, with d(the number of features) varying from 8 to 24 and k going from 10 to 50 and the number of data points generated for each of the clusters being a random value between 100 and 1000. For each of these datasets the ROBIN initialization method has performed better than the random selection method in terms of distortion, and has obtained results close to the optimal ones.

3 Variants

3.1 Bisecting K-means

Bisecting K-means[17] is a variant of the K-means algorithm that can perform both flat and hierarchical clustering. The name of the algorithm is related to the fact that, at each iteration it splits a cluster in two. The algorithm, in its simplest form, looks like this:

1. Select a cluster to split
2. Split the cluster into two, and repeat this for a configured number of times, keeping the split which produces the best clustering

3. Repeat the previous steps until k clusters have been obtained

There are different strategies for choosing a cluster to split in the first step: the largest cluster, the one with the least similarity, a combination of these. Experimentally, it has been shown that the performance difference between them is minimal.

The bisecting K-means has a few advantages over the basic algorithm. First of all, it is more efficient when k is large, because on the computation of each iteration, the distance to only two centroids are calculated. Secondly, the algorithm tends to produce more balanced clusters than the classic method in terms of cluster size. Thirdly, as shown in the original paper[17], bisecting K-means generally obtains smaller entropy than the simple K-means method.

3.2 Genetic Algorithms

Genetic Algorithms[18] are an heuristic search method inspired by nature. They are efficient and adaptive and can provide a near optimal evaluation of a function(fitness). The main advantage over other search methods is the fact that GAs can avoid getting stuck in local optima by using some random operations. Next I will explain a simple genetic algorithm. First, an initial set of random solutions (or chromosomes), called the population, is generated. Then a fitness value is computed for each solution. A selection is made based on the fitness of each solution. Those with a better fitness are more likely to be chosen. Two kinds of operations are then performed on the selected candidates: mutation(a small, random change in a solution) and crossover(a combination of two solutions). After this, a new population has been generated, and the process is repeated for a configured number of times or until the fitness reaches a certain threshold. The main advantage of genetic algorithms is that they can be used for a varied number of problems, the only aspects being determined by the problem is the representation of a solution and the computation of the fitness function. K-means, having the vulnerability of being stuck in local optima is a good candidate for a genetic algorithm implementation.

3.2.1 KGA-clustering

One such implementation, entitled KGA-clustering, is presented by Maulik and Bandyopadhyay[19]. Firstly, they chose to encode the solution as a list of cluster centers. Each chromosome encodes a sequence of D * K float numbers where, D is the dimension of the data and K is the desired number of clusters. The clustering metric used is the distortion, meaning the sum of euclidean distances from each point to its cluster mean. Similarly to the basic K-means algorithm, the initial centers are points chosen at random from the dataset.

For each chromosome, after decoding, the distances from each point and the new centers are calculated. The solution is then replaced by a new chromosome containing the updated means, and the fitness function is calculated. After going through all the population and if the termination criterion is not met, the genetic operators are applied. Selection is done following the survival of the fittest strategy, meaning that the chance of a chromosome to be selected is proportional to its fitness. The crossover operator used is single-point crossover with a fixed probability. A random point is selected and the portions of the chromosomes to the right of the point are exchanged. Mutation is also done with a fixed probability. A position is chosen inside a chromosome and is updated as in equation 1. In this equation, x^i is the selected feature of a point, $x^i_{\min}, x^i_{\max}$ are the minimum and maximum values over that dimension and λ is generated randomly in range $[-R, R]$. The function for R is given in figure 2, and M represents the value of the chromosome's fitness.

$$x^i = \begin{cases} x^i + \lambda \cdot (x^i_{\max} - x^i) & if \quad \lambda >= 0 \\ x^i + \lambda \cdot (x^i - x^i_{\min}) & otherwise \end{cases} \tag{1}$$

$$R = \begin{cases} \frac{M - M_{\min}}{M_{\max} - M_{\min}}, & \text{if } M_{\max} > M \\ 1, & \text{if } M_{\min} = M_{\max} \end{cases} \tag{2}$$

The KGA-clustering algorithm has been evaluated on several artificial and real life datasets with dimension going from 2 to 5, clusters from 2 to 6 and up to 262.144 data points. It has been observed that it obtained an optimal value more often than the simple K-means algorithm.

3.2.2 Improved Genetic K-means

Another variant of genetic K-means has been proposed by Guo et al.[20]. What is interesting in this approach is the fact that the number of clusters is not known in advance, but is part of the solution. Thus, a chromosome contains, k, the number of clusters, and all the chosen means.

The main goal of IGKM is to find a small number of compact clusters with a large separation between them. In order to do this, it uses the following fitness function : $fitness(k) = 1/k \cdot E_1/E_k \cdot D_k$. The first factor, k, is the number of clusters, and is used to penalize solutions with a lot of clusters. E_k is the sum of euclidean distances between points and their cluster centers, when we consider the first k clusters. The fitness function increases as the value of E_k decreases, meaning that it encourages the formation of compact clusters. D_k is used to measure the separation between clusters and is calculated as $D_k = max^k_{i,j=1} \|z_i - z_j\|^2$, where z_i and z_j are the centers for the clusters i and j. This encourages solutions that have at least 2 clusters with a high degree of separation.

The other steps are similar to KGA[19], with the consideration that there needs to be a handling method for invalid chromosomes resulting out of crossover.

IGKM was compared to the classic K-means method and to other genetic implementations over a set of small datasets (n ¡= 214) and has been shown to find the optimal number of clusters and the best partition in each of them.

4 Conclusions

TODO something conclusive

References

[1] "The most valuable resource." https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data. Accessed: 2018-11-11.

[2] "How much data do we create every day? the mind-blowing stats everyone should read." https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read #7f96d4a060ba. Accessed: 2018-11-11.

[3] "What will we do when the world's data hits 163 zettabytes in 2025?." https://www.forbes.com/sites/andrewcave/2017/04/13/what-will-we-do-when-the-worlds-data-hits-163-zettabytes-in-2025/#4ff934f6349a. Accessed: 2018-11-11.

[4] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651 – 666, 2010. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR).

[5] R. C. D. Anil K. Jain, *Algorithms for clustering data*, ch. Clustering Methods and Algorithms, pp. 96–97. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1988.

[6] A. Fahim, A.-B. M.Salem, F. A. Torkey, and M. Ramadan, "Efficient enhanced k-means clustering algorithm," *Journal of Zhejiang University*, vol. 7, no. 10, pp. 1626–1633, 2006.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, pp. 137–150, 2004.

[8] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *Cloud Computing* (M. G. Jaatun, G. Zhao, and C. Rong, eds.), (Berlin, Heidelberg), pp. 674–679, Springer Berlin Heidelberg, 2009.

[9] K. Krewell, "Whats the difference between a cpu and a gpu?." https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/, 2009. Accessed: 2018-11-14.

[10] Y. Li, K. Zhao, X. Chu, and J. Liu, "Speeding up k-means algorithm by gpus," *Journal of Computer and System Sciences*, vol. 79, no. 2, pp. 216 – 229, 2013. 10th IEEE International Conference on Computer and Information Technology, 2010.

[11] Y. Madhu, S. Rao Pathakota, and S. , "Enhancing k-means clustering algorithm with improved initial center," *International Journal of Computer Science and Information Technologies*, vol. 1, pp. 121 – 125, 01 2010.

[12] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, pp. 509–517, Sept. 1975.

[13] KindDragon33, "Kdtree figure 1." https://en.wikipedia.org/wiki/File:Kdtree_2d.svg, 2006. Accessed: 2018-11-15.

[14] MYguel, “Kdtree figure 2.” `https://en.wikipedia.org/wiki/File:Tree_0001.svg`, 2008. Accessed: 2018-11-15.

[15] S. J. Redmond and C. Heneghan, “A method for initialising the k-means clustering algorithm using kd-trees,” *Pattern Recognition Letters*, vol. 28, no. 8, pp. 965 – 973, 2007.

[16] M. A. Hasan, V. Chaoji, S. Salem, and M. J. Zaki, “Robust partitional clustering by outlier and density insensitive seeding,” *Pattern Recogn. Lett.*, vol. 30, pp. 994–1002, Aug. 2009.

[17] M. Steinbach, G. Karypis, and V. Kumar, “A comparison of document clustering techniques,” in *In KDD Workshop on Text Mining*, 2000.

[18] T. Segaran, *Programming Collective Intelligence*, ch. 5. O’Reilly, first ed., 2007.

[19] S. Bandyopadhyay and U. Maulik, “An evolutionary technique based on k-means algorithm for optimal clustering in rn,” *Information Sciences*, vol. 146, no. 1, pp. 221 – 237, 2002.

[20] H. Guo, T. Liu, S. Gao, and K. Zhu, “An improved genetic k-means algorithm for optimal clustering,” in *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW’06)(ICDMW)*, vol. 00, pp. 793–797, 12 2006.