K-MEANS CLUSTERING, AN ATTEMPT AT AN IMPROVED SEEDING PROCESS

by

Yaser Alkayale

Submitted in partial fulfillment of the requirements
for the degree of Bachelor's of Computer Science, Honours

at

Dalhousie University
Halifax, Nova Scotia
April 2018

This thesis is in dedication to my grandfather whom I was named after.

Table of Contents

Abstract

Clustering is a well-known task that has been studied and used for decades. The idea is to take a set of items and group them into a number of clusters based on a similarity measure. K-Means , although named differently back then, was proposed in 1957 by Stuart Lloyd and is one of the most widely used clustering algorithms and is still used today for its reasonably fast heuristic to find the clusters based on the Lloyd algorithm and more recent developments in that area. K-Means has two main parts to clustering, the initial seeding process and the iteration process. The seeding process picks k initial seeds as cluster centres, and highly affects the accuracy of the final result in the algorithm. The iteration process dominates running time to move the centres around until it converges to an optimum. In this paper, we discuss a new method of the seeding process that gives us more accurate seeds to start the algorithm.

Acknowledgements

A sincere thank you to my supervisor Dr. Norbert Zeh. Without his assistance, this project would not have seen the day of light. Thank you to Dr. Vlado Keselj who made himself available when we needed to consult with him. Also big thank you to Arazoo who was with me from the beginning, and went through my ideas with me.

Chapter 1

Introduction & Background

Clustering is the task of grouping certain things into separate or overlapping groups based on a similarity criteria. Clustering problems arise in many domains like natural language processing (Ravichandran et al., 2005), bioinformatics (Edgar, 2010) crash report analysis (Soto et al., 2016), and vehicle navigation (Maio et al., 1996). The notion of what is a good cluster highly depends on the domain and application at hand. Many clustering techniques like hierarchical clustering (Corpet, 1988) and graph-based ones (Schaeffer, 2007) exist, each serving a different purpose and are practical for different clustering tasks. K-Means clustering continues to be one of the most popular clustering algorithms for it's simplicity of implementation and relative efficiency (Jain, 2010).

Different clustering algorithms are suited for different tasks. Some of them, like K-Means , fit a given dataset into a given number of disjoint sets. Other algorithms allow points to belong to multiple clusters and are useful in applications like community clustering on social media graphs (Epasto et al., 2017). In other instances, clustering is used on datasets where the number of clusters is unknown; for example, clustering images of people using facial recognition (Schroff et al., 2015). The effectiveness of a given algorithm is determined by domain it is being used in and the problem at hand.

Formally, K-Means is the problem where one is given a set of n points in d-dimensional space, $\mathbb{R}^d$, and a number k. The goal is to split the n points into k disjoint clusters, minimizing the cost function ϕ, the sum of distances of each point to its cluster centre. K-Means does especially well, in terms of speed and accuracy, on convex shaped clusters as it minimizes the sum of distances of all points to their belonging cluster centres. However, it struggles to perform well in terms of accuracy and running time when clusters are not convex shaped or k is not the right number of actual clusters in the dataset.

Solving K-Means exactly is known to be NP-hard, and that is why heuristics like Lloyd's (Lloyd, 1982) iterations were introduced to give us an approximation of the solution by using local search. Lloyd is still the most widely used iterations process for its simplicity of implementation. It has been stated that K-Means "is by far the most popular clustering algorithm used in scientific and industrial applications" (Berkhin, 2002). A locally optimal solution is good enough in most cases, allowing us to have meaningful clusters in a reasonable amount of time. That, in addition to simplicity of implementation, makes K-Means a good option in a lot of cases, hence the popularity.

While K-Means is widely used for its proven practical performance, it is not perfect for many use cases. The algorithm can give highly inaccurate clusters if the incorrect number k is given to the algorithm. This is due to the nature of the algorithm forcing all the points into k clusters. Another problem with the algorithm is that it produces convex shaped clusters, which inherently produces highly inaccurate clusters if the correct clusters are not convex shaped. The algorithm also struggles when there is a lot of noise[1] in the data because having points between two clusters makes it difficult to force them into only one as membership becomes unclear. Inputting the incorrect number k to the algorithm and/or having incorrect data leads to the algorithm struggling with running time and accuracy.

In this paper, we introduce a new way to seed the K-Means algorithm which turned out to be no better than randomly seeding K-Means . The ideas of the algorithm and its pseudocode are outlines below along with results. We also explain the C++ implementation of the algorithm and ways it can be used for further development and research.

1.1 The K-Means algorithm.

The K-Means algorithm is simple and relatively efficient as it minimizes the objective function locally depending on its seeds. Here are the steps of the algorithm Arthur & Vassilvitskii (2007). Input is assumed to be the dataset X and the number of clusters required k:

[1]Note: Noise in this context is referring to the overlap of multiple clusters, where it becomes difficult to separate points if they are at the edges of two or more clusters.

1. Pick k points arbitrarily at random from X.

2. Point p in X belongs to cluster C_i if p is closer to c_i than it is to c_j for all $j \neq i$, $1 \leq i, j \leq k$

3. For i in {1...k} compute new centre $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$

4. Repeats steps 3 and 4 until centres do not change[2].

Of course, the algorithm may not always converge in a reasonable amount of time, so a stop condition is used when the centres do not change beyond a threshold at a certain point. Another heuristic that is used is to set a maximum number of iterations to run.

1.2 Seeding Process

The seeding process of the K-Means algorithm is crucial to the accuracy of the output because while the iterations converge the centres to an optimum, it is localized to the regions constrained by the initial seeds (Arthur & Vassilvitskii, 2007). Many new approaches to pick initial seeds like K-Means++ (Arthur & Vassilvitskii, 2007), and K-Means|| (Bahmani et al., 2012) have been introduced which help pick seeds that converge faster. It has been experimentally proven that having a better seeding process improves the algorithm by both lowering the objective function of the algorithm, and allowing it to converge faster with less iterations and running time (Arthur & Vassilvitskii, 2007).

Given the dataset X and the number $k \leq |X|$ to cluster, K-Means picks k points from the dataset uniformly at random. The Lloyd method does not specify any way of picking the initial seeds. It only required k seeds to run and converge an a local optimum. The widely used method of seeding K-Means is to do it uniformly at random. Picking the seeds uniformly at random helps works because allows the iterations to start at some points, but doesn't help in minimizing the objective function in any meaningful way. Newer ways have been introduced that substantially improve both running time and accuracy of the algorithm.

[2]In reality, the centres rarely ever stop changing, so heuristics need to be used to stop the algorithm in reasonable time.

It it important to note here that we are only able to compute an optimum, not the optimal solution because the problem is NP-hardMahajan et al. (2009). This means that we are unable to compute the optimal solution of any large enough instances in reasonable time. For that, a heuristic, i.e. Lloyd's iterations, are used to compute reasonably accurate centres in time.

1.2.1 K-Means++: A Better Seeding Process

K-Means++ uses an intuitive way to seed the initial clusters of the algorithm by trying to pick seeds that are as far apart as possible. This is done by giving a higher probability for points that are further away from the ones already picked. The way it works is that we pick a random initial point, and then given a higher probability of picking a point disproportional to its distance the seeds already picked, until k seeds are pickedArthur & Vassilvitskii (2007).

K-Means++ works well because by skewing the probabilities of picking seeds towards ones that are farther from the already picked ones, we are helping pick seeds in different clusters in the dataset. It works especially well because a concentration of points already has a high probability of picking at least one of them picked, and lowering the probabilities for points that are close to ones that have already been chosen allows us to probabilistically almost ensures having a point from each cluster.

K-Means++ works very well because it about evenly distributes the initial seeds in the dataset. It is important to note that the probabilistic model still gives a chance for any of the points to be chosen, just some higher than others. This is crucial as we are merely making "good guesses" as to what the initial seeds should be, but are in no way trying to compute the "perfect seeds" because that would be solving the K-Means problem, and as discussed earlier, that is NP-hard.

From my research, K-Means++ seams to be the best scientifically proven way to seed the K-Means algorithm. There, however, have been improvements on the K-Means++ seeding algorithm like K-Means|| which improves the algorithm by making it parallelizable for very large datasets.

1.3 Iterations

The iteration process of the K-Means clustering algorithm dominates the running time of the algorithm. Running time for random seeding is $O(k)$, while the iterations run in $cO(nkd)$. Depending on how quickly the iterations converge, the c constant may be very large. Even with K-Means++, running time for seeding is $cO(nkd)$ but $c = O(k)$, and in practice is still dominated by the iterations running time. Different approaches have been used to run the iterations and converge the centres. These approaches have had high success in reducing the running time of the algorithm Alsabti et al. (1997). However, the implementation of these methods are not as simple as Lloyd's iterations, making them not as widely used.

1.4 Objective Function

The objective function, ϕ, of the clustering task determines based on what the points are being grouped. In most tasks, including in everything discussed in this paper, the objective function is the sum of squared euclidean distances[3] of all the points to their cluster centres. More formally expressed as:

$$\phi = \sum_{x \in X} \min_{c \in C} |x - c|^2$$

This objective function is the reason why K-Means inherently produces convex shaped clusters as it tries to minimize the distance of each point to its cluster centre. If a point is marginally closer to a centre than another, it is grouped with the closer one. Geometrically this disallows concave shapes of any cluster. This is good for datasets where clusters are geometrically convex or close to it, but poses problems where the underlying data is not convex or is not clearly separable.

1.5 Difficulties

K-Means clustering is very good at clustering well-separated convex clustered where k the number of clusters in known or accurately measured in advance. However, this

[3]Note: Here squared distance is used because comparing distances and the square maintains the ordering of the points and saves a large amount of computational time as square rooting numbers is computationally costly.

is very rarely the case. Another issue with K-Means is that it fits every point into a single cluster. This means that a point has no way of belonging to different clusters at the same time.

Chapter 2

Improved Seeding Process?

Here we introduce the new seeding process for K-Means . The basic idea is to run the randomized seeding process T times, each time running a small, S, number of iterations which produce a set of k centres that have not converged. We then take the $(T * K)$ centres, and cluster those into k clusters.

2.1 The Algorithm Pseudocode

Algorithm 2.1: Consensus Seeding. Note: kmeans.cluster is a function that takes three values, the dataset, the number k of clusters and an optional value, the maximum number of iterations for the algorithm. The maximum number of iterations is set to infinity by default.

```
Input  : $k$: the number of clusters
         dataset: the $d$ dimentional dataset to be clustered
         $r$: the number of rounds to run for the consensus
         $m$: the maximum number of iterations to use for each round
Output: List of $k$ cluster centre seeds
1 runCentres ← {};
2 for $i = 0...r$ do
3     $runCentres \leftarrow runCentres \cup kmeans.cluster(dataset, k, m)$;
4 end for
5 return kmeans.cluster(runCentres,k)
```

2.2 Analysis & Datasets

To analyze the performance of the algorithm we ran it on 3 different datasets. The Cloud Data Set has 1025 values in 10 dimensions, and they represent the values for cloud cover (Collard, 1989). The Spambase Dataset consists of 4601 values in 58 dimensions, and it represents numerical representations for spam emails (Forman, 1999). The Dimred dataset comes from a dimensionality reduced textual corpora of crash reports, it has $108,669$ values in 2 dimensions.

Tables 2.1, 2.2, and 2.3 show experimental results for the accuracy and running time of the algorithms. They were all tested on a 2017 15-inch Apple Macbook Pro, with an 2.8 GHz Intel Core i7 and 16Gb of RAM. The iterations were set to maximum 200, and the T and m values were both set to 5 for K-MeansConsensus. Each algorithm was run 20 times for each combination of k-value, dataset and algorithm used, and the average and minimum of these were computed.

	cloud								
	Average ϕ			Minimum ϕ			Average T (seconds)		
k	K-Means	K-Means++	K-MeansConsensus	K-Means	K-Means++	K-MeansConsensus	K-Means	K-Means++	K-MeansConsensus
10	7.61e6	6.29e6	8.35e6	6.29e6	5.77e6	6.25e6	0.00e0	0.00e0	0.00e0
25	3.69e6	2.15e6	3.95e6	2.46e6	2.04e6	2.89e6	0.00e0	0.00e0	0.00e0
50	1.91e6	1.18e6	2.06e6	1.28e6	1.11e6	1.28e6	0.00e0	0.00e0	0.00e0
100	1.14e6	6.57e5	1.12e6	8.52e5	6.37e5	8.71e5	0.00e0	5.00e2	0.00e0

Table 2.1: Experimental results on the cloud dataset (n=1025, d=10)

	spam								
	Average ϕ			Minimum ϕ			Average T (seconds)		
k	K-Means	K-Means++	K-MeansConsensus	K-Means	K-Means++	K-MeansConsensus	K-Means	K-Means++	K-MeansConsensus
10	1.70e8	9.13e7	1.70e8	1.70e8	7.70e7	1.70e8	1.50e1	5.00e2	3.00e1
25	1.51e8	1.79e7	1.50e8	1.43e8	1.61e7	1.32e8	1.00e0	3.50e1	1.00e0
50	1.44e8	6.81e6	1.40e8	7.46e7	6.23e6	7.56e7	1.75e0	7.50e1	1.90e0
100	1.36e8	2.43e6	1.36e8	7.30e7	2.23e6	6.29e7	1.20e0	2.50e0	1.80e0

Table 2.2: Experimental results on the spam dataset (n=4601, d=58)

	dimred								
	Average ϕ			Minimum ϕ			Average T (seconds)		
k	K-Means	K-Means++	K-MeansConsensus	K-Means	K-Means++	K-MeansConsensus	K-Means	K-Means++	K-MeansConsensus
10	7.02e6	7.02e6	7.03e6	6.99e6	6.99e6	6.99e6	5.00e1	5.50e1	5.00e1
25	2.79e6	2.79e6	2.79e6	2.77e6	2.77e6	2.77e6	1.10e0	1.60e0	1.15e0
50	1.40e6	1.39e6	1.39e6	1.38e6	1.38e6	1.38e6	2.35e0	3.00e0	2.55e0
100	7.07e5	7.05e5	7.07e5	7.00e5	6.99e5	7.01e5	4.90e0	6.85e0	5.35e0

Table 2.3: Experimental results on the dimred dataset (n=108,669, d=2)

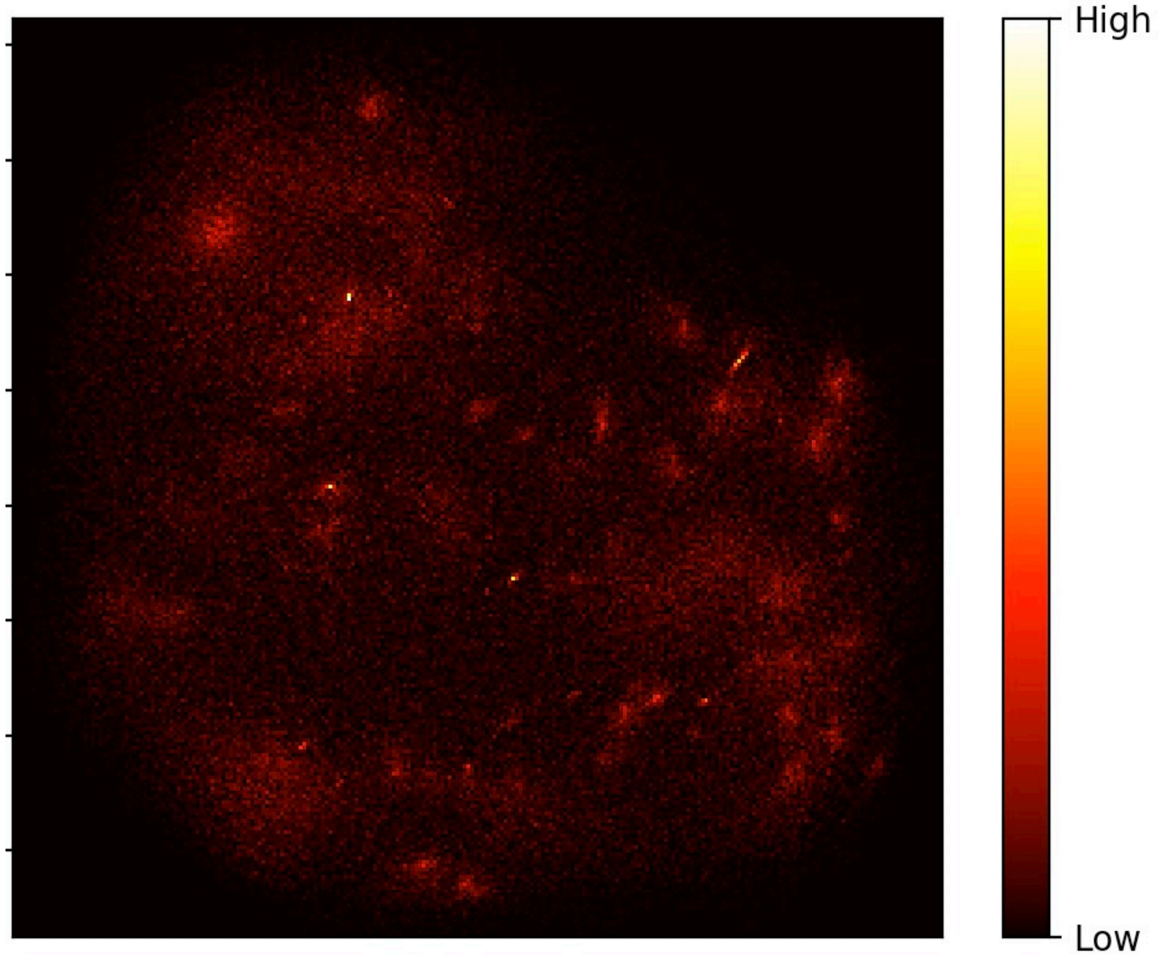


Figure 2.1: Heat map of the Dimred dataset. The lighter versions of the figure are for

Figure 2.1 is a heat map of the Dimred dataset which shows the concentration of points.

We can clearly see that the new new seeding process gave us marginally better results than K-Means in some situations. In most cases, it did equally as bad if not worse. We also note that K-Means++ did considerably better in most cases. From that we conclude that the new seeding process is no better than K-Means .

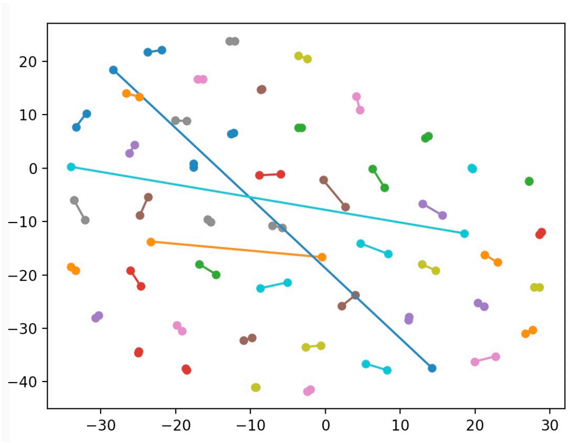
It is important to note here that on the dimred dataset, unlike the other datasets, all three algorithms achieve similar results.

2.3 Minimum Weight Perfect Matching on Bipartite Graphs

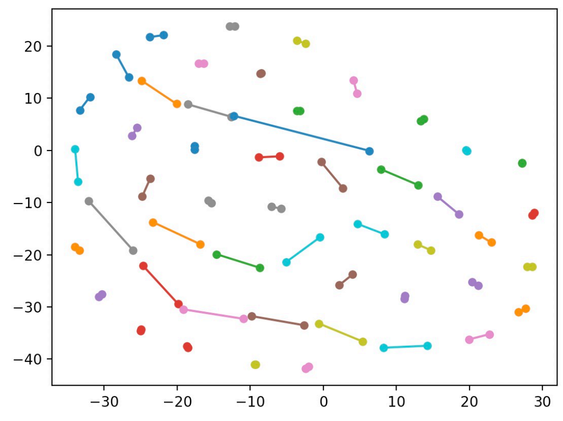
In an effort to understand the robustness of the algorithms, we decided to see how consistent the results of the algorithm were were multiple consecutive runs. For that, we used the Hungarian algorithm for perfect matching introduced by**?**. We learned a lot about the underlying properties of the algorithm by analyzing our perfect matching results. First, we realized how perfectly matching two runs of the algorithm does not necessarily mean perfectly matching the centres based on their distances in euclidean space. Second, minimizing the total distance between pairs in the matching may not be the best way to best match the centres.

Perfect matching is the problem where a graph is partitioned into pairs based on a heuristic. Minimum weight perfect matching is where the total costs of the edges connecting the pairs in the graph are minimized. The is an NP-hard problem on a general graph; however, it can be solved in $O(n^3)$. The Hungarian algorithm is one of the best known problems to solve the assignment problem, the more widely used name of the minimum weight perfect matching problem.

The assignment problem is where given a set of w workers and t tasks, we are asked to find the best worker-pair pairing so that the total cost to perform all tasks is minimized.



(a) Perfect matching using maximum overlap of points in centres



(b) Perfect matching using distances of cluster centres to each other.

Figure 2.2: Example of different cost functions for perfect matching.

The two figures are of clusters centres plotted in euclidean space. The axis are not labeled because it adds no extra information to our findings in this case. Every pairing of same-colour points on the plots come from 2 different runs of the algorithm. We perfectly match the 100 points (50 per run), and get some interesting results.

Here we can see Figures 2.2a & 2.2b are very different even though they are run on the same datasets. There is a very good explanation for this that may not be clear at first glance. Figure **??** seems to be inaccurate as there are clear pairings of points that weren't paired together. Rather than the closest points being paired, we see how there is a chain of pairings. This is because running the minimum weight perfect matching on the points minimizes the overall distances between all the pairs, but does not take into consideration the actual underlying pairing that we are looking for.

Switching the objective function in Figure 2.2a where we compute the maximum weight perfect matching of the centres based on the number of shared points in the clusters. Here, we notice a very interesting pattern where the pairings seem to be near perfect overall with a few outliers that is far apart. While this is not exactly the expected result, it makes sense because for this dataset, we are not sure of the exact underlying number of clusters k.

2.4 Choosing the Correct Matching Function

In our experiments, we were looking to perfectly match 50 centres on multiple runs of the algorithm on the dataset. To do that we initially computed the distances between every pair of cluster centres and ran the minimum-weight perfect matcher on that. We got the following result.

The objective function was later changed to maximum-weight perfect matching based on the number of shared points between clusters. Here, a very interesting property was observed. Most of the points were perfectly matched while the rest were

Chapter 3

Extendable C++ Implementation

Throughout the research for this paper, it was difficult to find modular code that is easily extendable for the K-Means algorithm that allow us to perform testing on it. For that, we created our own implementation that is available on Github Alkayale (2016). This implementation is free to use and licenced under the MIT License, allow users to modify it as they see fit without any warranty on functionality or fitness for a particular purpose.

The algorithm is implemented in highly optimized C++ with object oriented programming at its core. This allows other developers to extend the algorithm and test different combinations of optimizations in the code. It can be used by both individuals needing to use a K-Means implementation and ones who are looking to do research and optimize it further.

Every K-Means instance, contains templated instances of a SeedPicker and an IterationRunner along with a skeleton that allows analysis of the performance of each run of the algorithm. This allows us to use different IterationRunners and SeedPickers interchangeably in the algorithm depending on the algorithm at hand and the resources allowed. The analytical data being saved at each run is independent of the SeedPicker and IterationRunner, which makes it simple for developers wanting to extend the algorithm.

The modularity of the implementation also allows developers to create new SeedPicker and IterationRunner classes. To create one, all they are required to do is implement a function from their respective Base class. For example, the *kd*-tree implementation of the algorithm has be hooked into a K-Means instance by implementing it as an IterationRunner.

Chapter 4

Conclusion

K-MEANS is a problem that has proven to be applicable to many domains and applications. It is still widely used and continues to be so. We have shown that the new K-MEANSConsensus method of seeding the K-MEANS algorithm is no better than randomly seeding the algorithm.

One area of research that is needed is a way to compute the real number k for the algorithm. We have a method to compute that by calculating cliques in perfectly matched centres over multiple runs of the algorithm, but this needs further exploration.

Bibliography

Alkayale, Y. (2016). Kmeans.

Alsabti, K., Ranka, S., & Singh, V. (1997). An efficient k-means clustering algorithm. *Electrical Engineering and Computer Science.*, *43*.

Arthur, D. & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, (pp. 1027–1035). Society for Industrial and Applied Mathematics.

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., & Vassilvitskii, S. (2012). Scalable k-means++. *Proceedings of the VLDB Endowment*, *5*(7), 622–633.

Berkhin, P. (2002). Survey of clustering data mining techniques. Technical report.

Collard, P. (1989). Cloud data set.

Corpet, F. (1988). Multiple sequence alignment with hierarchical clustering. *Nucleic acids research*, *16*(22), 10881–10890.

Edgar, R. C. (2010). Search and clustering orders of magnitude faster than blast. *Bioinformatics*, *26*(19), 2460–2461.

Epasto, A., Lattanzi, S., & Paes Leme, R. (2017). Ego-splitting framework: from non-overlapping to overlapping clusters. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 145–154). ACM.

Forman, G. (1999). Spambase data set.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern recognition letters*, *31*(8), 651–666.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, *28*(2), 129–137.

Mahajan, M., Nimbhorkar, P., & Varadarajan, K. (2009). The planar k-means problem is np-hard. In *International Workshop on Algorithms and Computation*, (pp. 274–285). Springer.

Maio, D., Maltoni, D., & Rizzi, S. (1996). Dynamic clustering of maps in autonomous agents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *18*(11), 1080–1091.

Ravichandran, D., Pantel, P., & Hovy, E. (2005). Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, (pp. 622–629). Association for Computational Linguistics.

Schaeffer, S. E. (2007). Graph clustering. *Computer science review*, *1*(1), 27–64.

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 815–823).

Soto, A. J., Kiros, R., Keselj, V., & Milios, E. (2016). Machine learning meets visualization for extracting insights from text data. *AI Matters*, *2*(2), 15–17.