K-MEANSCLUSTERING, AN IMPROVED SEEDING PROCESS AND NOVEL METHOD TO COMPUTE THE NUMBER OF CLUSTERS FOR A DATASET

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 . . . . **iv**

Acknowledgements . . . . **v**

Chapter 1 Introduction & Background **1**

1.1 The K-MEANSalgorithm. 2

1.2 Seeding Process 3

1.2.1 K-MEANS++: A Better Seeding Process 3

1.3 Iterations 4

1.4 Objective Function 4

1.5 Difficulties 5

Chapter 2 Consensus Seeding **6**

2.1 The Algorithm Pseudocode 6

2.2 Analysis & Datasets 7

2.3 Minimum Weight Perfect Matching on Bipartite Graphs 8

2.4 Choosing the Correct Matching Function 10

Chapter 3 Determining the Number of Clusters *k* **11**

Chapter 4 Extendable C++ Implementation **13**

Chapter 5 Conclusion **14**

Bibliography **15**

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 as a reasonably fast heuristic to find clusters. K-Meanshas 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 moves the centres around until it converges to a local optimum. The iteration process dominates the running time. In this paper, we discuss a new method of the seeding process that promises to give us more accurate seeds to start the algorithm and a novel way to extract the underlying number of clusters in a given dataset to help K-Meansconverge faster and give more accurate results.

Acknowledgements

A sincere thank you to my supervisor Dr. Norbert Zeh. Without his assistance, this project would not have seen the light light of day. Thank you to my co-supervisor Dr. Vlado Keselj who made himself available. 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 criterion. Clustering problems arise in many domains such as 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. Many clustering techniques like hierarchical clustering (Corpet, 1988) and graph-based clustering (Schaeffer, 2007) exist. K-Meansclustering 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 such as 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 the domain it is being used in and the problem at hand.

Formally, K-Meansis 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 (subsets), minimizing the cost function ϕ, the sum of distances of each point to its cluster centre. A cluster centre is defined to be the mean of the points in the clusters. K-Meansdoes especially well, in terms of speed and accuracy, on convex shaped clusters as it minimizes the sum of distances of all points to their corresponding 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-Meansexactly 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. A locally optimal solution is good enough in most cases, allowing us to have meaningful clusters in a reasonable amount of time.

In this paper, we introduce a new way to seed the K-MEANSalgorithm. Preliminary experiments suggested that htis seeding method produced clusters of comparable quality as produced by advanced seeding strategies such as K-MEANS++ and K-MEANS||, while bing much simpler. Unfortunately, a more careful evaluation shawed that the clusters are not significantly better than those obtained using uniformly random seeds. The ideas of the algorithm and its pseudocode are outlined along with results. We also provide a C++ implementation of the algorithm and ways it can be used for further development and research.

1.1 The K-Meansalgorithm.

The K-MEANSalgorithm 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). The input is assumed to be the dataset X and the number of clusters required k:

1. Pick k arbitrary points 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 a new cluster centre $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$
4. Repeat steps 2 and 3 until centres do not change[1].

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 or a limit on the number of iterations is reached.

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

1.2 Seeding Process

Lloyd's algorithm, sketched above, finds a locally optimal set of clusters but may fail fo find the globally optimal solution. The objective function of these two solutions may defer significantly. Hence, the choice of of the K-Meansalgorithm is crucial to the accuracy of the output. 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, and to better local optima. Given the dataset X and the number $k \leq |X|$ to cluster, K-Meanspicks k points from the dataset uniformly at random. Lloyd's 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-Meansis to do it uniformly at random. The objective functions of the clusters computed using uniformly random seeds generally have hight variance. More robust results are obtained using more sophisticated seeding strategies. It it important to note here that mo matter the choice of seeds we are only able to compute a local optimum, not the global optimum solution because the problem is NP-hard (Mahajan et al., 2009).

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 give a higher probability of picking a point proportional to its distance from the seeds already picked, until k seeds are picked (Arthur & 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-Meansproblem, and as discussed earlier, that is NP-hard.

From my research, K-Means++ seems to be the best scientifically proven way to seed the K-Meansalgorithm. However, there 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-Meansclustering algorithm dominates the running time of the algorithm. The running time for random seeding is $O(k)$, while the iterations run in $O(tnkd)$ time, where t is the number of iterations.[2] Depending on how quickly the iterations converge, the number of iterations may be large. K-Means++ has a seeding cost of $O(nk^2d)$, but the cost of the iterations still dominates the running time of the algorithm for small k. 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, determines the set of clusters computed. The most common choice, also used in paper, is the sum of the squared Euclidean distances[3] of all the points to their cluster centres. More formally expressed as:

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

[2] Faster implementations of these iterations are possible using efficient data structures, but this is not the focus of this thesis.

[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.

This objective function is the reason why K-Meansinherently 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-Meansclustering is very good at clustering well-separated convex clusters where k the number of clusters is known or accurately measured in advance. However, this is very rarely the case. Another issue with K-Meansis 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

Consensus Seeding

Consensus seeding is a new seeding process for K-Means. The basic idea is to run the randomized seeding process T times, followed by a small number of iterations, S, which produce a set of k cluster centres each. We then take these $(T * K)$ centres and cluster them into k clusters. The resulting cluster centres are the seeds for the iteration phase of the entire dataset.

2.1 The Algorithm Pseudocode

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

Algorithm 2.1: Consensus Seeding.

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

We intuitively believed that consensus seeding would perform better than random seeding of the K-Meansalgorithm because of rerunning the algorithm many times. It has been experimentally shown that running K-Meansmultiple times and taking the best one of the results yields a better output (Arthur & Vassilvitskii, 2007). Our idea was to terminate each run of the algorithm early to save on running time while using the information to better seed the algorithm. We expected this would give us similar results to K-Means++

2.2 Analysis & Datasets

To analyze the performance of the algorithm we ran it on 3 different datasets. The Cloud dataset has 1025 values in 10 dimensions, and they represent the values for cloud cover, data extracted from pictures of clouds represented by their pixel values (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 K-Means, K-Means++, and K-MeansConsensus on these three datasets. 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 (milliseconds)		
k	kmeans	kmeans++	kmeansNew	kmeans	kmeans++	kmeansNew	kmeans	kmeans++	kmeansNew
10	7.62e06	6.12e06	8.73e06	6.29e06	5.76e06	6.29e06	4.90e00	3.40e00	6.51e00
25	3.81e06	2.21e06	3.63e06	2.62e06	2.06e06	2.45e06	8.08e00	9.17e00	1.13e01
50	2.03e06	1.17e06	2.10e06	1.51e06	1.10e06	1.54e06	1.08e01	2.03e01	1.86e01
100	1.06e06	6.55e05	1.16e06	8.15e05	6.30e05	8.73e05	1.07e01	6.08e01	3.15e01

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

	spam								
	Average ϕ			Minimum ϕ			Average T (milliseconds)		
k	kmeans	kmeans++	kmeansNew	kmeans	kmeans++	kmeansNew	kmeans	kmeans++	kmeansNew
10	1.70e08	8.67e07	1.70e08	1.70e08	7.70e07	1.70e08	2.67e02	7.32e01	2.98e02
25	1.51e08	1.70e07	1.52e08	1.43e08	1.56e07	1.44e08	8.68e02	2.80e02	1.08e03
50	1.49e08	6.76e06	1.40e08	1.26e08	6.06e06	6.51e07	1.56e03	7.49e02	1.94e03
100	1.27e08	2.41e06	1.18e08	6.66e07	2.20e06	1.99e07	1.84e12	2.17e03	1.55e03

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

	dimred								
	Average ϕ			Minimum ϕ			Average T (milliseconds)		
k	kmeans	kmeans++	kmeansNew	kmeans	kmeans++	kmeansNew	kmeans	kmeans++	kmeansNew
10	7.02e06	7.03e06	7.02e06	6.99e06	6.99e06	6.99e06	4.38e02	5.60e02	6.24e02
25	2.79e06	2.79e06	2.79e06	2.77e06	2.77e06	2.77e06	1.31e03	1.43e03	1.44e03
50	1.40e06	1.39e06	1.40e06	1.38e06	1.38e06	1.38e06	1.66e13	1.38e13	1.38e13
100	7.08e05	7.04e05	7.07e05	7.00e05	7.01e05	6.99e05	8.30e12	2.58e13	6.46e12

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

We can clearly see that the new seeding process gave us marginally better results than K-Meansin 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.

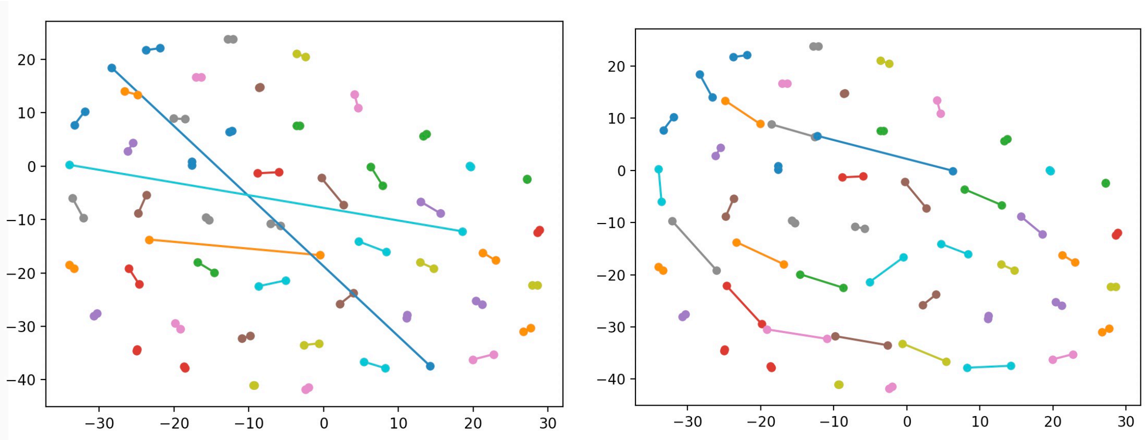
It is important to note here that on the dimred dataset, unlike the other datasets, all three algorithms achieve similar results.

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 2.1: 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 2.1a & 2.1b 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 2.1a 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

Determining the Number of Clusters k

From our experiments and research, it was clear that inputting the incorrect number of clusters k into the K-MEANSalgorithm, substantially affects performance, and accuracy. For that, we came up with an algorithm to help approximate the right number of clusters in the underlying dataset, before running the K-MEANSalgorithm.

The idea of the algorithm is relatively simple relying on the assumption that the real cluster centres of the underlying dataset will be present in most random runs of the algorithm, while the rest of the cluster centres will be relatively random based on the local area obtained through the initial seeding process.

The algorithm takes a number which is the maximum number of clusters k believed to be for the dataset, and the dataset. K-MEANS++ is run with the dataset k times, and the centres produced are recorded. Then, we perfectly match centres from every unique pair of runs. Based on the matchings, we build a graph where every cluster centre in every run is a node that is connected to all the other nodes it was connected to by the matchings. Then we extract the number of disconnected components in the graph, and that tells us the estimated underlying number of clusters in the dataset. Because it is an estimate, running the above algorithm multiple times and taking the most frequent number is a heuristic that we found worked well.

The reason we believe this method works is because cluster centres that are accurate to the underlying dataset will be consistent in all the runs even if the inputted k is not accurate. However, if the inputted k is higher, then we will consistently get cluster centres that are somewhat random depending on how the algorithm was seeded. For that, in our experimental tests, we consistently had components of size k, while others had larger sizes. We believe that the components of size k are the ones that are true for the underlying dataset.

Based on our experiments on a generated dataset, shown in Figure **??**, with 4 clusters and $5,000$ per cluster, we were able to consistently get results between 3 and

5 for maxK input of 4 to 9. At maxK value of 10, we got consistent results of 1. This is a bit surprising, but makes sense because we can see how with a number k that is higher than the real one in the underlying dataset will have a high probability of cluster centres matching to other components, grouping them all into one by the end.

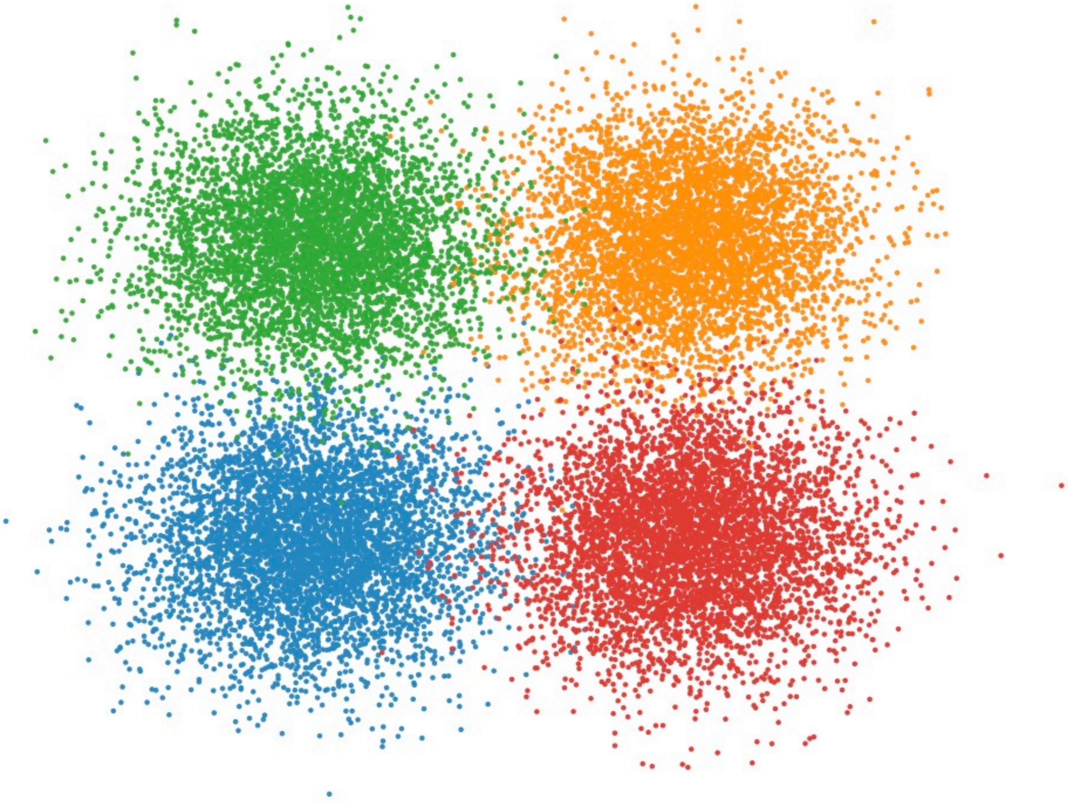


Figure 3.1: Heat map of the git brDimred dataset. The lighter versions of the figure are for

Chapter 4

Extendable C++ Implementation

Throughout the research for this paper, it was difficult to find modular code that is easily extendable for the K-Meansalgorithm that allow us to perform testing on it. For that, we created our own implementation that is available on Github [1]. 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-Meansimplementation and ones who are looking to do research and optimize it further.

Every K-Meansinstance, 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-Meansinstance by implementing it as an IterationRunner.

[1] The code is available at https://github.com/technoligest/kmeansII

Chapter 5

Conclusion

K-Meansis a method 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-Meansalgorithm 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

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.

Collard, P. (1989). Cloud data set. https://archive.ics.uci.edu/ml/datasets/Cloud.

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. https://archive.ics.uci.edu/ml/datasets/spambase.

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 of the 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.