Technology Review: K-Means, kernel K-means, K-Medians, K-Medoids

Billy Li
zl20@illinois.edu
Department of Computer Science

October 24, 2021

1 Introduction

K-Means, kernel K-Means, K-Medians, and K-Medoids are four closely related cluster analysis algorithms for spatial data, aiming to partition a set of observations represented as data points in an $n-$dimensional data space into a predefined number of clusters. This technology review will compare and contrast the 4 methods and discuss the strengths and shortcomings of each method via evaluating their performance on datasets with different properties.

2 Overview

K-means is defined as follows: for clustering n data points in an $m-$dimensional space into k desired clusters, the algorithm can be described as follows:

1. Randomly select k points in in the data space as initial centers.
2. Iterate until convergence:
3. **Assign** Assign each of the n data points to the closest center using L_m norm as the distance measure.
4. **Update** the centers of each of the k clusters to minimize the total distance between the data points in each data cluster and its center.

Each of the remaining 3 methods are based off of K-Means and therefore follow a similar control flow. Specifically, kernel K-Means replaces the L_m norm with a kernel function, K-Medians replaces the L_m norm with the L_1 norm (Manhattan Distance), and K-Medoids uses actual data points as the cluster centers instead of arbitrary points in the data space.

The basic K-Means algorithm has several shortcomings, and we will briefly introduce what each method attempts to rectify:

- **Kernel K-Means**: K-Means fails when clusters are not linearly separable in the native $n-$dimension space. A kernel is a function which computes the distance between two data points $K(x_i, y_i) = \phi(x_i)^T \phi(y_i)$ mapped to a higher $m-$dimensional space, $m > n$ via dot product without explicitly specifying or computing the n to m-dimension mapping ϕ[1]; Kernel K-Means can successfully identify said clusters through choosing an appropriate kernel which maps the clusters to a higher dimensional space where they are linearly separable.

- **K-Medians**: K-Medians handles certain noisy datasets better than K-Means due to the L_1 norm being less affected by outliers in the data compared to the L_p norm, similarly to how skewed data is better represented by the median compared to the mean[4].

- **K-Medoids**: By using actual data points as medoids, K-Medoids can be used with arbitrary distance measures instead of being required to use an L_p norm for an efficient solution [5]. The ability to use a custom distance measure can be crucial in separating irregular clusters as seen in Figure 3: to the best of our knowledge, no L_p norm (or kernel that is not overly complex if using Kernel K-Means) can successfully separate the depicted clusters. However, K-Medoids can be used to separate the clusters using the connectivity metric:

$$d(x, y) = \begin{cases} 0 & \exists \text{ path from } x \text{ to } y \text{ in } G \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

  Where G is the K-nearest neighbor graph constructed from the data points.s

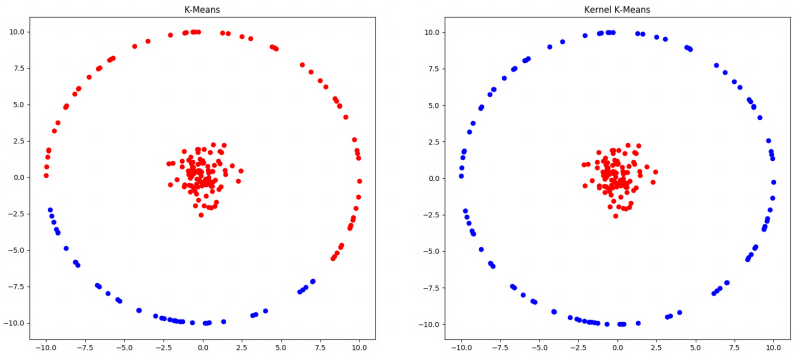

Figure 1: 2 clusters (2D Gaussian, circle) which are not linearly separable in 2-dimensional space. Kernel K-Means is able to distinguish the two clusters using the polynomial kernel $K(x, y) = (x^T y)^2$ while K-Means is unable to. Example taken from [2].

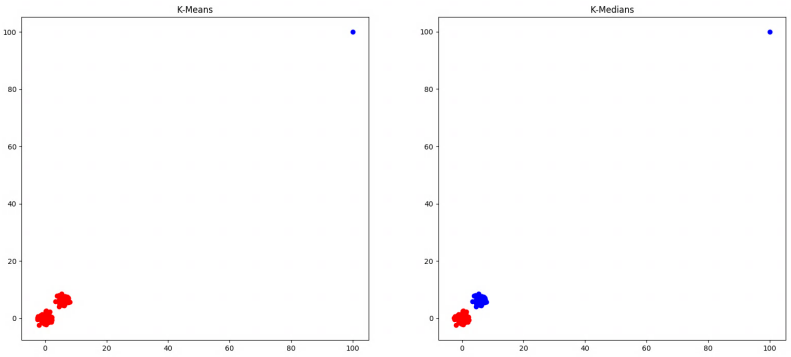

Figure 2: 2 2D Gaussian clusters and a noise data point. K-Medians is able to distinguish the clusters despite the noise data point while K-Means is unable to. K-Medians code adapted from [3].

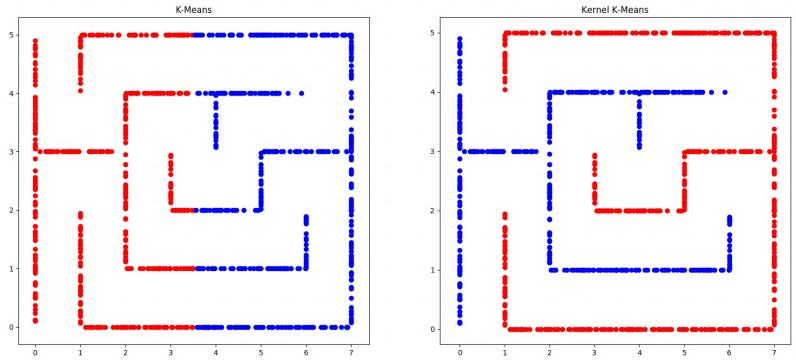

Figure 3: 2 clusters defined as separate halves of a maze. K-Medoids is able to distinguish the clusters using a custom metric while K-Means is unable to.

Table 1: A comparison between clustering methods

Method	Distance metric	Centers
K-Means	L_p norm	Any points
kernel K-Means	Kernel function	Any points
k-Medians	L_1 norm (Manhattan distance)	Any points
k-Medoids	Any metric	Data points

3 Experimentation

3.1 Overview

We will now examine the effectiveness of the clustering algorithms on large-scale datasets for testing clustering algorithms[6]. We will be using the following implementations for each algorithm:

- The sklearn library in Python has an implementation of the K-Means algorithm [7].
- To the best of our knowledge, there is currently no library supporting kernel K-Means; instead, we implemented kernel K-Means by (1) explicitly computing the mapping for n to m dimensions and (2) computing K-Means in the higher m dimension. However, due to the difficulty of computing mappings for kernel functions in high dimensions, kernel K-Means will be excluded from experimentation.
- To the best of our knowledge, there is currently no library supporting K-Medians. The closest we can find is the sklearn-extension implementation of Fuzzy K-Medians [8]. We implemented our own K-Medians algorithm adapted from [3].
- The sklearn library has an implementation of the K-Medoids algorithm [9].

For each algorithm, we will also investigate the effectiveness of the following input parameters:

- K-Means, K-Medians: Random centroid initialization, K-Means++
- K-Medoids: Random, 'heuristic', K-Medoids++, 'build' initialization
- K-Medoids: L_p norm, Polynomial kernel $k(x, y) = (x^T y)^p$, Gaussian kernel $k(x, y) = exp(-\|x - y\|^2)$
- K-Medians: Maximum number of iterations (10, 50).

Where K-Means++ is a method of initializing centers by randomly selecting the first center, then selecting subsequent centers with probability inversely proportional to the distance to the closest existing center[10], K-Medoids++ is based on K-Means++, 'heuristic' picks the data points with the smallest sum distance to other data points as the initial medoids, and 'build' is the initialization method proposed in the original K-Medoids paper[11]. our implementation of K-Medians uses random centroid initialization.

We will evaluate the clustering quality via Normalised Mutual Information (NMI), which is the harmonic mean of the Homogenity score - How similar are items in the same cluster and the Completeness score - How much similar items are clustered together[12]. For each dataset, we run each algorithm 10 times to obtain an average clustering quality an15d the average amount of time it takes to compute the clusters.

The selected datasets provide distinct environments for evaluating the clustering algorithms; A1 is a standard dataset with a large number Gaussian clusters, S1 introduces clusters with different shapes, G2 features 2 large overlapping clusters, and dim032 has a high dimension data space[6].

Table 2: Summary of datasets used in experimentation from [6]

Dataset	Number of points (N)	Number of clusters (k)	Dimension (d)
S1	5000	15	2
A1	3000	20	2
G2	2048	2	2
dim032	1024	16	32

3.2 Results

The results are shown in Table 3. In terms of clustering quality:

- In the initialization methods for K-Means, if clustering quality is the concern, K-Means++ should always be chosen over Random initialization. K-Means++ also provides the best clustering quality across all 4 datasets, especially on the large-scale S1 and A1 datasets where no other algorithm ties with it.

- the Build method has the best performance out of the initialization methods for K-Medoids. It is however stated to be very non-robust in the presence of outliers, which is fortunately not the case as all 4 datasets have no noise due to being manually generated. On the other hand, the heuristic method is the worst performer; this is expected as its greedy initialization results in the initial centroids being close to each other in the center of the dataset, resulting in the algorithm converging to low-quality local maxima.

- If using a custom distance metric with K-Medoids, it is important to choose a metric that matches the characteristics of the dataset. This is clearly not the case with the Polynomial and Gaussian kernels used in the experiments as evident in their poor performance. In particular, the usage of the Gaussian kernel often results all points being assigned to 1 cluster in all 4 datasets.

- It can be seen that that the K-Medians method performs worse compared to the K-Means and K-Medoids methods when the data is noise-free. Furthermore, the L_1 norm is arguably not a suitable metric as the shape of the boundary it assigns to

clusters is inconsistent with the actual spherical shapes due to the clusters being generated from Gaussian distributions.

In terms of the algorithm runtimes:

- For the K-Means algorithm, the random initialization method tends to converge in fewer iterations compared to K-Means++ and likely converges to a local maximum, missing out on a better solution. The opposite is true for the K-Medoids algorithm; additionally, the Heuristic method tends to be the second slowest, while the Build method is the slowest, verifying the claim that it is 'slow on large datasets' in its documentation.
- Between K-Means and K-Medoids, K-Means is in general faster than K-Medoids. However, compared to K-Means, the runtime of K-Medoids scales quadratically on the largest cluster size as it needs to evaluate pairwise distances[11], resulting it being slower than K-Means on datasets where large clusters are present such as the G2 dataset.
- As expected, the chosen distance function has a large impact on the runtime on K-Medoids, and for both the Polynomial and Gaussian kernels the time it takes to compute the distance matrix dominates the time to perform the actual clustering.
- Our own implementation of K-Medians is very inefficient compared to the established implementations of K-Means and K-Medoids in the sklearn library as expected.

Table 3: Results from running the different clustering algorithms on the 4 datasets

S1 dataset			**A1 dataset**		
Method	Time (s)	NMI	Method	Time (s)	NMI
K-Means Random	1.3760	0.9603	K-Means Random	1.3910	0.9606
K-Means++	2.2347	**0.9867**	K-Means++	0.7466	**0.9975**
K-Medians 10 iters	7.6406	0.8841	K-Medians 10 iters	3.7630	0.7766
K-Medians 50 iters	42.3062	0.9209	K-Medians 50 iters	21.4077	0.7837
K-Medoids Random	0.7657	0.9049	K-Medoids Random	0.3507	0.9230
K-Medoids++	**0.6578**	0.9668	K-Medoids++	**0.3286**	0.9758
K-Medoids Heuristic	0.8802	0.9261	K-Medoids Heuristic	0.5317	0.8415
K-Medoids Build	1.5034	0.9856	K-Medoids Build	0.7427	0.9975
K-Medoids Polynomial	179.7229	0.3168	K-Medoids Polynomial	48.6812	0.3237
K-Medoids Gaussian	123.4710	0.0003	K-Medoids Gaussian	44.9754	0.0007
G2 dataset			**dim032 dataset**		
Method	Time (s)	NMI	Method	Time (s)	NMI
K-Means Random	**0.1602**	**1.0000**	K-Means Random	0.1269	0.9711
K-Means++	0.2943	**1.0000**	K-Means++	1.9952	**1.0000**
K-Medians 10 iters	0.5876	0.0000	K-Medians 10 iters	5.7947	0.8533
K-Medians 50 iters	5.6248	0.1000	K-Medians 50 iters	107.1507	0.8685
K-Medoids Random	0.2756	**1.0000**	K-Medoids Random	**0.0469**	0.9140
K-Medoids++	0.2497	**1.0000**	K-Medoids++	0.0601	**1.0000**
K-Medoids Heuristic	0.2259	**1.0000**	K-Medoids Heuristic	0.1076	0.8248
K-Medoids Build	0.2691	**1.0000**	K-Medoids Build	0.2176	**1.0000**
K-Medoids Polynomial	34.5654	0.0049	K-Medoids Polynomial	5.7923	0.3935
K-Medoids Gaussian	42.8348	0.0548	K-Medoids Gaussian	6.2445	0.0000

To summarize our findings, we propose the following as the desirable situations to use each algorithm in:

- Efficiency is the main concern and an approximate clustering is sufficient for the task:
  - Large amount of small clusters: **K-Medoids++**
  - Small amount of large clusters: **K-Means Random**
- Classification quality is the main concern:
  - Time is not a concern, want best results: **k-Means++**
  - Want results in a reasonable amount of time: **K-Medoids Build**
- Other methods (Author's K-Medians, K-Medoids Heuristic/Random) are generally not recommended for usage.

4 Summary

In this technology review, we compare and contrast the clustering methods of K-Means, Kernel K-Means, K-Medians and K-Medoids. We use examples to provide a brief overview on the characteristics of data where each algorithm likely succeeds or fails. We also conduct experiments using large clustering test datasets to investigate the effectiveness and efficiency of each method in more realistic settings. We summarize our findings and provide recommendations for choosing the method to use based on the characteristics of the task at hand.

References

[1] Dhillon, Inderjit S., Yuqiang Guan, and Brian Kulis. "Kernel k-means: spectral clustering and normalized cuts." Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. 2004.

[2] https://stats.stackexchange.com/questions/152897/how-to-intuitively-explain-what-a-kernel-is

[3] https://medium.com/@pasdan/k-clustering-means-medians-via-python-2a5f251582ee

[4] https://www.quora.com/What-is-key-difference-two-kmeans-variance-K-center-clustering-and-K-medians-clustering

[5] https://en.wikipedia.org/wiki/K-medoids

[6] http://cs.joensuu.fi/sipu/datasets/

[7] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[8] http://wdm0006.github.io/sklearn-extensions/fuzzy_k_means.html

[9] https://scikit-learn-extra.readthedocs.io/en/stable/generated/sklearn_extra.cluster.KMedoids.html

[10] Arthur, D., & Vassilvitskii, S. (2006). k-means++: The advantages of careful seeding. Stanford.

[11] Kaufman, L., & Rousseeuw, P. J. (1990). Partitioning around medoids (program pam). Finding groups in data: an introduction to cluster analysis, 344, 68-125.

[12] https://towardsdatascience.com/v-measure-an-homogeneous-and-complete-clustering-ab5b1823d0ad