Fast K-means for Large Scale Clustering

Qinghao Hu[1,2], Jiaxiang Wu[1,2], Lu Bai[4], Yifan Zhang[1,2], Jian Cheng[1,2,3*]
[1]Institute of Automation, Chinese Academy of Sciences , Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]Center for Excellence in Brain Science and Intelligence Technology, CAS, Beijing, China
[4]School of Information, Central University of Finance and Economics, Beijing, China
{ qinghao.hu, jiaxiang.wu, yfzhang, jcheng}@nlpr.ia.ac.cn, bailucs@cufe.edu.cn

ABSTRACT

K-means algorithm has been widely used in machine learning and data mining due to its simplicity and good performance. However, the standard *k*-means algorithm would be quite slow for clustering millions of data into thousands of or even tens of thousands of clusters. In this paper, we propose a fast *k*-means algorithm named multi-stage *k*-means (MKM) which uses a multi-stage filtering approach. The multi-stage filtering approach greatly accelerates the *k*-means algorithm via a coarse-to-fine search strategy. To further speed up the algorithm, hashing is introduced to accelerate the *assignment* step which is the most time-consuming part in *k*-means. Extensive experiments on several massive datasets show that the proposed algorithm can obtain up to 600× speed-up over the *k*-means algorithm with comparable accuracy.

CCS CONCEPTS

• Theory of computation → Unsupervised learning and clustering; **Bloom filters and hashing**; **• Information systems → Nearest-neighbor search**;

KEYWORDS

K-means; Clustering; Hashing

1 INTRODUCTION

The *k*-means problem is to cluster *N* data points into *K* clusters by finding *K* centroids to mininize the sum of squared distances between data points and their assigned centroids. However, the *k*-means problem has been proven to be NP-hard. Lloyd's algorithm is the simplest and most popular algorithm providing approximate solutions to the *k*-means problem. It has been widely used in machine learning and data mining for various tasks during the past decades.

Lloyd's algorithm, also referred as the standard *k*-means algorithm, starts with *K* arbitrary centroids. Each point is then assigned to its nearest centroid in the *assignment* step, followed by the *update* step, where each centroid is updated with the average of data points assigned to it. Lloyd's algorithm repeats these two steps until reaching the maximum number of iterations or some criterion is met. In general, the standard *k*-means becomes slower with *N* and *K* increasing, where *N* and *K* denote the number of samples and the number of clusters respectively. The time complexity of assignment step is $O(NKD)$ while it is $O(ND)$ for the update step. The standard *k*-means becomes prohibitively expensive when *N* and *K* are very large. Therefore, how to speed up the *k*-means has become a challenge when dealing with large *N* and *K*. A lot of approaches have been proposed to accelerate the standard *k*-means. These approaches can be categorized into two classes. One is called accelerated exact *k*-means [4, 5, 7], the other one is called approximate *k*-means [1, 11, 12]. Accelerated exact *k*-means algorithms achieve the same result as the standard *k*-means. Elkan *et al.* [5] maintains lower bounds between all *N* samples and *K* centroids, and unnecessary distance calculations are avoided using these bounds. However, these lower bounds requires extra $O(NK)$ memory, which is unaffordable for large *N* and *K*. Hamerly's method [7] reduces the *NK* lower bounds of [5] to *N* lower bounds but only performs well in low dimensional space. Ding *et al.* [4] made a compromise between [5] and [7]. Approximate *k*-means algorithms find the approximate solutions compared with standard *k*-means. Philbin *et al.* [11] proposed an approximate *k*-means (AKM) which uses a forest of randomized k-d trees to speed up the *assignment* step. A mini-batch sampling method is also used in speeding up *k*-means in [12]. This algorithm becomes slow for large *K*. Avrithis *et al.* [1] proposed the inverted-quantized *k*-means (IQ-means). IQ-means requires a learning step which costs a lot of time. Here we focus on the approximate *k*-means. In this paper, we propose a fast *k*-means algorithm named multi-stage *k*-means (MKM) which uses a multi-stage filtering approach. The multi-stage filtering approach reduces computational complexity greatly via a coarse-to-fine search strategy. Since the *assignment* step is the most time-consuming part in *k*-means, hashing algorithm is introduced to speed up this step. By using hashing, the distances between samples and centroids can be calculated very fast via XOR and *__popcnt* instruction as provided in most of modern CPUs. Experiments on three publicly available datasets show that the proposed MKM algorithm can achieve up to 600× speedup over the standard *k*-means with comparable accuracy.

2 OUR APPROACH

In this section, we give details of our proposed algorithm. In the first stage, we use sequential mini-batch sampling to select a subset of samples. By using group filtering in the second stage, centroids

CIKM'17 , November 6–10, 2017, Singapore, Singapore

ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3133091

*The corresponding author.

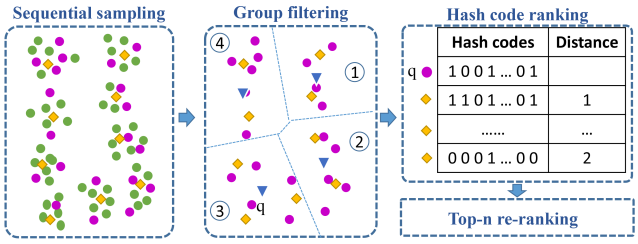


Figure 1: The framework of our proposed multi-stage k-means. Samples are denoted by green points. A mini-batch of samples (purple points) are selected using sequential mini-batch sampling. The centroids (yellow diamond) are divided into M groups with k-means clustering. The triangles denote the group centroids. For each selected sample (purple point q), centroids in the m-nearest groups (group 2 and 3 as for q) are selected as the candidates. Top-n candidates are then chosen with respect to the Hamming distance, which are further re-ranked by the exact Euclidean distance. Best viewed in color.

far way from query samples are filtered out. Then a hash code ranking is used for choosing the top-n ($n \ll K$) centroids candidates in the third stage. Exact Euclidean distance between these candidates and the selected samples are calculated for searching the nearest centroid in the last stage. The overall workflow of our proposed algorithm is depicted in Fig.1.

2.1 Sequential Mini-batch Sampling

In [12], Sculley proposed the mini-batch k-means which uses only a subset of samples in each iteration to speed up the *assignment* step. A mini-batch samples are usually selected by uniform random sampling. Selecting samples at random may cause that some samples are never visited and some samples are visited more than the average frequency. To solve this problem, we use sequential mini-batch sampling method. All samples are shuffled once in the beginning and divided into multiple small mini-batches. Then a mini-batch is selected sequentially in each iteration. This sampling scheme is often used in training neural networks, but not mini-batch k-means. By using sequential mini-batch sampling, the number of unique samples used for k-means grows linearly with the number of iterations until reaching the size of dataset.

Theorem 2.1. *Given dataset X which contains N samples, the expectation of unique samples' amount U after sampling with replacement P times from X is:*

$$E(U) = N\left(1 - \left(\frac{N-1}{N}\right)^P\right) \tag{1}$$

From Theorem 2.1, we know that uniform random sampling always visits less or equal number of unique samples than sequential mini-batch sampling.

2.2 Group Filtering

After sequential mini-batch sampling, the nearest centroid search is only required for samples within the mini-batch. We design a coarse-to-fine strategy to speed up this step. A coarse search is achieved by centroids grouping. We divide the centroids into M groups in the beginning of algorithm via several iterations of k-means on the centroids. We denote the centroid of each group as the group centroid. By computing the Euclidean distance between each sample and group centroids, we can get the m-nearest groups for each sample. Then centroids in these m groups are then selected as candidates. This procedure is called group filtering. This step filters out centroids which are far way from query sample. In general, different combinations of M and m can be used to balance the efficiency-recall trade-off. In this paper, we cluster the centroids into 128 groups and centroids from top-4 nearest groups are selected as candidates.

In the *update* step, centroids are updated which may invalidate the current centroid grouping in the next iteration. To solve this, we run one iteration of k-means on centroids after the *update* step. Such update of the centroid grouping costs little time, which is totally affordable.

2.3 Hash Code Ranking

After group filtering, only a small set of candidates are selected. But it is still computational expensive to conduct exhaustive search on all candidates in the Euclidean space. Instead, calculating hamming distance between samples and centroid candidates can be much faster. Here samples and centroids are mapped to the Hamming space via locality sensitive hashing (LSH) functions. LSH has been well studied [2] and used for many applications [3, 10]. We don't use a learning-based hashing function because the training of hashing function usually costs a lot of time. Different LSH families can be used for different distance measure functions. For the choice of LSH functions, we choose the LSH function proposed by Charikar in [2]. Although this hash function is designed to preserve the cosine distance by using random projection, it also performs well under the Euclidean distance measurement.

By using LSH functions, candidates from the second stage will be mapped to an L-dimensional Hamming space, then the Hamming distance between samples and centroid candidates are computed based on L-bits hash codes. The length of hash bits L influences the speed and precision of hash code ranking. For the search efficiency and precision, We choose L=256 in the experiments.

2.4 Top-n Re-ranking

By hash code ranking, top-n ($n \ll K$) candidates are selected. An exhaustive search is then be conducted in Euclidean space. Each query sample is assigned to its nearest centroid among the n candidates. We call this procedure as Top-n re-ranking.

3 EXPERIMENTS

3.1 Datasets and Expriments Settings

Experiments are mainly conducted on three publicly available datasets i.e. SIFT1M, GIST1M, and ILSVRC2012. For ILSVRC2012 dataset, we extract 4096-dimensional feature vector for each image from the $fc7$ layer of Alexnet [9]. By using Principal Component Analysis (PCA), these 4096-dimensional features are reduced to 128 dimensions.

We implement our algorithm in C++ language and use BLAS library for matrix operation. We compare our proposed method with standard k-means, AKM, Mini-batch k-means and IQ-means.

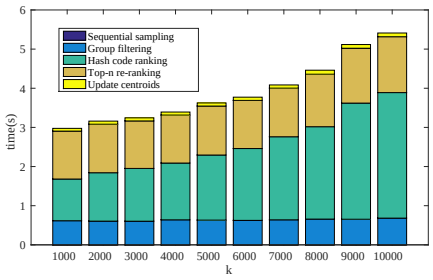


Figure 2: runtime of each stage in our algorithm

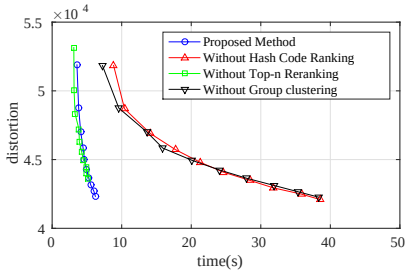


Figure 3: average distortion vs time on SIFT1M

Yael[1] library is used for the standard k-means clustering. For IQ-means[1], we use the code provided by the author. We carefully implemented Approximate k-means (AKM) [11] and Mini-batch k-means [12] in C++ language. For all comparison methods, we use the same initial centroids which are picked at random. All experiments are conducted on a desktop PC running Ubuntu 14.04 OS with i7-4790k CPU and 32GB run-time memory.

3.2 Evaluation Metric

For SIFT1M and GIST1M dataset, we use average distortion to evaluate the proposed algorithms. Let $X = \{x_1, x_2, \cdots x_N\}$ be the dataset. These N points are partitioned to k clusters $C = \{C_1, C_2, \cdots, C_k\}$. The average distortion is defined as follows:

$$J = \frac{1}{N} \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - c_j\|_2^2 \tag{2}$$

where c_j is the centroid of cluster C_j.

For ILSVRC2012 dataset, we use mean purity to evaluate the proposed method following [6]. Given a dataset with c classes, each cluster is assigned to the most frequent class. Then purity on an cluster is defined as the proportion of correctly assigned samples in this cluster. Then mean purity is computed by averaging the purities for all clusters.

3.3 Experiments Result

Time Analysis To better analyse the multi-stage k-means, we conduct experiments to show the time of each stage. Fig. 2 reports the time of each stage in multi-stage k-means under varying k from 1000 to 10000 on SIFT1M dataset. We found that the sequential sampling costs so little time that it almost cannot be seen. The update step also costs little time compared to other stages. Hash code ranking and centroids update stage take most of time.

[1]https://gforge.inria.fr/projects/yael/

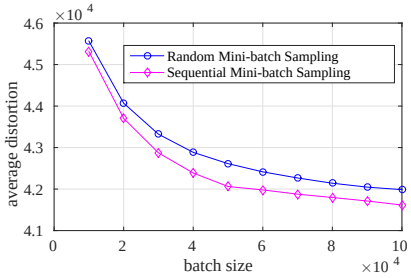


Figure 4: average distortion vs batchsize on SIFT1M

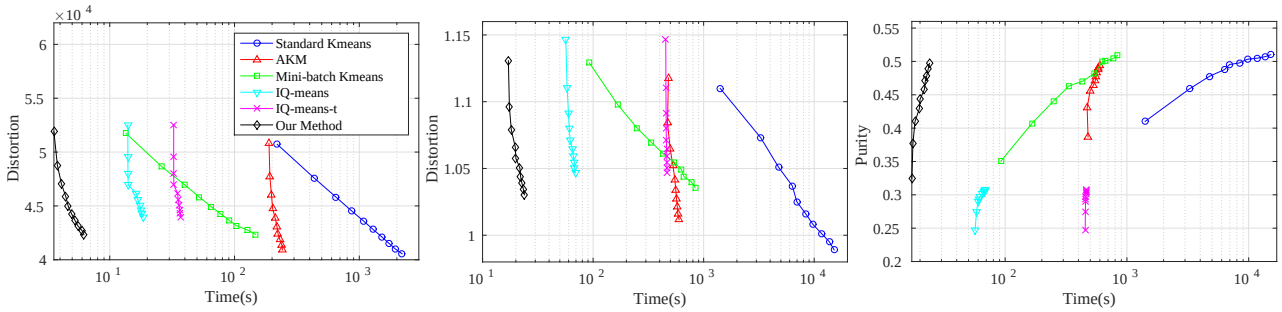
Clustering Results Comparison We compared our proposed algorithm with standard k-means and three approximate k-means algorithms as mentioned above. The time of IQ-means reported in [1] only includes encoding time, inverting time, and searching time. Since the learning time of IQ-means is very large, here we also report the overall time of IQ-means (denoted as IQ-means-t) including the learning time. Here Fig.5a and Fig. 5b report the average distortion versus time after 20 iterations on SIFT1M and GIST1M dataset respectively while k varies from 1000 to 10000. It' clear that our algorithm is the fastest among compared algorithms. Besides, the average distortion of our methods is much lower than IQ-means and close to the mini-batch k-means. For ILSVRC2012 dataset, we report the mean purity versus time in Fig. 5c with k varying from 1000 to 10000. Fig.5c shows that our algorithm outperforms IQ-means in both speed and purity. With k increasing, the mean purity of our method gets close to the standard k-means, but our method is hundreds of times faster than the standard k-means.

We also report the speed-up ratio of our algorithm and the three approximate algorithms over the standard k-means in Table 1. Our algorithm obtains up to 634 × speed-up over the standard k-means on three datasets. For IQ-means, it's clear that the learning time is very large by comparing IQ-means and IQ-means-t. Even the learning time is excluded, our algorithm is still much faster than IQ-means.

Sampling Method Comparison In Fig. 4, we reported the distortion of Mini-batch k-means using random mini-batch sampling and sequential mini-batch sampling respectively while sampling batchsize varies from 10000 to 100000. Fig. 4 shows that sequential mini-batch sampling achieves lower distortion than random mini-batch sampling. Besides, the distortion decreases slowly after batchsize is larger than 60000, so we set batchsize to 60000 for proposed method to achieve a better time-distortion trade off.

Component Analysis In order to further analyse the effect of each stage in our algorithm, we conduct a series of experiments on SIFT1M by dropping one stage of proposed method and fixing other stages unchanged. Fig. 3 shows that dropping hash code ranking stage will increase the time greatly but only bring a little decrease on distortion, showing that hashing algorithms bring a large speed-ups. Dropping group clustering stage also slows down the algorithm very much. Besides, dropping Top-n re-ranking stage will increase the average distortion and also decrease a little of time, which means top-n re-ranking mainly aims to control the distortion.

Comparison of ANN method We also compared our proposed



(a) Average distortion vs time on SIFT1M **(b) Average distortion vs time on GIST1M** **(c) Mean purity vs time on ILSVRC2012**

Figure 5: Comparison of clustering result on SIFT1M,GIST1M, and ILSVRC2012 dataset

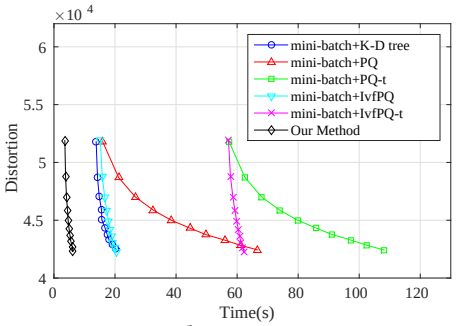


Figure 6: average distortion vs time on SIFT1M

method with mini-batch k-means combined with various Approximate Nearest Neighbor (ANN) method such as K-D tree, Product Quantization (PQ) [8], and Inverted-file based Product Quantization (IvfPQ) [8]. Since PQ and IvfPQ require learning the codebook, we also report the overall time (including the learning time) of PQ and IvfPQ which are denoted as mini-batch+PQ-t and mini-batch+IvfPQ-t respectively. Fig. 6 shows that our method achieves comparable result but is much faster than other compared methods.

4 CONCLUSION

By using a multi-stage filtering scheme, we propose a fast approximate k-means algorithm named multi-stage k-means (MKM). Extensive experiments show that our multi-stage k-means outperforms IQ-means in both speed and distortion. Compared to standard k-means, our method can achieve up to 634 × speed-ups with comparable accuracy for grouping millions of data into tens of thousand clusters.

ACKNOWLEDGEMENTS

This work was supported by National Natural Science Foundation of China (No.61332016, No.61602535, No.61503422 and No.61572500), the Scientific Research Key Program of Beijing Municipal Commission of Education (KZ201610005012), Youth Innovation Promotion Association CAS, and Jiangsu Key Laboratory of Big Data Analysis Technology.

Table 1: Speed-up over the standard k-means on SIFT1M, GIST1M, and ILSVRC2012 datasets.

Dataset	Method	K				
		2000	4000	6000	8000	10000
SIFT1M	AKM	2.3	4.3	5.9	7.4	9.1
	Mini-batch k-means	16.7	16.6	16.7	16.6	14.9
	IQ-means	31.1	61.5	77.02	97.4	119.3
	IQ-means-t	13.4	26.5	36.5	47.5	59.5
	Our Method	99.3	179.1	289.6	341.9	365.4
GIST1M	AKM	7.0	12.0	15.0	20.4	25.3
	Mini-batch k-means	19.7	18.9	15.5	17.7	18.1
	IQ-means	56.4	104.0	128.9	173.8	210.2
	IQ-means-t	7.2	13.8	18.1	25.3	32.5
	Our Method	191.1	320.8	386.3	532.2	634.6
ILSVRC2012	AKM	2.2	4.1	5.7	6.8	7.8
	Mini-batch k-means	21.3	21.1	21.3	21.6	21.1
	IQ-means	28.3	51.7	73.4	96.5	116.0
	IQ-means-t	12.6	24.0	35.3	47.0	57.1
	Our Method	120.5	240.6	341.7	351.2	416.4

REFERENCES

[1] Yannis Avrithis, Yannis Kalantidis, Evangelos Anagnostopoulos, and Ioannis Z Emiris. 2015. Web-scale image clustering revisited. In *Proceedings of the IEEE International Conference on Computer Vision*. 1502–1510.

[2] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 380–388.

[3] Jian Cheng, Cong Leng, Jiaxiang Wu, Hainan Cui, and Hanqing Lu. 2014. Fast and accurate image matching with cascade hashing for 3d reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–8.

[4] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. 2015. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 579–587.

[5] Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *ICML*, Vol. 3. 147–153.

[6] Yunchao Gong, Marcin Pawlowski, Fei Yang, Louis Brandy, Lubomir Bourdev, and Rob Fergus. 2015. Web scale photo hash clustering on a single machine. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 19–27.

[7] Greg Hamerly. 2010. Making k-means Even Faster.. In *SDM*. SIAM, 130–140.

[8] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2011), 117–128.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[10] Cong Leng, Jiaxiang Wu, Jian Cheng, Xiao Bai, and Hanqing Lu. 2015. Online sketching hashing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2503–2511.

[11] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. 2007. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.

[12] David Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*. ACM, 1177–1178.