Analysis and Implementation of K-Means++ with Parallel Initialization

Bin Han, Lingxi Song

Department of Statistical Science
Duke University, Durham, NC

April 27, 2019

Abstract

k-Means is one of the popular unsupervised clustering algorithms which has been widely used over a half century. However, the major drawback of K-Means is that it does not guarantee globally optimal solution, which is usually the case, due to that fact that the initialization of centroids is critically important but completely at random. Additionally, the number of iterations to calculate distance in pairs grows exponentially when number of data points increments, making K-Means not scalable with large data set. K-Means++, an improved algorithm based on K-Means, modifies the initiation process to provably generate the solution that is close to the globally optimal one. However, similarly, due to the inherent nature of sequential initialization of centers, K-Means++ must look over k data chunks and pick one randomly from each chunk, leading to the result that K-Means++ is not well-capable of handling massive data. Based on the idea of K-Means++, the researchers designed a new way to initialize centers parallelly, which dramatically reduce the number of passes. K-Meansll only requires log(k) passes to assign the centers at the beginning but is able to generate a nearly globally optimal solution.

Keywords K-Means; parallel initialization; time-complexity; optimal solution

Original Paper Bahmani,B., et al.(2012), Scalable K-Means++. Proceeding of VLDB Endowment, Volume 5, Issue 7,Page 622-633

1 Introduction

Unsupervised clustering is a popular machine learning technique that classifies a set of objects into specific groups. Objects in the same group should have more similar features than objects belonging to the different groups. There are many clustering methods, among which is K-Means the most famous and widely-used one. K-Means method starts with some randomly selected data point as initial cluster centers and then update the centers as the centroid of each cluster. The process stops when the computed results remain the same between two iterations. The iterative process is called **Lloyd's iteration**. The idea of K-means is simple and easy due to its nature of iterative calculation, which is the primary reason K-means has been popular for a long time.

As mentioned in the original paper, however, "from a theoretical standpoint, k-means is not a good clustering algorithm in terms of efficiency or quality: the running time can be exponential in the worst case and even though the final solution is locally optimal, it can be far away from the global optimum." (Bhmani el., 2012) The main reason is because that K-means depends heavily on the choice of initial centers, which is purely random. The chance of initializing the "right" centers leading to the global optimum gets smaller when the sample size increases.

To improve K-Means algorithm, scientists have focused on the initialization process. Noticing that a satisfying final clustering is spread out, we could design an initialization process that tends to select dispersed initial centers. Under this concern, Ostrovsky et al., Arthur and Vassilvitskii created an algorithm called K-Means++ that reinforced the initialization process of K-Means, which guarantees a solution that is closer to the global optimum than K-means, and shrinks the running time. "The algorithm selects only the first center uniformly at random from the data. Each subsequent center is selected with a probability proportional to its contribution to the overall error given the previous selections." (Bahmani et al., 2012) The main idea of K-means++ is to generate centers that are spread out. As mentioned in the paper, it can be formally proved that K-means++ can achieve O(logk) approximation of the optimum.

However, "the major drawback of k-means++ initialization from a scalability point of view is its inherent sequential nature: the choice of the next center depends on the current set of centers." (Bahmani et al., 2012) If we simply implement K-means++ with k desired clusters,

we need to make k passes over the samples to generate initial points, which gets worse with massive data or when we aim at large number of clusters. Therefore, the problem leads to the exploration and design of the algorithm – K-meansll, with similar quality of output solution as K-means++ together with the ability to be efficiently parallel implemented.

K-meansll is simply a parallel version of K-means++. In stead of sampling one point from each iteration, K-meansll samples O(k) points in O(logn) iterations. At the end of the all iterations, there are many points which are re-clustered to yield a solution that is "within a constant factor away from the optimum" (Bahmani et al., 2012). With their experiment, the researchers concluded that the solutions of K-meansll are consistently good or even better than any other method given the iteration number is at least 5. Additionally, K-meansll run must faster than K-means, as well as any existing parallel version of K-means++.

2 Algorithms

In this section, we follow along the structure of the original paper to provide the details of two algorithms – K-means++ and K-meansll. To ensure the accuracy and correctness of the researchers' work, we use the same notation and mathematical formulas from the paper. In addition to the K-meansll algorithm itself, we incorporated the section 6 from the paper, mathematical proof of the running time, in this section, so that readers can have a comprehensive idea of the algorithm before we dive into the algorithm testing in section 3.

2.1 Notation

Given a set of points $X = \{x_1, x_2, ..., x_n\}$, The distance between a point x and a subset of $Y \subseteq X$ is defined as $d(x, Y) = min_{y \in Y} \parallel x - y \parallel$ and the centroid of subset Y is given by

$$centeroid(Y) = \frac{1}{\mid Y \mid} \sum_{y \in Y} y$$

Let $C = \{c_1, ..., c_k\}$ be a set of clustering centers selected from X, the cost of Y with respect to C is defined as

$$\phi_Y(C) = \sum_{y \in Y} d^2(y, C) = \sum_{y \in Y} min_{i=1,...,k} \parallel y - c_i \parallel^2$$

The cost $\phi_Y(C)$ is a good numeric measure of how the points in the same group are clustered. Therefore, the goal of K-Means clustering is to find the optimal solution C to minimize $\phi_Y(C)$.

Let ϕ^* be the globally optimal cost value of K-means. We call that a set $\mathcal{C}$ of centers is $\alpha - approximation$ to K-means if $\phi_X(C) \leq \alpha\phi^*$

2.2 K-means++

As mentioned in the introduction, K-means++ generates initial centers one by one in a controlled manner. Provably, after the initialization process, K-means can achieve 8logk approximation to the global optimum in expectation. In the algorithm, we sample one center uniformly over the data X and add it to the set $\mathcal{C}$. While the number of points in the set is less than desired number of clusters k, we sample x from X with some probability in each iteration. See Algorithm 1 for details.

Algorithm 1 K-means++ (k)initialization.

1: $\mathcal{C} \leftarrow$ sample a point uniformly at random from X
2: **while** $|\mathcal{C}| <$ k **do**
3: Sample x ∈ X with probability $\frac{d^2(x,C)}{\phi_X(C)}$
4: $\mathcal{C} \leftarrow \mathcal{C} \cup \{x\}$
5: **end while**

2.3 K-meansll

2.3.1 Algorithm

The intuition behind K-meansll is to combine the advantages from two previously mentioned algorithms. It is an algorithm that performs with a small number of iterations, chooses more than one points in each round in a non-uniform way, and generates solution that is closer to the global optimum.

Largely inspired but different from K-means++, K-meansll applies an *oversampling factor* $l = \Omega(k)$, which is used in the probability when choosing which data points to be included in the set of $\mathcal{C}$. The algorithm picks one center uniformly over data X and computes its initial cost. Following by logψ iterations, the algorithm samples each x with probability $\frac{l*d^2(x,C)}{\phi_X(C)}$. The sampled data points are added into set $\mathcal{C}$ and $\phi_X(C)$ is updated. When the iteration ends, we have a set of data points and we assign each point a weight, which will be used later to in the re-clustering process. The details are in the algorithm 2.

Algorithm 2 K-meansll (k, l)initialization.

1: $\mathcal{C} \leftarrow$ sample a point uniformly at random from X
2: $\psi \leftarrow \phi_X(C)$
3: **for** $O(log\psi)$ times **do**
4: $\mathcal{C}' \leftarrow$ Sample each point x ∈ X independently with probability $p_x = \frac{l*d^2(x,C)}{\phi_X(C)}$
5: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$
6: **end for**
7: For x ∈ C, set w_x to be the number of points in X closer to x than any other point in $\mathcal{C}$
8: Recluster the weighted points in $\mathcal{C}$ into k clusters

2.3.2 Theorem – Constant Approximation

The researchers formally proved that K-meansll guarantees the quality of the output, that is the solution is closer to the global opimum. We will use the same mathematical proof from the original paper to ensure correctness and accuracy.

Theorem 2.1. *If an initialization is used in Step 8, then k-measll is an O(logk)-approximation.*

Proof: Suppose that cluster A is the optimal k-means solution and |A| = T. We sort the points in A in an increasing order of their distance to the center of A: let the ordering be $a_1, \ldots, a_T$. Let q_t be the probability that a_t is the first point that is picked up by k-meansll and q_{T+1} be the probability that no point is picked up from A. Let p_t be the probability of selecting a_t, then we have $p_t = \frac{l*d^2(a_t,C)}{\phi_X(C)}$. Since k-meansll picks each poing independently, for any $1 \leq t \leq T$, we have $q_t = p_t \prod_{j=1}^{t-1}(1-p_j)$, and $q_{T+1} = 1 - \sum_{t=1}^{T} q_t$.

If a_t is the first point picked up as a new center, we can either assign all the points in A to a_t, or just use the current clustering of A. Therefore, let

$$s_t = min \left\{ \phi_A, \sum_{a \in A} ||a - a_t||^2 \right\}$$

we have

$$E[\phi_A(\mathcal{C} \cup \mathcal{C}')] \leq \sum_{t=1}^{T} q_t s_t + q_{T+1}\phi_A(C)$$

By conducting a mean-field analysis, in which we assume all $p'_t s (1 \leq t \leq T)$ to be equal to some value p. We could think of it as all the points in A are very far from the current clustering. Under this circumstance, we have $q_t = p(1-p)^{t-1}$ and $q_{t1 \leq t \leq T}$ is a monotone decreasing sequence, By the ordering on $a'_t s$, let

$$s'_t = \sum_{a \in A} ||a - a_t||^2$$

We then know that $s'_{t1 \leq t \leq T}$ is an increasing sequence. Therefore

$$\sum_{t=1}^{T} q_t s_t \leq \sum_{t=1}^{T} q_t s'_t \leq \frac{1}{T}(\sum_{t=1}^{T} q_t \sum_{t=1}^{T}) s'_t,$$

where the last inequality, an instance of Chebyshev's sum inequality is using the inverse monotonicity of sequences q_t and s'_t. It is easy to see that $\frac{1}{T}\sum_{t=1}^{T} s'_t = 2\phi^*_A$. Therefore,

$$E[\phi_A(\mathcal{C} \cup \mathcal{C}')] \leq (1 - q_{T+1})2\phi^*_A + q_{T+1}\phi_A(C)$$

This shows that in each iteration of k-meansll, for each optimal cluster A, it removes a fraction of ϕ^*_A. Therefore, steps 1-6 generate a constant factor approximation to k-means.

2.3.3 Theorem – Cost Reduction

In this section we prove theoretically that k-means‖ can reduce the cost of current solution in each round significantly. The Mathematical expression is as followed.

Theorem 2.2. *Let $\alpha = exp(-(1 - e^{\frac{-l}{2k}})) \approx e^{\frac{-l}{2k}}$. If C is the set of centers at the beginning of an iteration of k-means‖ and C′ is the random set of centers added in that iteration, then*

$$E[\phi_X(C \cup C\prime)] \leq 8\phi^* + \frac{1+\alpha}{2}\phi_X(C),$$

with $\phi_X(C)$ and ϕ^* being the cost before and after adding new set C′ respectively.

To prove this theorem we need to define some notations first. A is a cluster from the optimal solution of k-means clustering with T points, and let $a_1, a_2, ..., a_T \in A$ be the points that are sorted in an increasing order with respect to their distance to centroid(A). The probability that a_t is selected by this algorithm is denoted as p_t, and by definition of the algorithm p_t is given by $\ell d^2(a_t, C)/\phi_x(C)$. Let q_t be the first selection probability as

$$q_t = Pr[a_t \in C\prime, a_j \notin C\prime, \forall 1 \leq j < t].$$

q_t is the probability that the first sampled point is a_t, and we denote by q_{T+1} the probability that no point is sampled from cluster A. Under the consideration that this algorithm samples each point independently, q_t can be calculated as

$$q_t = p_t \sum_{j=1}^{t-1}(1 - p_j).$$

If a_t is already selected, then we can either update a_t as a new centroid or stick to the current clustering of A. Therefore, let

$$s_t = min\{\phi A, \sum_{a \in A} \parallel a - a_t \parallel^2\}, \forall 1 \leq t < T, s_{T+1} = \phi_A$$

Then we denote the cost of cluster A after the current round of the algorithm as $\phi\prime_A$, $\phi\prime_A = \phi_A(C \cup C\prime)$.

Lemma 2.3. *The expected cost of clustering an optimum cluster A after a round of k-means‖ algorithm is bounded as*

$$E[\phi\prime_A] \leq \sum_{t=1}^{T+1} q_t s_t$$

Proof. We can recall the definition of s_t and rewrite the expectation of the clustering cost $\phi\prime_A$ for cluster A after one round of the algorithm as follows:

$$E[\phi\prime_A] = \sum_{t=1}^{T} q_t E[\phi\prime_A | a_t \in C\prime, a_j \notin C\prime, \forall 1 \leq j < t] \leq \sum_{t=1}^{T+1} q_t s_t$$

□

In order to minimize $\sum_{t=1}^{T+1} q_t s_t$, we tend to select the point close to the optimal center compared to those further away. The sample design of k-means‖ based on $D^2(.)$ implies a constraint on the probability that an early point is not picked, which is detailed below.

Lemma 2.4. *Let* $\eta_0 = 1, \forall 1 \le t \le T$, $\eta_t = \prod_{j=1}^{t}(1 - \frac{D^2(a_j)}{\phi_A}(1 - q_{T+1}))$. *Then* $\forall 0 \le t \le T$, $\sum_{r=t+1}^{T+1} q_r \le \eta_t$

Proof. Since $q_{T+1} = \prod_{t=1}^{T}(1 - p_t) \ge 1 - \sum_{t=1}^{T} p_t$, we have

$$1 - q_{T+1} \le \sum_{t=1}^{T} p_t = \ell \frac{\phi_A}{\phi}$$

Thus

$$p_t = \frac{\ell D^2(a_t)}{\phi} \ge \frac{D^2(a_t)}{\phi_A}(1 - q_{T+1})$$

By the definition of q_r, we have

$$\begin{aligned}
\sum_{r=t+1}^{T+1} q_r &= (\prod_{j=1}^{t}(1 - p_j)) . \sum_{r=t+1}^{T+1} \prod_{j=t+1}^{r-1} (1 - p_j) p_r \\
&\le \prod_{j=1}^{t}(1 - p_j) \\
&\le \prod_{j=1}^{t}(1 - \frac{D^2(a_t)}{\phi_A}(1 - q_{T+1})) \\
&= \eta_t
\end{aligned}$$

□

Lemma 2.5. *The expected potential of an optimal cluster A after a sampling step in k-means‖ algorithm is bounded as*

$$E[\phi\prime_A] \le (1 - q_{T+1}) \sum_{t=1}^{T} \frac{D^2(a_t)}{\phi_A} s_t + \eta T \phi_A.$$

Proof. Since the points in A are sorted in an increasing order with respect to their distances to the centroid, let

$$s\prime_t = \sum_{a \in A} \| a - a_t \|^2,$$

Noticing that $s\prime_1 \le \cdots \le s\prime_T$, we have $s_1 \le \cdots \le s_T \le s_{T+1}, \forall 1 \le t \le T$

Then we can bound the value of $min_{\alpha_0, \ldots \alpha_T} \sum_{t=0}^{T} \eta_t \alpha_t$ and hence the value of the primal(which by the previous two lemmas is an upper bound on $E[\phi\prime_A]$) as follows:

$$\begin{aligned}
E[\phi\prime_A] &\le \sum_{t=0}^{T} \eta_t \alpha_t \\
&= \sum_{t=1}^{T} s_t(\eta_{t-1} - \eta_t) + \eta_T s_{T+1} \\
&= \sum_{t=1}^{T} s_t \eta_{t-1} (\frac{D^2(a_t)}{\phi_A})(1 - q_{T+1}) + \eta_T s_{T+1} \\
&\le (1 - q_{T+1}) \sum_{t=1}^{T} \frac{D^2(a_t)}{\phi_A} s_t + \eta_T \phi_A,
\end{aligned}$$

where the last step follows since $\eta_t \le 1$. □

By the triangle inequality, we have a corollary of this lemma:

Corollary 2.5.1. $E[\phi\prime_A] \le 8\phi^*_A(1 - q_{T+1}) + \phi_A e^{-(1-q_T+1)}$

Proof. We can find the boundary for $\sum_{t=1}^{T} \frac{D^2(a_t)}{\phi_A} s_t$ and $\eta_T \phi_A$ respectively:

$$\begin{aligned}
\sum_{t=1}^{T} \frac{D^2(a_t)}{\phi_A} s_t &\le \sum_{t=1}^{T} (\frac{2}{T} + \frac{2}{T} \frac{s\prime_t}{\phi_A}) s_t \\
&= \frac{2}{T} \sum_{t=1}^{T} s_t + \frac{2}{T} \sum_{t=1}^{T} \frac{s\prime_t s_t}{\phi_A} \\
&\le \frac{2}{T} \sum_{t=1}^{T} s\prime_t + \frac{2}{T} \sum_{t=1}^{T} s\prime_t \\
&= 8\phi^*_A,
\end{aligned}$$

where in the last inequality, we used $s_t \le s\prime_t$ for the first summation, and $s_t \le \phi_A$ for the second one. And for the second part we have:

$$\begin{aligned}
\eta_T &= \prod_{j=1}^{T}(1 - \frac{D^2(a_t)}{\phi_A}(1 - q_{T+1})) \\
&\le exp(-\sum_{j=1}^{T} \frac{D^2(a_t)}{\phi_A}(1 - q_{T+1})) \\
&= exp(-(1 - q_{T+1}))
\end{aligned}$$

Combining the former two inequalities and we can prove this lemma. □

Finally we are able to prove the main conclusion:

$$E[\phi_X(C \cup C\prime)] \le 8\phi^* + \frac{1 + \alpha}{2} \phi_X(C)$$

.

Proof. Let $A_1, \ldots, A_k$ be the clusters in the optimal solution C_{ϕ^*}. We partition these clusters into "heavy" and "light" as follows:

$$C_H = \{A \in C_{\phi^*} | \frac{\phi_A}{\phi} > \frac{1}{2k}\}$$

and $C_L = \frac{C_{\phi^*}}{C_H}$ recall that

$$q_{T+1} = \prod_{j=1}^{T}(1 - p_j) \le exp(-\sum_{j} p_j) = exp(-\frac{\ell \phi_A}{\phi})$$

then by corollary, for any heavy cluster A, we have

$$\begin{aligned}
E[\phi\prime_A] &\le 8\phi^*_A(1 - q_{T+1}) + \phi_A e^{-(1-q_{T+1})} \\
&\le 8\phi^*_A + exp(-(1 - e^{-\frac{\ell}{2k}}))\phi_A \\
&= 8\phi^*_A + \alpha\phi_A.
\end{aligned}$$

summing up over all $A \in C_H$, we got $E[\phi\prime_{C_H}] \leq 8\phi^*_{C_H} + \alpha\phi_{C_H}$. Then by noting that

$$\phi_{C_L} \leq \frac{\phi}{2k} \mid C_L \mid \leq \frac{\phi}{2k}k = \frac{\phi}{2}$$

and that $E[\phi\prime_{C_L}] \leq \phi_{C_L}$, we have

$$\begin{aligned} E[\phi\prime] &\leq 8\phi^*_{C_H} + \alpha\phi_{C_H} + \phi_{C_L} \\ &= 8\phi^*_{C_H} + \alpha\phi + (1-\alpha)\phi_{C_L} \\ &\leq 8\phi^*_{C_H} + (\alpha + (1-\alpha)/2)\phi \\ &\leq 8\phi^* + (\alpha + (1-\alpha)/2)\phi. \end{aligned}$$

□

3 Implementation & Optimization

In Python, the package **sklearn** already has function **Kmeans**, with both random initialization (K-Means) and the sequential initialization (K-Means++). However, to better compare the performances of different algorithms in later work, both visually and numerically, we decided to implement all three algorithms from scripts, generating quantities that can help us evaluate three methods. For instance, the package can only generate the value of the cost function from the last iteration, while in our algorithms we save all the cost values so that we can visually depict the magnitude of the change between each iteration. The code is publicly available in our Github repository.

When we coded all three algorithms at the first attempt, we applied our own function to calculate the distance matrix, in order to determine which cluster each data point belongs to. Our function utilizes two embedded for loops, which are intuitively convenient but add complexity in the function and increase the running time. The situation gets worse when the number of data points increases, leading to an exponential increment in time complexity.

```
for i in range(# of rows):
    for j in range(# of clusters):
        distance[i,j] = ...
```

We tried to solve this issue by using some package functions in Python. We figured out an available function **pairwise_distances** under **sklearn.metrics** package. We compared the two distance calculation methods on K-Means using a sample data and table 1 compares the running time. Notice that since the only difference between K-Means, K-Means++, and K-Meansll is the initialization process, the optimization process is valid to be applied on the rest two methods.

The **pairwise_distances** function reduces the running time almost 40 times compared to the two embedded for-loops. Therefore, we decided to proceed with the **pairwise_distances** function within our algorithm.

Table 1: Distance Calculation Time

Function	Time (ms)
Embedded For-Loop Function	$110 \pm 3.18^{(1)}$
Pairwise Distance Function	$4380 \pm 501^{(2)}$

(1): std is for 7 runs, 10 loops each

(2): std is for 7 runs, 1 loop each

4 Algorithm Testing

In order to determine if K-Meansll indeed improves the performance in terms of running faster and is closer to optimal solution, compared with K-Means and K-Means++, we tested three models altogether on one real-world data set and one simulated data set. We tested the clustering correctness, running time, and cost reduction on simulated data, while tested the running time and cost reduction on the real data set. The results show that K-Meansll yields clusters that are closer to the optimal solution, while not necessarily running faster, which conflicts with the main conclusion from the paper. In this section, we will provide explanation of the data sets and the result analysis. As for the discrepancy between our result and the work from paper, we will discuss in section 5.

4.1 Simulated Data

The simulated data contains randomly generated data points in R^2, which forms five clusters with some small overlapping areas as noise. We choose to simulate data in 2D space because of the convenience to visualize the clusters and centers, giving direct visual demonstrations of how the algorithm works. Since the simulated data is relatively small, all three algorithms converged ultimately.

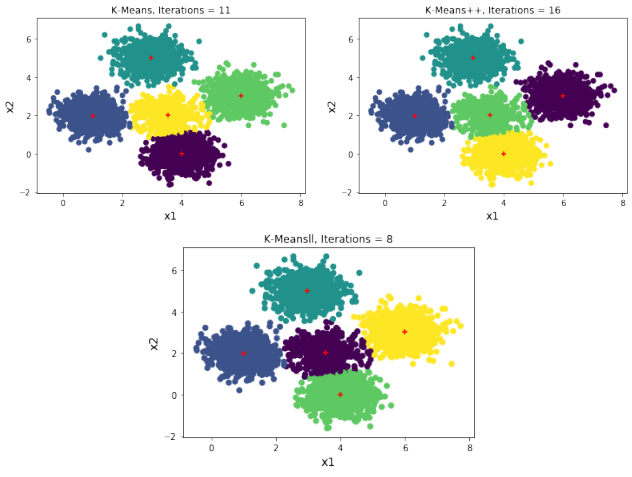

Figure 1: Final cluster plots from three algorithms, with number of iterations displayed

In terms of the clustering correctness visualized by the clustering plots, all three methods generate five clusters

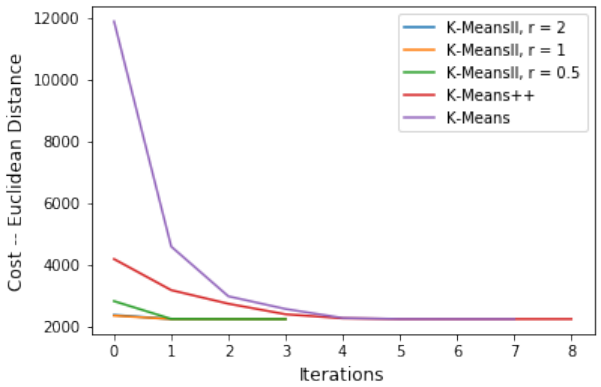

Figure 2: Cost Values by Iteration Plot

Table 2: Simulation

Method	**Time (ms)**
K-Means	$18.6 \pm 0.95^{(1)}$
K-Means++	$24.3 \pm 2.69^{(1)}$
K-Meansll, l = 0.5	$352 \pm 95.9^{(2)}$
K-Meansll, l = 1	$562 \pm 136^{(2)}$
K-Meansll, l = 2	$996 \pm 99.3^{(2)}$

(1): std is for 7 runs, 10 loops each

(2): std is for 7 runs, 1 loops each

that almost exactly match the original clusters (see figure 1). Notice that for K-meansll, we tested l, the oversampling factor, with values of 0.5, 1, and 2. But we only displayed the clustering plot of l=0.5. The cost plot by iterations (figure 2) shows the change of costs calculated at each iteration. We can see that at the initial point, K-meansll has smaller cost value than K-means++ and K-means, indicating that the initialization step already leads to a solution that is closer to the global optimum. There is no significant difference of K-meansll with different oversampling factors, and K-meansll generally converges faster than other two algorithms.

On the contrary, Table 2 displaying the running time disagrees with the conclusion from the paper. It shows that K-meansll actually increases the running time in a large magnitude compared with K-means++ and K-Means.

4.2 Real Data - Spam

The real-world data set we tested with is the SPAM data, which is the collection of emails that were labeled as spam or non-spam. It is publicly available from the University of California – Irvine, Machine Learning Repository. It is also one of the data sets the original paper tested on. The data set contains 4601 rows and 58 features. To be consistent with the original study, we tested three different k values (number of clusters): $k \in \{20, 50, 100\}$.

We compare the same two covariates, running time and cost, among three algorithms.

Figure 3 shows the cost values by iterations with k = 20. The first figure combines all five line plots together. Since the initial cost value from K-means is so large that the change of cost from other methods are almost invisible, we separated the cost-iteration plots for K-means++ and K-meansll. Similar as the results we arrived at with the simulation data, we can determine that with the Spam data, K-Meansll dramatically lower the initial cost after the initialization of centers, and the final cost is smaller as well, inferring as a solution closer to the global optimum. However, we noticed that this time, K-meansll with l = 0.5 and 1 did not converge faster than K-means++, while it was faster with l = 2. But generally, larger l corresponds to a lower initial cost. The results with k = 50 and 100 are very similar. Therefore, we did not display the results.

However, we observed that it is not always the case to have a lower initial cost with K-Meansll, both with the simulated data and the spam data. Occasionally, the initial cost of K-Meansll is higher than K-Means++, or even K-Means, with different trials we tested on. We will analyze the reason in section 5.

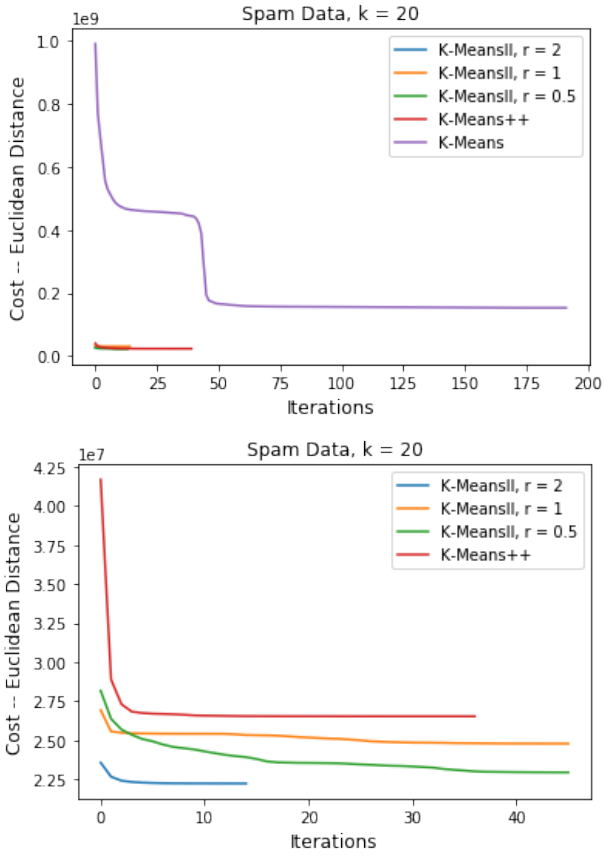

Figure 3: Cost Values by Iteration plot

Table 3, correspondingly, displays the runnnig time

Table 3: Spam Data, $k \in \{20, 50, 100\}$

Method	Time (s)*		
	k=20	k=50	k=100
K-Means	0.83 ± 0.12	1.39 ± 0.5	0.59 ± 0.13
K-Means++	0.22 ± 0.05	0.64 ± 0.14	0.94 ± 0.09
K-Meansll, l = $\frac{1}{2}$	2.27 ± 0.14	6.25 ± 1.04	9.49 ± 0.38
K-Meansll, r = 1	3.93 ± 0.29	9.81 ± 0.32	18.9 ± 1.21
K-Meansll, r = 2	8.56 ± 0.65	19.2 ± 0.91	38.3 ± 1.49

*:All the std are of 7 runs, 1 loop each

for three clustering algorithms. Here we listed the test results for $k \in \{20, 50, 100\}$. Similarly, the running time of K-Meansll is way longer than that of K-Means and K-Means++, which is opposite to the conclusion from the paper.

Based on the test with simulated data and the real data, it does not seem like a coincidence that K-meansll runs slower than other two methods. We are concerned about this issue and we conducted further analysis about potential reasons.

5 Discussion

Based on the results from the tests on simulated data and real-world data, we figured out some consistence, as well as some conflict from the original paper.

The original paper tested the same three initialization methods – Random (K-Means), K-Means++, and K-Meanll. The partially consistent observation we both found out is that the K-Meansll drastically lower the cost. Table 4 shows the cost value after the initiation step on the Spam data from the paper. Nevertheless, we did get some results when K-Meansll has higher initial cost than K-Means++ and K-Means.

Table 4: Spam Data - Cost After Initialization

Method	k=20	k=50	k=100
Random	-	-	-
K-Means++	460	110	40
K-Meansll, l=k/2	310	82	29
K-Meansll, l=2k	260	69	24

Values are in the unit of 10^5

The running time shows dramatic conflicts. One of the standing point the paper is trying to prove is that: "k-meansll runs in a fewer number of rounds when compared to k-means++, which translates into a faster running time especially in the parallel implementation" (Bahmani, et al., 2012). From our results we can see that both in the simulated data and the Spam data, K-Meansll actually performed much more slowely than K-Means++, or even K-Means. That means using the K-Meansll algorithm we implemented, the only improvement is the result that is closer to the optimal solution.

We proposed several potential reasons to explain the big divergence:

- Even though K-Meansll applies parallel initialization, with many data points yielded at the initial step, the generation process still has some random factors that could produce some non-desired results occasionally.
- K-means applies parallel initialization on K-means++ algorithm, while the paper further applied parallel implementation on K-meansll, using a computation model named **MapReduce**. Because of the time limitation to dive into details of the model and not being familiar with its open source version, Hadoop, we were not able to implement K-meansll parallel. This might be the major cause of the time differences.
- We created multiple functions that can be called within the main function to do calculations for each algorithm, which adds complexity in the process. The way the researchers implemented could be more efficient and effective, as they mentioned that they generated the same conclusion wth sequential version of K-meansll.

6 Conclusions

Based on our implementation and experimental results articulated above, we can conclude that K-meansll improves the clustering performance on the top of K-means++, as it yields solutions that are closer to the globally optimal solutions than K-Means++ or equally better. It has been tested on both simulated data and real world data. However, from our standpoint, we are not able to testify that K-meansll runs faster than K-means++, or even K-means. Some potential reasons have been discussed in section 5. In order to better reproduce the original work, parallel implementation of K-meansll needs to be tested.

`Repository: https://github.com/BeanHam/STA-663-Project`

References

[1] Bahmani, B., et al.(2012), Scalable K-Means++. Proceeding of VLDB Endowment, Volume 5, Issue 7,Page 622-633. Retrieved from: https://dl.acm.org/citation.cfm?id=2180915