SE3352: Algorithm Design Nov 27, 2021

Notes 1 – Clustering

Instructor: *Guoqiang Li* Scribes: *Haotian Wang, Yiyan Li*

1 Introduction

Clustering can be considered the most important unsupervised learning problem; As every other problem of this kind, it deals with finding a structure in a collection of unlabeled data.

Definition 1. *A cluster is a collection of objects which are "similar" between them and are "dissimilar" to the objects belonging to other clusters. Clustering is the algorithm that recognizes clusters from a given data set.*

Note that this is a very rough definition. Part of common application domains in which the clustering problem arises are as follows:

- **Multimedia Data Analysis:** Learning image or video representations without manual annotations. e.g. When using Streaming media platform, face clustering can recognize all the actors in any frame. [**?**, **?**]
- **Responding to public health crises:** With the increasing number of samples, the manual clustering of COVID-19 data samples becomes time-consuming. Clustering helps classify medical datasets deterministically.[**?**]
- **Social Network Analysis:** Clustering provides an important understanding of the community structure in the network. Results can be used for customer segmentation and sending ads. Put it in a formal way, **clustering groups the nodes of the graph into clusters**, taking into account the edge structure of the graph in such a way that there are several edges within each cluster and very few between clusters. [**?**]
- **Intermediate Step for other fundamental data mining problems:** Clustering can be considered as a form of data summarization. Many clustering methods are closely related to dimensionality reduction methods. Such methods can be considered a form of data summarization.
- **Intelligent Transportation:** Under **the online scenario**, data is in the form of streams, i.e., the whole dataset could not be accessed at the same time. In future intelligent transportation, low-latency online vehicle tracking is essential and can be solved by online clustering.[**?**]

Today we'll start from the naive K-means clustering and improve the algorithm step by step. The lecture has four main topics that we'll go through.

2 Problem Description

Definition 2. ***Cluster centroid****(or center) is the middle of a cluster. A centroid is a vector that contains one number for each variable, where each number is the mean of a variable for the observations in that cluster. The centroid can be thought of as the multi-dimensional average of the cluster.*

The k-means clustering problem is one of the oldest and most important questions in all of computational geometry. Given an integer k and a set of n data points in $\mathbb{R}^d$, the goal of this problem is to choose k centers so as to minimize the total squared distance between each point and its closest center.[**?**]

There are several kinds of k-means algorithms among which the most common algorithm , also called naive k-means algorithm, was first proposed by Stuart Lloyd[**?**] of Bell Labs in 1957.

For a k-means problem, we are given an integer k and a set of data vector $(x_1, x_2, x_3 \dots x_n)$ in d-dimension. And we need to choose k centroids to partition the n vectors into k types T $(T_1, T_2 \dots T_k)$ with the minimum within-cluster sum of squares $(WCSS)$ cost:

$$Cost(C_{1:k}, c_{1:k}) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - c_i\|^2$$

where c_i is the centroid of cluster C_i.

Lemma 3. *Let C be a cluster of points with its mean to be μ, and let c to be and arbitrary point.Then* $\sum_{x \in C} \|x - c\|^2 = \sum_{x \in C} \|x - \mu\|^2 + |C| \cdot \|c - \mu\|^2$

So we denote that:

$$\begin{aligned} Cost(C_{1:k}, c_{1:k}) &= \sum_{i=1}^{k} \sum_{x \in C_i} \|x - c_i\|^2 \\ &= \sum_{i=1}^{k} (\sum_{x \in C_i} \|x - \mu_i\|^2 + |C_i| \cdot \|c_i - \mu_i\|^2) \\ &= Cost(C_{1:k}, mean(C_{1:k})) + \sum_{i=1}^{k} |C_i| \cdot \|c_i - \mu_i\|^2 \end{aligned}$$

3 Algorithms

3.1 The K-means algorithm

3.1.1 Algorithm Details

The k-means algorithm is a simple and fast algorithm for this problem, although it offers no approximation guarantees at all. It iteratively calculates the sum of distance within a cluster and updates the partition.The details are as follows. 1

1. Arbitrarily choose and initial k centroids $\mathcal{C} = \{c_1, c_2 \dots c_k\}$
2. For each $i \in \{1, 2 \dots k\}$, set the cluster C_i to be the set of points that are closer to c_i than they are to c_j for all $j \neq i$

3. For each $i \in \{1, 2 \dots k\}$, set c_i to be the center of all points in C_i where $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$

4. Repeat Step 2 and Step 3 until C no longer changes.

Algorithm 1 K-means

Input: k: number of output cluster; Data: input data
Output: $\mathcal{S}$: set of all clusters S_i

```
Arbitrarily initialize k centroids C = {c_1, c_2 ... c_k}
repeat
  for each point x in Data S do
    for i = 0 → k do
      for j = 0 → k do
        set x to be a member of cluster S_i where ||x − c_i||^2 < ||x − c_j||^2
      end for
    end for
  end for
  for i = 0 → k do
    c_i ← (1/|S_i|) Σ_{x∈S_i} x
    set c_i to be the centroid of all points in cluster S_i
  end for
until S stays unchanaged
```

Output: $\mathcal{S}$: set of all clusters S_i

3.1.2 Convergence

Lemma 4. *Let $x_1, x_2 \dots x_n$ be n vectors in $\mathbb{R}^d$, then $f(x) = \sum_{i=1}^{n} \|x_i - x\|^2$ gets its minimum iff. $x = \frac{1}{n} \sum_{i=1}^{n} x_i$*

证明.

$$\begin{aligned}
\frac{df(x)}{dx} &= \frac{d \sum_{i=1}^{n} \|x_i - x\|^2}{dx} \\
&= -2 \sum_{i=1}^{n} (x_i - x) \\
&= 0 \\
x &= \frac{1}{n} \sum_{i=1}^{n} x_i
\end{aligned}$$

$x = \frac{1}{n} \sum_{i=1}^{n} x_i$ is a stationary point of this function. Owing that it is a strictly convex function, the stationary point is alse the only minimun point of that function. So the function gets its minimum at $x = \frac{1}{n} \sum_{i=1}^{n} x_i$. □

Lemma 5. *During the course of the k-means algorithm, the cost monotonically decreases.*

证明. Let $c_1^{(t)}, \dots c_k^{(t)}, C_1^{(t)}, \dots C_k^{(t)}$ denote the cetnroids and clusters at the start of t^{th} iteration of K-means. The first step of the iteration assigns each point to its nearest center, therefore:

$$Cost(C_{1:k}^{(t+1)}, c_{1:k}^{(t)}) \leq Cost(C_{1:k}^{(t)}, c_{1:k}^{(t)})$$

On the second step, each cluster re-centered at its mean. By Lemma 3:

$$Cost(C_{1:k}^{(t+1)}, c_{1:k}^{(t+1)}) \leq Cost(C_{1:k}^{(t+1)}, c_{1:k}^{(t)})$$

□

3.1.3 Time Complexity

证明. Naive K-means algorithm's time complexity is $O(k * n * i)$

In each iteration there are such steps:

- Distance calculation: To calculate the distance from a point to the centroid, we can use the squared Euclidean proximity function, which is thought to be $O(1)$
- Comparisons betweeen distances.
- Centroid calculation.

So the total number of operations in one iteration is:

$$\begin{aligned} C &= distance\ calculation + comparisons + centroids\ calculation \\ &= k * n * O(1) + (k-1) * n * O(1) + k * n * O(1) \\ &= O(kn) \end{aligned}$$

where k denotes the number of clusters, n denotes the count of data vectors and d denotes vector dimension.

And the whole process takes i iterations in total so the time complexity of K-means algorithm is $O(kni)$. □

3.1.4 DrawBack

The naive K-means algorithm do great work for the simplicity and efficiency, but it also has drawbacks. In the caes of initializing k centroids using naive K-means algorithm (usually Lloyd's algorithm), we use randomization. The initial k centroid are picked in the range of data set randomly. However, this initialization strategy could result in initialization sensitivity. The final formed clusters could be affected greatly by the initial picked centroids.

Here are a few figures showing the potential:

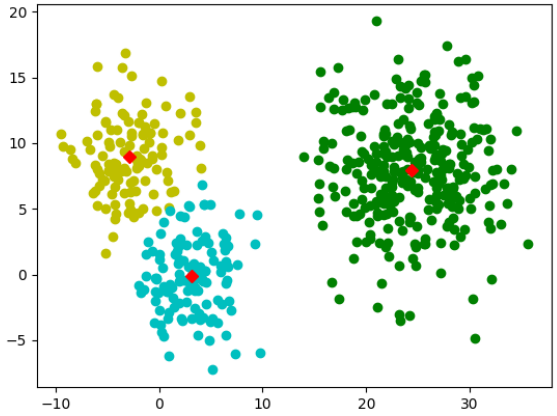

图 1: Good Cluster

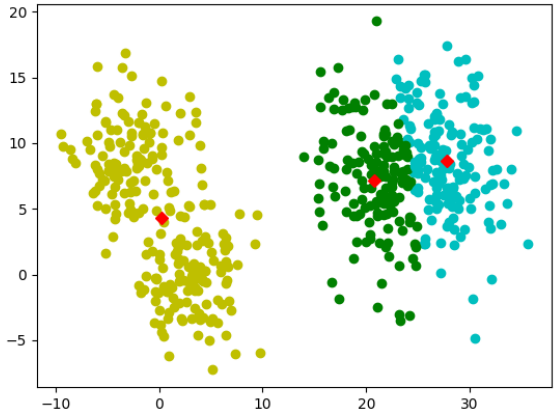

图 2: Bad Cluster

In the above images, the final formed clusters are pretty different. The good cluster's 1 initial k-centroids are initialized in different clusters, leading to a good result. While the other one's 2 initial centroids were unfortunately initialized in the same cluster and got a result which was not as expected.

3.2 Optimization of K-means

In last several decades, there has been significant work on improving Lloyd's algorithm both in terms of reducing MSE and running time. The follow up work on Lloyd's algorithm can be broadly divided into three categories:

1. Better seed selection
2. Selecting ideal value for number of clusters

3. Bounds on data point to cluster centroid distance

Next, we will introduce some typical work in these categories. Section 3.3 introduces K means++ for seed selection, section 3.4 and 3.5 helps bound on data point to cluster centroid distance and section 3.6 introduces several typical methods to select ideal value for number of clusters briefly. Let's begin.

3.3 The K-means++ algorithm

The K-means++ algorithm was proposed by David Arthur and Sergei Vassilvitskii in 2006, which outperforms K-means in terms of both accuracy and speed [**?**].

The K-means++ algorithm uses totally a different way to initialize the k centroids. Rather than uniformly randomly pick point in the range of all points, it uses special to make the k centroids as far away from each other as possible. And the updated algorithm does exactly the same in the iteration steps. The algorithm goes as follows:

1. Pick the first centroid c_1 randomly from the dataset S
2. Compute the distance of all points in the dataset from nearest, previously chosen centroid c_i. The distance of a point x could be calculated by

$$Dist(x) = \min_{c_i \in C} \|x - c_i\|$$

3. Take a new centroid c_i, choosing $x_j \in$ from all points with probability p_j

$$p_j = \frac{Dist^2(x_j)}{\sum_{x_j \in S} Dist^2(x_j)}$$

4. Repeat the above steps until k cetnroids are found
5. Iteration steps are exactly the same as naive K-means algorithm

k-means++ consistently outperformed k-means, both by achieving a lower potential value, in some cases by several orders of magnitude, and also by completing faster. With the synthetic examples, the k-means method does not perform well, because the random seeding will inevitably merge clusters together, and the algorithm will never be able to split them apart. The careful seeding method of k-means++ avoids this problem altogether, and it almost always attains the optimal results on the synthetic datasets.[**?**]

证明. K-means++ algorithm's initialization time complexity is $O(k * n)$

In each iteration there are such steps:

- Distance calculation: To calculate the distance from a point to a newly selected centroid, we can use the squared Euclidean proximity function, which is thought to be $O(1)$
- Comparisons betweeen distances.
- Centroid calculation.

So the total number of operations in i^{th} iteration is:

$$\begin{aligned} C_i &= distance\ calculation + comparisons + centroids\ selection \\ &= n * O(1) + n * O(1) + O(1) \\ &= O(n) \end{aligned}$$

where k denotes the number of clusters, n denotes the count of data vectors.

And the whole process takes k iterations in total so the time complexity of K-means++'s initialization is $O(kn)$. □

3.4 Avoid redundant computation

In both K-means and K-means++ algorithm, if there are n data points in $\mathbb{R}^d$ space and k clusters for partition, each iteration involves $n * k$ distance computations, which would significantly slow down algorithm. Based on the observation of unnecessary calculation in the process of iteration, Charles Elkan put forward a strict method to avoid unnecessary computation.[**?**] The key idea is to bound on data point to cluster centroid distance and use triangle inequality to avoid redundant computations of distance between data points and cluster centroids.

3.4.1 Inspiration

The accelerated algorithm applyies the triangle inequality in two different ways, keeping track of lower and upper bounds for distances between points and cluster centroids. Its inspiration is based on the fact that most distance calculations in standard K-means are redundant.

If a point is **far away from a centroid**, it is not necessary to calculate the exact distance between the point and the centroid in order to know that the point should not be assigned to this centroid. Conversely, **if a point is much closer to one center than to any other**, calculating exact distances is not necessary to know that the point should be assigned to the first center.

3.4.2 Augmentation target

There are 3 properties the accelerated K-means algorithm should satisfy:

- **Start with any initial centers**, so that all existing initialization methods incuding original K-means and K-means++ can continue to be used.
- **Correct results,** it always produces exactly the same final centers as the standard algorithm.
- **Support any black-box distance metric,** so it should not rely for example on optimizations specific to Euclidean distance.

This strong properties mean that more basic K-means algorithms are able to install this augmentation. e.g., heuristics for merging or splitting centers(like ISODATA, which not memtioned in our note) can be used together with the new algorithm.

3.4.3 Preliminary results

The algorithm firstly find the only black box property of any distance metrics. That is: for any 3 points x, y, z, satisfy triangle inequality:

$$d(x, z) \leq d(x, y) + d(y, z)$$

in which $d()$ means distance function.

Let x be a point and let b and c be centers, we need to know that $d(x, c) \geq d(x, b)$ in order to avoid calculating $d(x, c)$. Now I'd like to introduce 2 Lemmas.

Lemma 6. *Let x be a point and let b and c be centers. If $d(b, c) \geq 2d(x, b)$ then $d(x, c) \geq d(x, b)$.*

证明. Use triangle inequality, $d(b, c) - d(x, b) \leq d(x, c)$. And bring in $d(b, c) \geq 2d(x, b)$, we can get the conclusion. □

Lemma 7. *Let x be a point and let b and c be centers. $d(x, c) \geq max(0, d(x, b) - d(b, c))$.*

证明. Use triangle inequality $d(x, c) \geq d(x, b) - d(b, c)$, with $d(x, c) \geq 0$, we can get the conclusion. □

Let x be any data point, let c be the center to which is currently assigned, let s become any other center. So with the lemma, we can assert that:

Corollary 8. *if $\frac{1}{2}d(c, s) \geq d(x, c)$ then $d(x, s) \geq d(x, c)$, and we don't need to compute $d(x, s)$. (Proved by Lemma 6)*

Corollary 9. *Suppose that we don't know $d(x, c)$ exactly, and we do know an upper bound u such that $u \geq d(x, c)$: For any other possible choice, we only need to compute $d(x, c), d(x, s)$ iff $u > \frac{1}{2}d(c, s)$.(Proved by Corollary 8)*

Corollary 10. *Suppose that $u \leq \frac{1}{2}d(c, s)$ for any possible s, all distance calculations for x can be avoided. (Proved by Corollary 9)*

Next, Let x be any data point, let c be any center, let s become previous version of same center. Suppose that in the previous iteration we knew a lower bound g such that $d(x, s) \geq g$. Then we can infer a lower bound h for current iteration:

$$d(x, c) \geq max\{0, d(x, s) - d(s, c)\} \geq max\{0, g - d(s, c)\} = h$$

This can be easily proved by Lemma 7. So we can assert that:

Claim 11. *If center moved a small distance($d(s, c)$ is small, the lower bound only make a small move.*

3.4.4 Augmentation detail

In practical application, as the centers are converging to their final positions, the vast majority of the data points have the same closest center from one stage to the next. A good algorithm would exploit this coherence to improve running time. We use $u(x)$ to represent upper bound of distance between a given point x and its currently assigned center c. $l(x, c')$ is the lower bound on the distance between x and some other center c'. If $u(x) \leq l(x, c')$, we don't need to calculate $d(x, c), d(x, c')$.

Initially, we set $l(x, c) = 0$ for each point x and center c. Then assign each x to its closest initial center, using Lemma 6 to firstly reduce redundant distance computations. Then repeate this until convergence.(Each time $d(x, c)$ is calculated, we update $l(x, c)$. Similarly $u(x)$ when computing $d(x, c(x))$.)

1. For all centers c and c', compute $d(c, c')$. Set $s(c) = \frac{1}{2} min_{c \neq c'} d(c, c')$.
2. Identify all points x such that $u(x) \leq s(c(x))$. (Refer to Corollary 10)
3. For each pair of remaining x and c, which satisfy: i) $c \neq c(x)$ and ii) $u(x) > l(x, c)$(obviously) and iii) $u(x) > \frac{1}{2} d(c(x), c)$(Corollary 9):
   (a) If $r(x) = true$, compute $d(x, c(x))$ and assign $r(x) = false$. Otherwise, $d(x, c(x)) = u(x)$. (We don't have to update according to Claim 11)
   (b) If $d(x, c(x)) > l(x, c)$ or $d(x, c(x)) > \frac{1}{2} d(c(x), c)$, then compute $d(x, c)$ and decide if swap c for x. (Corollary 9)
4. For each center c, compute centroid, store in m(c).
5. For each pair of x and c, set $l(x, c) = max(l(x, c) - d(c, m(c)), 0)$
6. For each point x, set $u(x) = u(x) + d(m(c(x)), c(x))$
7. For each point x, set $r(x) = True$
8. Really replace c by m(c)

Logically, step (2) is redundant because its effect is achieved by condition 3(iii). Computationally, step (2) is benefificial in real experiment because reduce x's size for later steps. And note that u(x) and c(x) may change during the execution of step (3), so we can't discard condition 3(iii).

The most significant part is step (3), we use many restrictions to avoid redundant computations. In 3(a) step, we only compute $d(x, c(x))$ for at most one time. Just when in pervious step (7) r(x) is set to True, we update $d(x, c(x))$, or else we use u(x) to replace. Why step 3's restrictions are effcient is that at the start of each iteration, the upper bounds and lower bounds for x are tight enough. If at j^{th} iteration is tight, it'll be tight at $(j+1)^{th}$ iteration, because the location of most centers changes only slightly, and hence the bounds change only slightly.

3.4.5 Augmentation analysis

Compared to naive K-means++ algorithm, in 6 typical benchmark, using this optimazation speeds up algorithms from 11.3× to 351×.

3.5 Relax the restriction

In fact, the previous augmentation is designed under strict limitations. Still based on the same observation, the team of Siddhesh Khandelwal[**?**] makes contributions to help reduce this cost to $n * k'$ ($k' << k$) by generating candidate cluster list (CCL) of size k' for each data point. This augmentation makes trade-off between loss function and running time and relaxes the previous algorithm's restrictions. We'll show how this heuristic works in detail.

3.5.1 Inspiration

The optimization is based on the observation that across all iterations of K-means or K-means++, a data point changes its membership only among a small subset of clusters. The heuristic considers only a subset of **nearby cluster as candidates** for deciding membership for a data point. This heuristic has advantage of speeding up K-means and K-means++ clustering with marginal increase in loss functuion(MSE in our case). Note that the optimization can be applied in any algorithm to solve K-means problem with steps to caculate distance between points and centers, acting as augmentation.

3.5.2 Augmentation target

Let A be any variant algorithm of K-means problem and B be the same variant augmented with this geuristic. Let T be the time required for A to converge to MSE value of E. Let T' be the time required for B to converge to MSE value of E'. Our target is:

- **For convergence time:** $T' < T$
- **For loss:** $E' \leq E$ or $E' \overset{marginally}{>} E$

Note that the augmentation has no request of providing same results with naive K-means algorithm.

3.5.3 Augmentation detail

We assume that k' is significantly smaller than k. We will show how to choose k' later.We build candidate cluster list (CCL) based on top k' nearest clusters to the data point after first iteration of K-means.

Consider a data point p1 and cluster centroids represented as c_1, c_2..., c_k. We assume that $k' = 4$, and $k' << k$. After first iteration of K-means c_5, c_6, c_8, and c_{11} are the top four closest centroids to p_1 in the increasing order of distance. This is the candidate cluster list for p_1. If we run K-means for second iteration, p_1 will compute distance to all k centroids. After second iteration, there are two possible cases:

1. The list do not change but only members'ranking changes.
2. Several members of the centroids in the previous list are replaced with other centroids which were not in the list.

It seems that the augmentation makes no sense. But what makes this method succeed is **real world data rarely makes case 2 happen**. That is, the set of top few closest centroids for a data point **remains almost unchanged even though order among them might change**.

3.5.4 Augmentation analysis

Overhead analysis:

- **Computation overhead:** $O(nklog(k))$ for creating CCL at first. We have to compute the distance to each cluster's centroid for each point and sort them to create CCL.
- **Memory overhead:**$O(nk')$ to maintain CCL.

Working in the same 6 typical datasets, augmenting triangle inequality K-means(introduced in section 3.4) with this heuristic **speeds up from** $1.5\times$ **to** $3.1\times$ **compared with not augmenting this heuristic.**

3.6 Choose a proper K

How to evaluate goodness of clustering for various potential values of number of clusters? We will introduce several common methods.[**?**, **?**, **?**, **?**] Let's review our optimization. We speed up the algorithm with either lenient or strict conditions and seed the initial K centers properly. But still exists the only hyper-parameter K we don't know how to tune. So let's begin.

3.6.1 Use a range of set

We choose a set of K and compare their performance. In this set, we need to choose k significantly smaller than the number of objects in the data sets and let it be resonably large based on this.

3.6.2 statistical measures

There are several statistical measures available for selecting K. These measures are often applied in combination with probabilistic clustering approaches. They are calculated with certain assumptions about the underlying distribution of the data.

e.g. The Bayesian information criterion is calculated on data sets which are constructed by a set of **Gaussian distributions**. Monte Carlo techniques, which are associated with the null hypothesis, are used for assessing the clustering results and also for determining the number of clusters.

3.6.3 visualization

Visual verification is applied widely because of its simplicity and explanation possibilities. In my own practice, I usually use PCA or other methods to draw points on a planar graph to check how many clusters exist in Machine Learning course's project. But it has many restrictions. e.g. The application of visualization techniques implies a data distribution continuity in the expected clusters. In fact, Visual examples are often used to illustrate the drawbacks of an algorithm.

3.6.4 Determined in a later processing step

When K-means clustering is used as a pre-processing tool, the number of clusters is determined by the specific requirements of the main processing algorithm. No attention is paid to the effect of the clustering results on the performance of this algorithm. In such applications, the K-means algorithm is employed just as a black box without validation of the clustering result.

4 Key properties

4.1 K-means problem is NP Complete

K-means problem is an NP Hard problem, and two teams have proved them using 3-SAT and Exact Cover by 3-Sets respectively.[**?**, **?**] For lack of space, we won't describe their proof in detail. For reduction

from planar 3-SAT to k-means, the author corresponds a simple circuit to each variable x_i in 3-SAT. And each circuit has an even number of vertices marked on it. Circuits are partitioned into pairs of adjacent vertices according to the value of x_i. Then, each edge becomes a representation of value. The authors set the distance between vertices and uniquely determine a layout that gives a correct reduction from planar 3-SAT to planar k-means. We spent much time on this paper but still couldn't illustrate it clearly. Please refer to origin paper.

But anyway we can use K-means to solve 3-SAT. That is, K-means problem is NP-Complete problem. Existing local search algorithms that computes a certain local (and not necessarily global) optimum for this problem are what we really feel interested in. In fact, Lloyd's algorithm and its variants never provide any significant guarantee about how well the solution that it computes approximates the optimal solution, but just "reasonable approximation guarantees".

4.2 Algorithm complexity analysis

Please check our section 3 and every algorithm's complexity is analyzed soon after we give out its details.

5 Application

In real world, K-means are widely used in clustering for its simplicity and effcient. Its core concept is extremely easy to understand and the updated version algorithm itself only cost a few iterations to output pretty good results. And here are a few applications of K-means algorithm in real life.

5.1 Customer Segmentation

The segmentation of the customers' base allows operators to better serve customers and target advertising precisely to them[**?**]. The concept of segmentation relies on the high probability of persons grouped into segments based on common demands and behaviours to have a similar response to marketing strategies.

And the concept of segmentation coincides with that of clustering. A company can run K-means analysis base on customers' consumption habits, browsing histories, financial conditions and many other features to precisely partition their customers into different clusters so that they can implement different strategies.[**?**]

5.2 Optimization of Truck-Drone Delivery Netword

The K-means algorithm is used to optimize truck-drone in tandem delivery network. One of the objectives is to investigate the time, energy, and costs associated to a truck-drone delivery network compared to standalone truck or drone.

In 2016, Sergio Ferrández, Timothy Harbison, Troy Weber, Robert H. Sturges and Robert Rich proposed a method[**?**], optimizing the delivery netword of drone powerd by K-means algorithm to find the appropriate launch locations with the minimum cost. The optimal solution is determined by finding the minimum cost associated to the parabolic convex cost function.

5.3 Document Classification

Clustering documents in multiple categories based on tags, topics, and the content of the document is a very standard classification problem and k-means is a highly suitable algorithm for this purpose.[**?**]

All we need to do is to pre-processing the documents, representing each of them as a high-dimension vector and using term frequency to identify commonly used terms that help classify the document. The document vectors are then treated as input of K-means algorithm to help identify similarities in document groups.[**?**]

5.4 Research based on COVID-19 Dataset

Now we'll present a k-means clustering-based COVID-19 analysis to determine the clusters according to the health care quality of the countries. In fact we don't think this research meaningful, but just refer to their way to use K-Means in application.[**?**]

The team uses Principal Component Analysis (PCA) while determining the centroids.

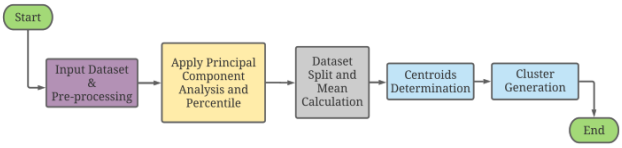

图 3: Flowchart of centroids deciding

With this method to decide on centroids, the author argues that their method uses much less iterations times and execution time for convergence. Compared with existing K-means++, their algorithm's performance is constant and faster.

But to be honest, we have to argue that their research just used K-means to generate clusters at last. In fact, with PCA, we can clearly observer that countries can be divided into 4 categories on the issue of COVID-19 situations.(which fits our impression) What K-means does just verifies countries in detail. That is, without enough knowledge of dataset, we have to decide on cluster number using other methods. After that, K-means will generate clusters for us finally. That's what section 3.6.4 implies.

6 Conclusion

Bringing in the K-means algorithm from the beginning, we give the initailization strategy and iteration steps of naive K-keans algorithm. Given the proof that the object function decrements in iteration, the objective always converges at a certain point in time. But the drawbacks are annoying and fatal, which may lead to far different results than expected and increase of iterations, so we have some optimization of them, bring us K-means++ and some other way to select initial centroids, avoid redundant computation, select proper magic number k etc. K-means are wildly used in many aspects of life, making it easier to do customer-partitioning, clsasifying documents, optimizing resource allocation, pattern recognizing in images and so many other application in industry and daily life.

Finally, we would like to give out some open questions we have about K-means for further research.

- **Lower bound of calculations:** Whether one can find a lower bound on how many distance calculations are needed by any implementation of exact K-means? With a theoretical lower bound, we can futher speed up the algorithm.
- **Use running time information to find K:** We found all papers don't take full advantage of information from running time. Can we fine tune the value of K during running the algorithm?

Appendix

A K-means Algorithm Code in Python

```
import math
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


def loadData():
    df = pd.read_csv("./data/data.csv")
    return df.values


def euclideanDistance(vector1, vector2):
    return math.sqrt(sum(np.power(vector1 - vector2, 2)))


def initRandomCentroids(data, k):
    count, dim = data.shape
    centroids = np.zeros((k, dim))
    colMax = np.max(data, axis=0)
    colMin = np.min(data, axis=0)
    colRange = colMax - colMin
    for i in range(k):
        centroid = colMin + np.random.rand(dim) * colRange
        centroids[i, :] = centroid
    print(centroids)
    return centroids


def kmeans(k):
    data = loadData()
    count = data.shape[0]
    centroids = initRandomCentroids(data, k)
    clusterBound = np.zeros((count, 2))
```

```
34      index = np.zeros((count, 1))
35      processing = True
36      while processing:
37          processing = False
38          for i in range(count):
39              minIndex = 0
40              minDist = float("inf")
41              for j in range(k):
42                  distance = euclideanDistance( centroids[j, :], data[i, :])
43                  if distance < minDist:
44                      minDist = distance
45                      minIndex = j
46
47              if clusterBound[i, 0] != minIndex:
48                  processing = True
49                  clusterBound[i, :] = minIndex, minDist ** 2
50          index[:, 0] = clusterBound[:, 0]
51          for j in range(k):
52              newCentroid = data[np.all(index == j, axis=1), :]
53              centroids[j, :] = np.mean(newCentroid, axis=0)
54      print("k means finished!")
55       visualization (centroids, clusterBound, data)
56
57
58  def visualization (centroids, clusterBound, data):
59      plotMarkList = ['oy', 'og', 'or', 'oc', '^m', '+y', 'sk', 'dw', '<b', 'pg']
60      centroidMarkList = ['Dr', 'Dc', 'Dm', 'Dy', '^k', '+w', 'sb', 'dg', '<r', 'pc']
61      k = centroids.shape[0]
62      count = data.shape[0]
63      if data.shape[1] != 2:
64          print("too many dimensions to draw :(")
65          return
66      if k > len(plotMarkList):
67          print("too many centroids to draw :(")
68          return
69      for i in range(count):
70          mark = plotMarkList[int(clusterBound[i, 0])]
71          plt.plot(data[i, 0], data[i, 1], mark)
72      for i in range(k):
73          mark = centroidMarkList[i]
74          plt.plot( centroids[i, 0], centroids[i, 1], mark)
75      plt.show()
76
77
```

```
78 if __name__ == "__main__":
79     kmeans(3)
```