Clustering

K-means

Haotian Wang and Yiyan Li

School of Software, Shanghai Jiao Tong University

Introduction to clustering

Clustering

Definition

A cluster is a collection of objects which are "similar" between them and are "dissimilar" to the objects belonging to other clusters.

Definition

Clustering is the algorithm that recognizes clusters from a given data set.

Clustering

Definition

A cluster is a collection of objects which are "similar" between them and are "dissimilar" to the objects belonging to other clusters.

Definition

Clustering is the algorithm that recognizes clusters from a given data set.

Part of common application domains in which the clustering problem arises are as follows:

- Multimedia Data Analysis
- Responding to public health crises
- Intermediate Step for other fundamental data mining problems
- Intelligent Transportation

Face clustering

Face clustering is a very hot research area. In recent years, many streaming media platforms use this technique.

K-means Algorithm

K-means

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.

The most common K-means algorithm was first proposed by Stuart Lloyd of Bell Labs in 1957.

K-means

The objective function to minimize is the 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

Definition

Cluster centroid 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.

Lemma

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:

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

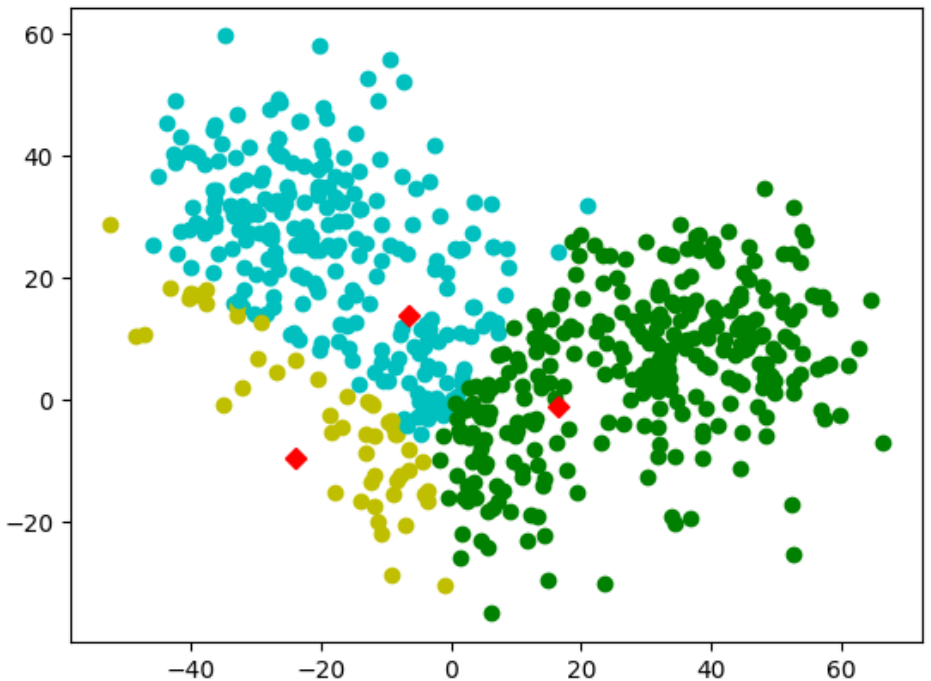
Toward a K-means Algorithm

The k-means algorithm iteratively calculates the sum of distance within a cluster and updates the partition.

1. Arbitrarily choose and initial k centroids $\mathcal{C} = \{c_1, c_2 \ldots c_k\}$
2. For each $i \in \{1, 2 \ldots k\}$, 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 $i \in \{1, 2 \ldots k\}$, 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 $\mathcal{C}$ no longer changes.

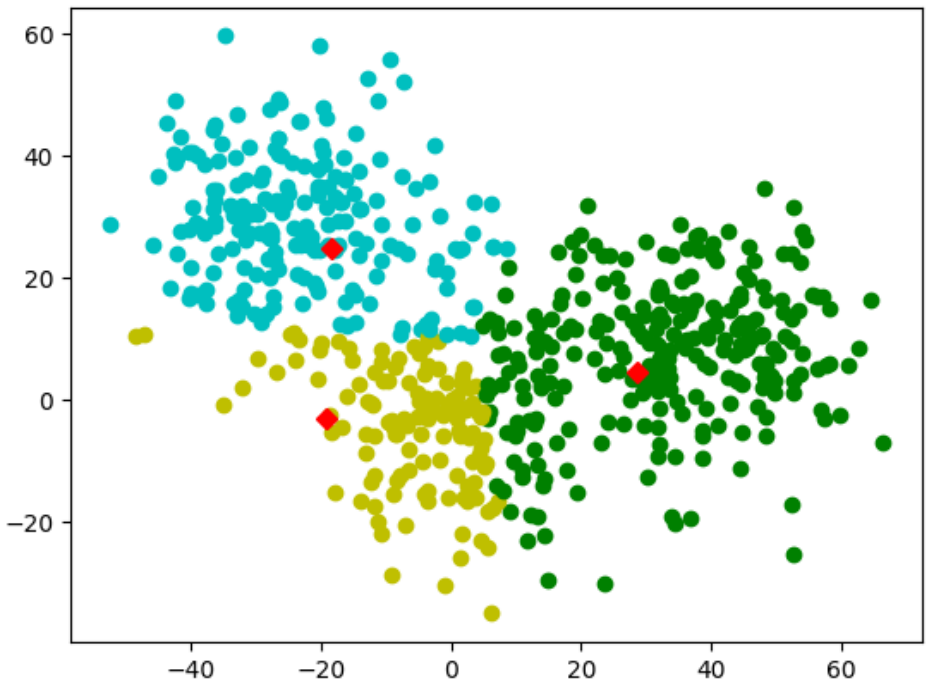
Toward a K-means Algorithm

Iteration = 1



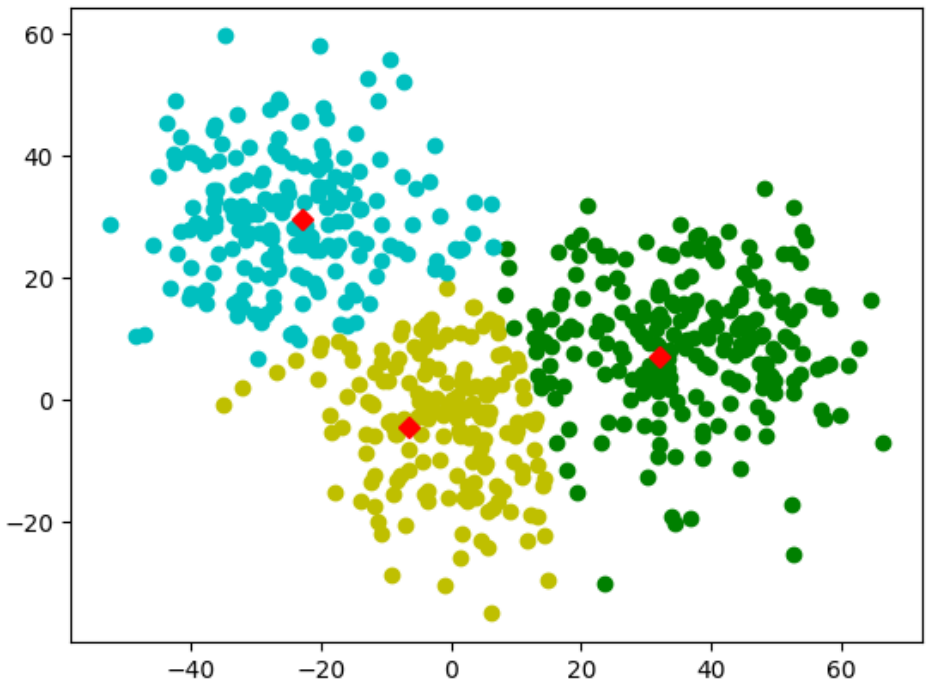
Toward a K-means Algorithm

Iteration = 2



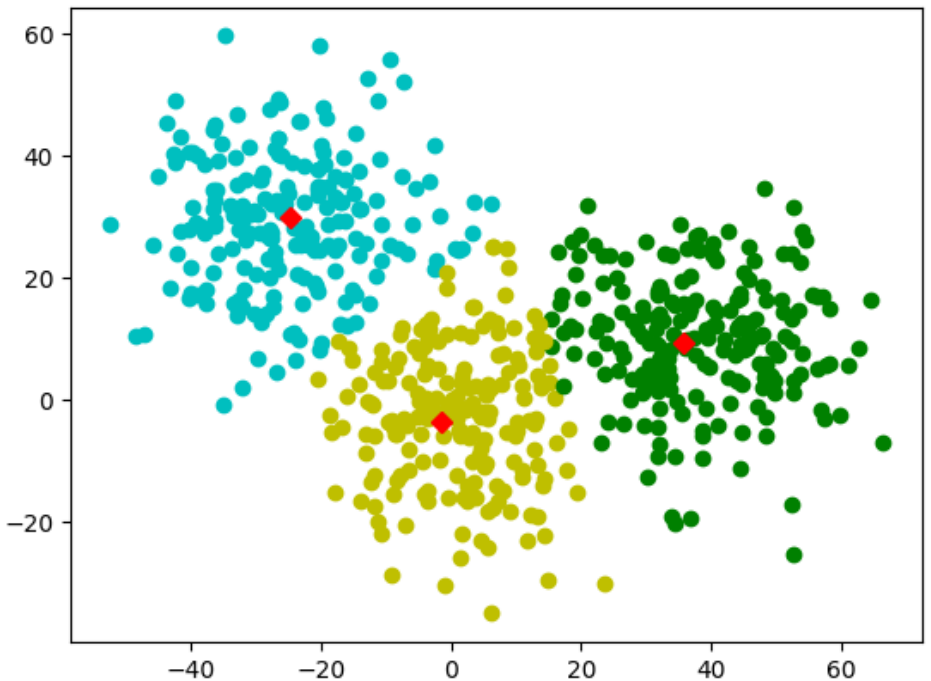
Toward a K-means Algorithm

Iteration = 3



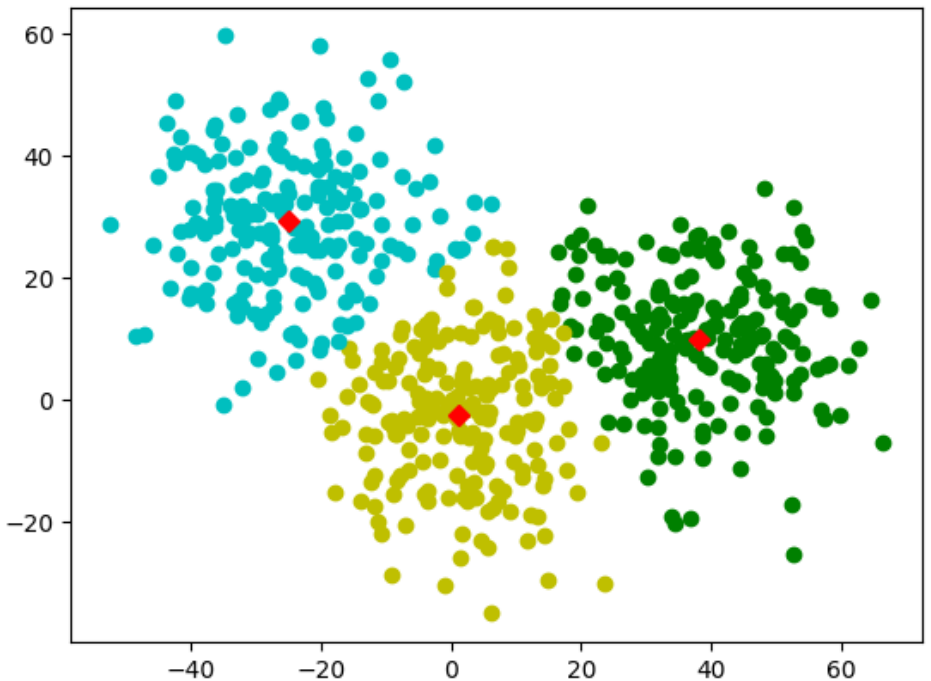
Toward a K-means Algorithm

Iteration = 4



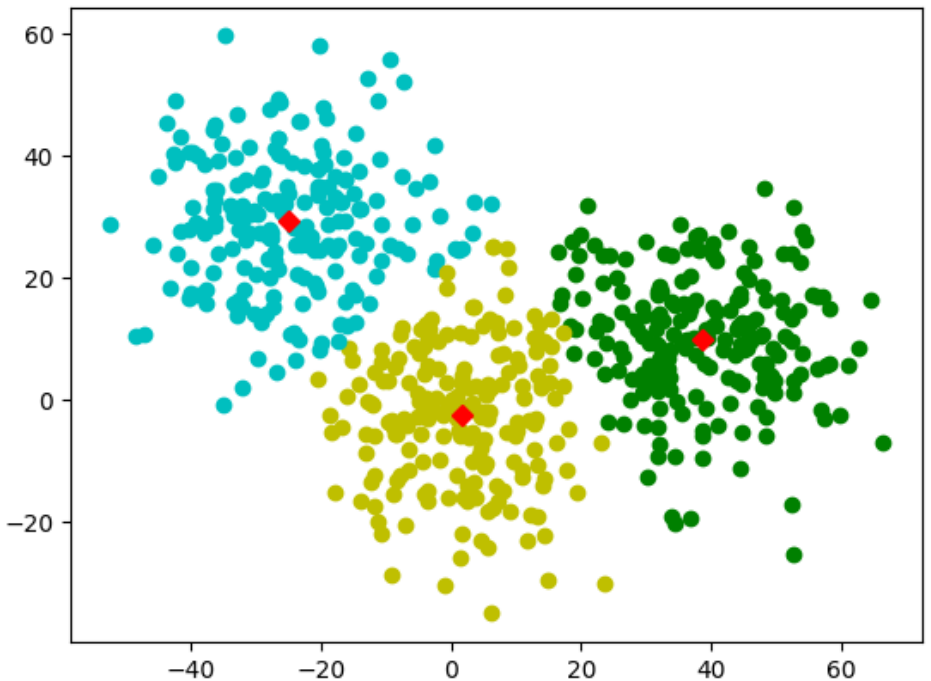
Toward a K-means Algorithm

Iteration = 5



Toward a K-means Algorithm

Iteration = 6



Convergence of K-means

Proof.

During the course of the k-means algorithm, the cost monotonically decreases. □

Convergence of K-means

Proof.

During the course of the k-means algorithm, the cost monotonically decreases. □

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.

Convergence of K-means

Proof.

During the course of the k-means algorithm, the cost monotonically decreases. □

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:

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

Convergence of K-means

Proof.

During the course of the k-means algorithm, the cost monotonically decreases. □

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 above:

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

Time Complexity

Proof.

Naive K-means algorithm's time complexity is $O(kni)$ □

Time Complexity

Proof.

Naive K-means algorithm's time complexity is $O(kni)$ □

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.

Time Complexity

Proof.

Naive K-means algorithm's time complexity is $O(kni)$ □

So the total number of operations in one iteration is:

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

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 $O(kni)$.

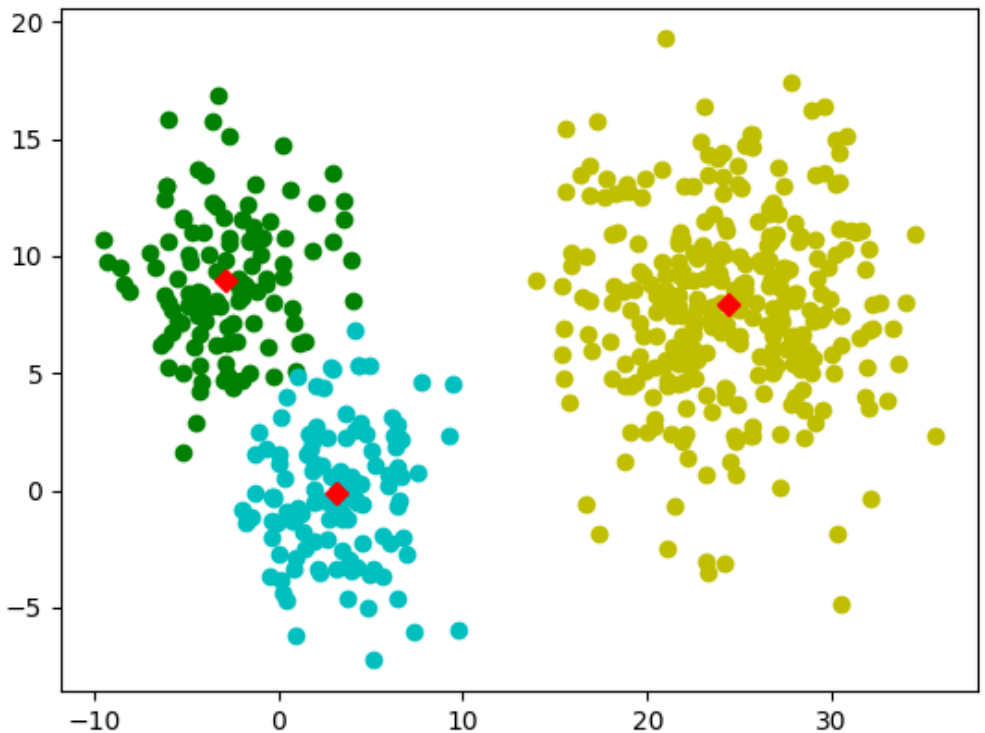
Drawback of K-means

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.

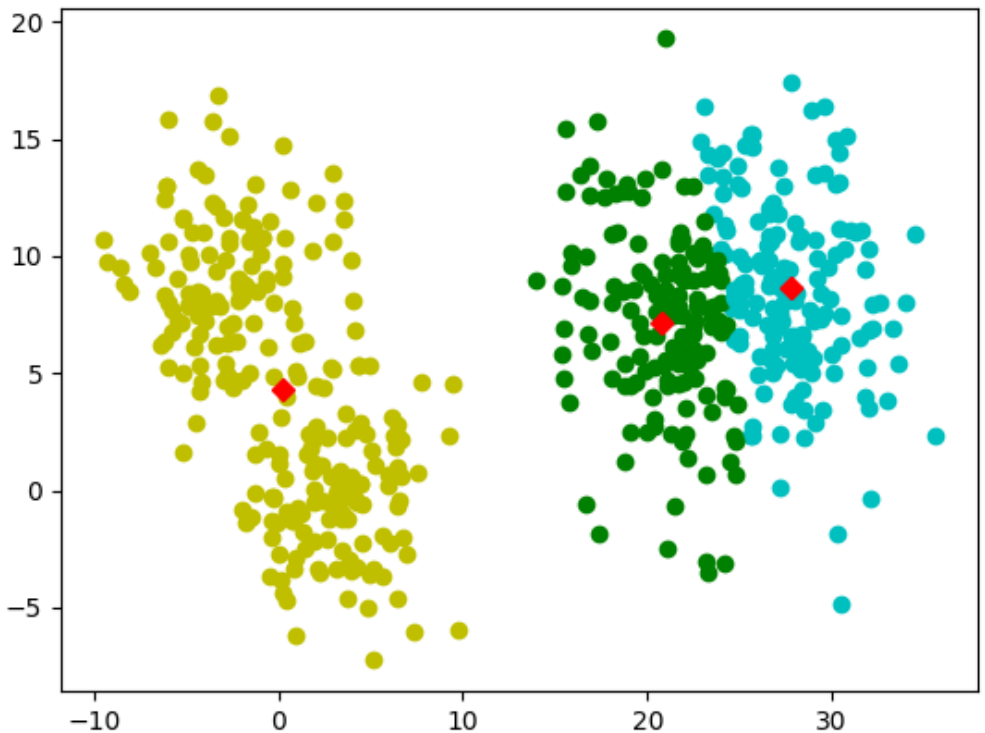
Drawback of K-means

An instance of good clustering result:



Drawback of K-means

An instance of poor clustering result:



K-means optimizations

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.

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. In this way, the results are always good as expected.

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. In this way, the results are always good as expected.

And the updated algorithm does exactly the same in the iteration steps.

K-means++ Algorithm

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++ Algorithm

Proof.

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.

K-means++ Algorithm

Proof.

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

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

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

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

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

Avoid redundant computation

There are n data points in $\mathbb{R}^d$ space and k clusters for partition, each iteration involves $n * k$ distance computations.

There exist many unnecessary calculations in the process of iteration!

Avoid redundant computation

There are n data points in $\mathbb{R}^d$ space and k clusters for partition, each iteration involves $n * k$ distance computations.

There exist many unnecessary calculations in the process of iteration!

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.

Avoid redundant 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.

Lemma

Triangle inequality: For any 3 points x, y, z, $d(x, z) \leq d(x, y) + d(y, z)$.

The only black box property of any distance metrics.

Avoid redundant computation

Lemma

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

Proof.

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. $\square$

Avoid redundant computation

Lemma

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

Proof.

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

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

Proof.

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

Avoid redundant computation

Corollary

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* $d(x,s)$*.*

Avoid redundant computation

Corollary

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* $d(x,s)$*.*

Corollary

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* $u > \frac{1}{2}d(c,s)$ *.*

Avoid redundant computation

Corollary

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

Avoid redundant computation

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:

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

Avoid redundant computation

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:

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

Claim

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

Augmentation detail

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'.

Claim

If $u(x) \leq l(x, c')$, *we don't need to calculate* $d(x, c), d(x, c')$.

Augmentation preparation

Initially, we set $l(x, c) = 0$ for each point x and center c. Then assign each x to its closest initial center.

Each time $d(x, c)$ is computed, set $l(x, c) = d(x, c)$. At last, set upper bounds $u(x) = min_c(d(x, c))$.

Augmentation detail

Then repeate this until convergence.

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 $u(x) \leq s(c(x))$.
3. For each pair of remaining x and c, which satisfy: i) $c \neq c(x)$ and ii) $u(x) > l(x, c)$ and iii) $u(x) > \frac{1}{2} d(c(x), c)$:
   - 3.1 If $r(x) = true$, compute $d(x, c(x))$ and assign $r(x) = false$. Otherwise, $d(x, c(x)) = u(x)$.
   - 3.2 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.

Augmentation detail

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)

Discussion

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

Relax the restriction

Another contribution helps reduce the iteration 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.Their target:

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

How it works

Consider a data point p1 and cluster centroids represented as c_1, c_2..., c_k. We assume that $k' << k$ and there is a candidate cluster list for p_1.

How it works

Consider a data point p1 and cluster centroids represented as c_1, c_2..., c_k. We assume that $k' << k$ and there is a 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.

In real world data rarely makes case 2 happen! That is, the set of top few closest centroids for a data point **remains almost unchange**.

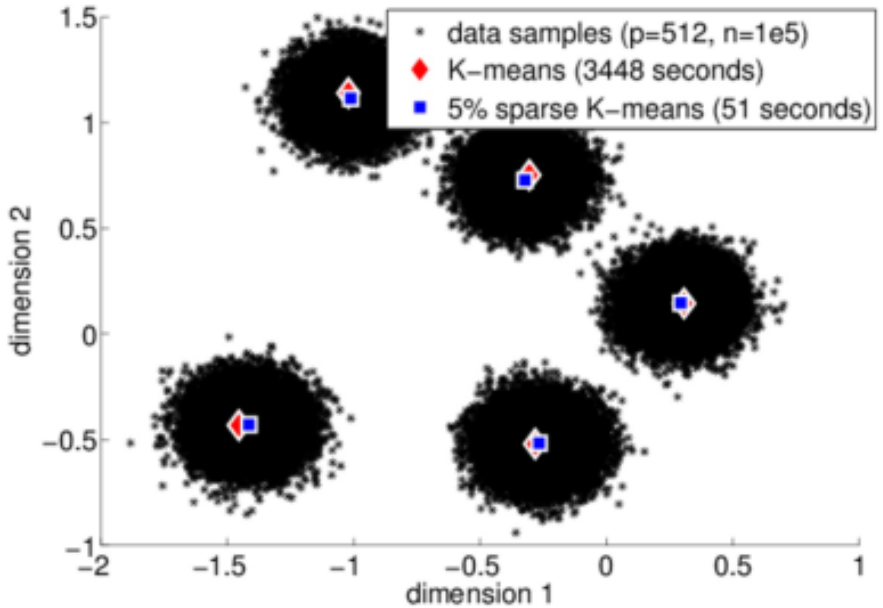
Augmentation analysis

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.

Speed up

From 3448 seconds to 51 seconds, with centroid moving little distance.



Choose a proper K

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.

Choose a proper K

statistical measures: There are several statistical measures available for selecting K. They are calculated with certain assumptions about the underlying distribution of the data.

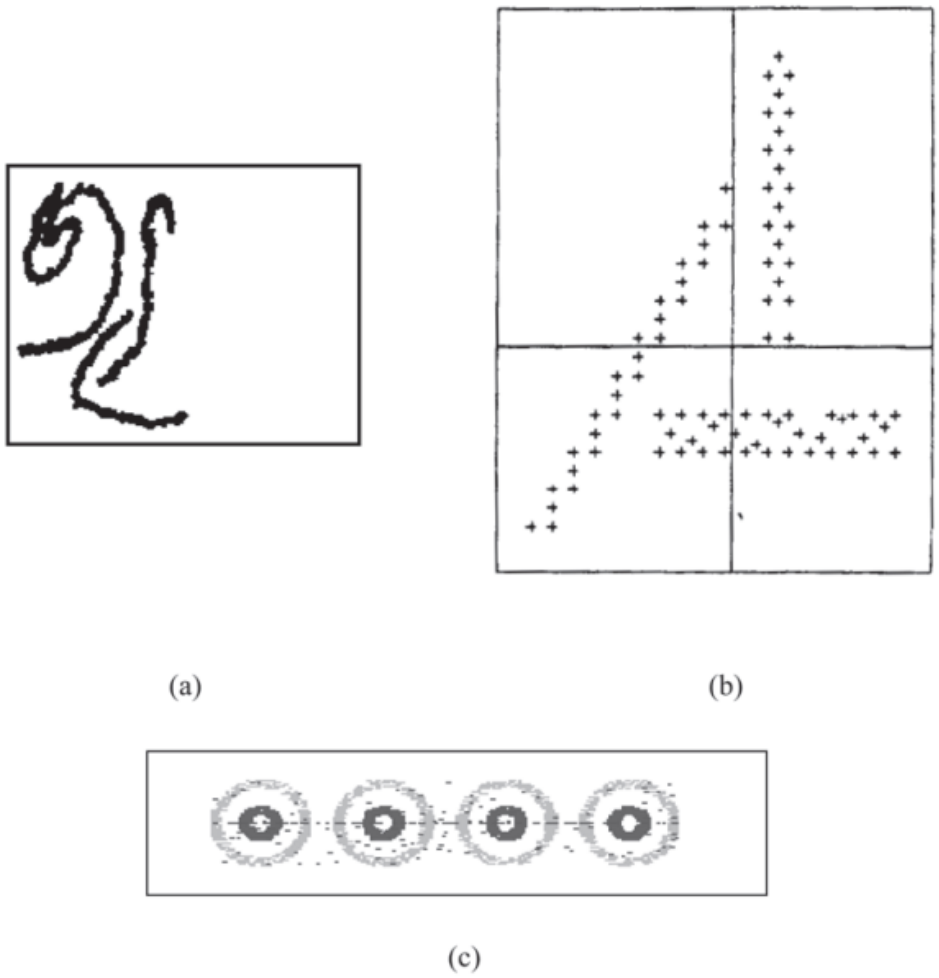
e.g. The Bayesian information criterion is calculated on data sets which are constructed by a set of **Gaussian distributions**.

Choose a proper K

Visualization: Visual verification is applied widely because of its simplicity and explanation possibilities.

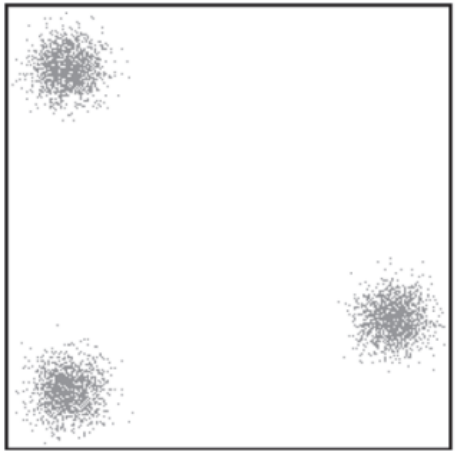
Visualization example

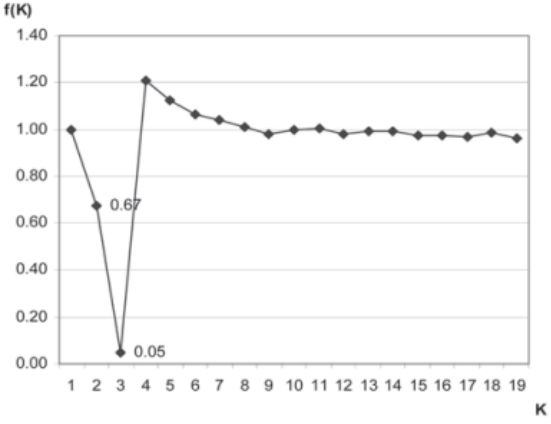
a) 4 clusters, b) 3 clusters, c) 8 clusters.



Visualization example

Combine many methods and create a function f(k) to tune K value.





Key properties

K-means problem is NP Complete

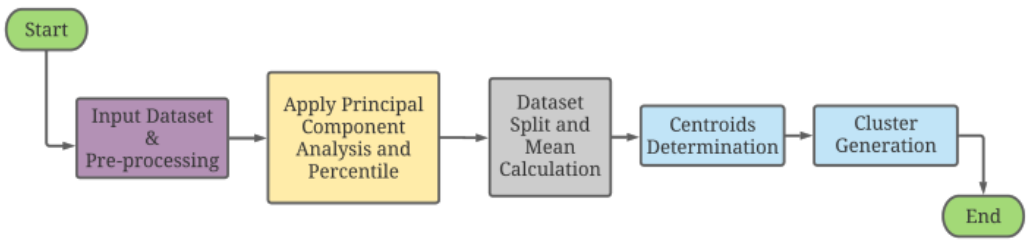
We can prove this using 3-SAT and Exact Cover by 3-Sets respectively. Existing local search algorithms that computes a certain local (and not necessarily global) optimum for this problem are what we really feel interested in.

Application

COVID-19 Dataset

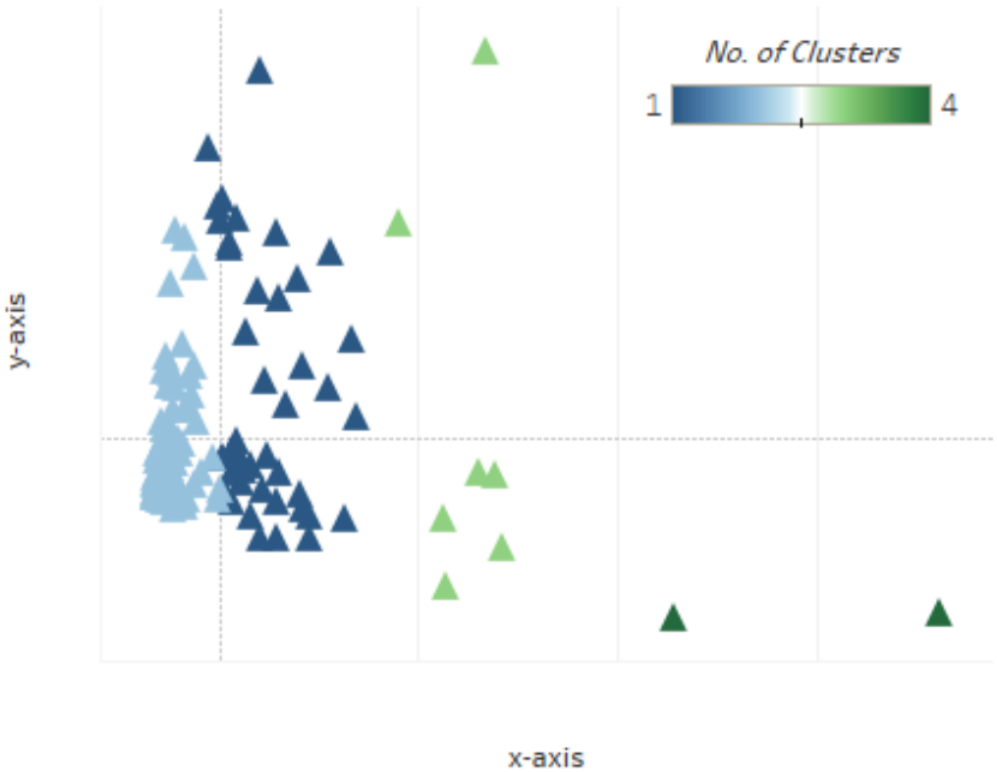
A team uses Principal Component Analysis (PCA) while determining the centroids. With this method to decide on centroids, their method uses much less iterations times and execution time for convergence. Compared with existing K-means++, their algorithm's performance is constant and faster.

Their flowchart of centroids deciding:



COVID-19 Dataset

Their result:



Discussion

Discussion

Discussion

Discussion

Conclusion

Discussion