Lecture Note of K-Means

Kangze Zheng
School of Data and Computer Science, Sun Yat-sen University,
Guangzhou 510006, China
1196401638@qq.com

Abstract

1. Introduction

Image segmentation is one of the most important procedures before image analysis. Besides, the performance of image analysis has great influence on comprehension of the image for machine in computer vision. Therefore, image segmentation plays an significant role in computer vision. According to certain rules, the image is divided into some regions with different characteristics, and we extracted the objects that we are interested in. The corresponding technology and process is called image segmentation. Why we need image segmentation is because we can have great concentration on the main part of the image or the part that we suppose to research on conveniently after image segmentation. There are a number of effective methods for image segmentation, including threshold segmentation, boundary tracking, cluster segmentation and so on.

As the name suggests, the core of threshold segmentation is that each pixel of the image is divided into the target or the background by the appropriate threshold that we select [1]. This method is simple and easy to operate. Moreover it has quite good robustness. However, it requires a certain difference between gray value of the target and the background. Another method, boundary tracking, is starting from the gradient of the image. Via searching and connect the adjacent edge points in turn, the boundary is detected finally [3]. The third method is cluster segmentation. The process of dividing a data set into groups, where the similarity of objects in the same group is high and the similarity of objects in different groups is fairly low, is named cluster. The division is carried out by clustering algorithms. Therefore, different algorithms lead to different divisions. Note that cluster is one kind of unsupervised learning. In other words, cluster need no labels of data. There are four kinds of clustering techniques: partition clustering, hierarchical clustering, grid based clustering and density based clustering. Different techniques have different advantages and are suitable for different circumstances.

In this paper, we mainly introduce one kind of clustering algorithms, that is k-means. K-means is a typical kind of partition clustering. It is applied widely, especially in computer vision. However, it does have some shortcomings. Thus, we present the improved algorithm which is derived from the k-means algorithm. The improved algorithm is k-means++. Finally, we make a comparison between them and the conclusion comes out.

The rest of paper consists of 3 sections. In section 2, we introduce the k-means algorithm and figure out how it works. We also do some experiments about k-means. In section 3, we introduce the improved algorithm named k-means++ and we make a comparison between two algorithms. Through the comparison, the conclusion comes out. Finally, the conclusion of this paper is shown in Section 4.

2. K-Means

This section has five subsections. In Subsection 2.1, we introduce the principle of k-means. Then, we display the pseudocode of the algorithm and figure out how it works in Subsection 2.2. In Subsection 2.3, we list some computational approaches about similarity, which can be used in k-means for similarity calculation. Next in Subsection 2.4, we do some experiments to experience the effect of k-means and make an acquaintance with the usage of it. Finally, we analysis the algorithm in Subsection 2.5.

2.1. Principle

In this subsection, we explain the core of the k-means algorithm as follows.

The constant k in the name "k-means" means the final number of clusters. Initially, select k points randomly as as the centers of k clusters. Then, classify all the points into the most similar cluster by calculating the similarity between each point and each center. After classification, recalculate the center of each cluster. Repeat the process of classification and calculation until all the centers is no longer changed. Finally, the division which each point belongs to and the center of the division are determined. Because the similarity between each point and each center is calculated each loop, the convergence speed of k-means is relatively slow. As for how to calculate the similarity, there are many computational approaches. The most common way is Euclidean distance. In Subsection 2.3, we present some computational approaches for similarity specifically.

2.2. Pseudocode

In this subsection, we show the pseudocode of k-means and figure out how it works.

```
Algorithm 1 K-Means
Input: Sample set $\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$
       Number of clusters $k$
Output: Cluster $\mathbf{C} = \{C_1, C_2, ..., C_n\}$
Procedure:
 1: Randomly choose $k$ samples from $\mathbf{D}$ as an initial
    mean vector group $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_n\}$;
 2: repeat
 3:     Let $C_i = \emptyset \ (1 \le i \le k)$;
 4:     for $j = 1, 2, ..., m$ do
 5:         Calculate the distance of sample $\mathbf{x}_j$ and each
 6:         mean vector $\boldsymbol{\mu}_i \ (1 \le i \le k)$: $d_{ji} =$
 7:         $\|\mathbf{x}_j - \boldsymbol{\mu}_i\|_2$;
 8:         Determine the cluster label of $\mathbf{x}_j$ according
 9:         to the nearest mean vector: $\lambda_j =$
10:         $\arg\min_{i \in \{1,2,...,k\}} d_{ji}$;
11:         Add sample $\mathbf{x}_j$ into the corresponding
12:         cluster division $C_{\lambda_j} = C_{\lambda_j} \cup \{\mathbf{x}_j\}$;
13:     end for
14:     for $i = 1, 2, ..., k$ do
15:         Calculate new mean vector: $\boldsymbol{\mu}'_i = \frac{1}{|C_i|}$
16:         $\sum_{\mathbf{x} \in C_i} \mathbf{x}$;
17:         if $\boldsymbol{\mu}_i \ne \boldsymbol{\mu}'_i$ then
18:             Update the current mean vector $\boldsymbol{\mu}_i$ to
19:             $\boldsymbol{\mu}'_i$;
20:         else
21:             Keep the current mean vector
22:             unchanged;
23:         end if
24:     end for
25: until The current mean vectors have not changed
```

According to algorithm (1), there are a data set D of vectors ($D = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$) which represent features of objects and a number k which is the number of divisions after clustering as input. At first, we select k samples(e.g., vectors in the input data set) randomly as an initial mean vector group $\{\mu_1, \mu_2, ..., \mu_k\}$. Then, we start the following procedure. At step 1, initialize a set group $\mathbf{C} = \{C_1, C_2, ..., C_k\}$, where $C_i = \emptyset \ (1 \le i \le k)$. At step 2, calculate out the distance d_{ji} of each sample $\mathbf{x}_j$ and each mean vector μ_i by equation $d_{ji} = \|\mathbf{x}_j - \mu_i\|_2$. At step 3, for each sample $\mathbf{x}_j$, find the index λ_j of a certain mean vector which has the shortest distance from $\mathbf{x}_j$. At step 4, add sample $\mathbf{x}_j \ (1 \le j \le m)$ into the set C_{λ_j}. At step 5, for each set $C_i \ (1 \le i \le k)$, calculate the new mean value μ'_i, and update the current mean value if $\boldsymbol{\mu}_i \ne \mu'_i$. Repeat the above procedure until all the mean vectors keep unchanged.

2.3. Similarity computational Approaches

In this subsection, we represent some similarity computational approaches. Different approaches have different influence on final clusters, which is meant that different similarity approaches lead to different clusters. In addition, in a certain circumstance, a certain computational approach may perform better that other approaches. Therefore, you can adopt different approaches in experiment and compare their performances. After that, choose the best one to apply in experiment.

Euclidean distance is the most common and intuitive method. Assume that there are two n-dimension vectors $\mathbf{v}_1 = (v_{11}, v_{12}, ..., v_{1n})$ and $\mathbf{v}_2 = (v_{21}, v_{22}, ..., v_{2n})$. Their Euclidean distance is represent as follows:

$$d^2 = \sqrt{(\mathbf{v}_1 - \mathbf{v}_2)(\mathbf{v}_1 - \mathbf{v}_2)'}.$$

Manhattan distance is the sum of absolute distance on each axis in standard coordinate axis. Assume that there are two n-dimension vectors $\mathbf{v}_1 = (v_{11}, v_{12}, ..., v_{1n})$ and $\mathbf{v}_2 = (v_{21}, v_{22}, ..., v_{2n})$. Their Manhattan distance is represent as follows:

$$d = \sum_{i=1}^{n} |v_{1i} - v_{2i}|.$$

Cosine distance refers the cosine value of angle between two vectors. When angle is small, the directions of two vectors are almost the same and the cosine value of angle between two vectors is small. Thus, cosine distance is defined as 1 minus cosine value of angle between two vectors. Assume that there are two n-dimension vectors $\mathbf{v}_1 = (v_{11}, v_{12}, ..., v_{1n})$ and

$\mathbf{v}_2 = (v_{21}, v_{22}, ..., v_{2n})$. Their cosine distance is represent as follows:

$$d = 1 - \frac{\mathbf{v}_1\mathbf{v}_2'}{\sqrt{(\mathbf{v}_1\mathbf{v}_1' + \mathbf{v}_2\mathbf{v}_2')}}.$$

Correlation distance refers correlation coefficient, which is a method to measure the correlation of random variables X and Y. The larger the absolute value of correlation coefficient of random variables X and Y is, the higher the correlation of random variables X and Y is. Besides, the value range of correlation coefficient is in $[-1, 1]$. Thus, correlation distance is defined as 1 minus correlation coefficient. Assume that there are two n-dimension vectors $\mathbf{v}_1 = (v_{11}, v_{12}, ..., v_{1n})$ and $\mathbf{v}_2 = (v_{21}, v_{22}, ..., v_{2n})$. Their correlation distance is represent as follows [2]:

$$d = 1 - \frac{(\mathbf{v}_1\mathbf{v}_1')(\mathbf{v}_1\mathbf{v}_1')}{\sqrt{(\mathbf{v}_1\mathbf{v}_1') - (\mathbf{v}_2\mathbf{v}_2')}\sqrt{(\mathbf{v}_1\mathbf{v}_1') - (\mathbf{v}_2\mathbf{v}_2')}}.$$

2.4. Experiment

In this subsection, we experience the effect of k-means firstly. Then, we apply the k-means algorithm to a gray image and a color image for image segmentation, respectively.

We use Python to simulate the algorithm. Following is the program.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn import metrics
n_samples = 5000
random_state = 10
centers = 5
x, y = make_blobs(centers=centers,n_samples=n_samples, random_state=random_state)
plt.figure(1)
plt.subplot(1,1,1)
plt.scatter(x[:,0],x[:,1],c=y)
plt.title('initial data')

inertia = []
calinski_harabasz_score = []
for i in range(2,10):
    km = KMeans(n_clusters=i,n_init=10,init='k-means++').fit(x)
    y_pred=km.predict(x)
    center_=km.cluster_centers_
    inertia.append([i,km.inertia_])
    z=metrics.calinski_harabasz_score(x, y_pred)
    calinski_harabasz_score.append([i,z])
    plt.figure(i)
    plt.subplot(1,1,1)
    plt.scatter(x[:,0],x[:,1],c=y_pred)
    plt.scatter(center_[:,0],center_[:,1],color='red')
    plt.title('k_clusters=%s'%i)

inertia=np.array(inertia)
plt.figure(10)
plt.subplot(1,1,1)
plt.plot(inertia[:, 0], inertia[:, 1])
plt.title('SSE - k_clusters')
plt.figure(11)
plt.subplot(1,1,1)
calinski_harabasz_score = np.array(calinski_harabasz_score)
plt.plot(calinski_harabasz_score[:, 0], calinski_harabasz_score[:, 1])
plt.title('calinski_harabasz_score - k_clusters')
plt.show()
```

We use sklearn module to generate 500 random samples, which belong to 5 clusters, respectively. Then, we use sklearn module to perform the k-means algorithm on the data set, which is generated just now. We try to divide the data set into 2 clusters, 3 clusters, ..., 9 clusters. The results are shown in Figure 1.

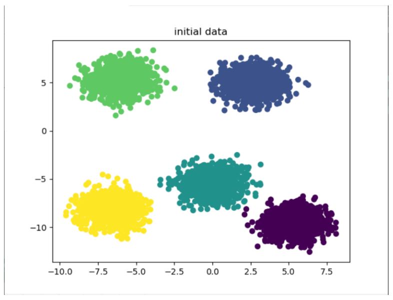
Now that the samples are divided, we need to figure out which k is better. There are two indicators to measure the classification. The first one is within-cluster sum of squared errors, which is also known as cluster inertia (CI). CI is formulated as:

$$CI = \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|_2^2,$$

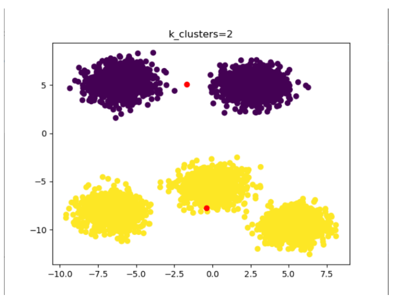
where i, j denote indices; k denotes the number of clusters; C_i denotes a cluster; $\mathbf{x}_j$ denotes a sample; and $\boldsymbol{\mu}_i$ denotes the mean vector of C_i. The second one is Calinski-Harabasz index (CHI), which is formulated as:

$$CHI = \frac{tr(B_k)}{tr(W_k)} \cdot \frac{m-k}{k-1},$$

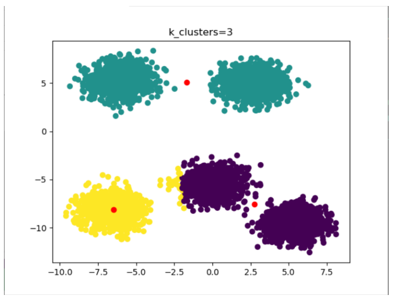
where m denotes the number of samples; k is the number of clusters; B_k denotes between-class scatter matrix; W_k denotes within-class scatter matrix; and tr denotes the track of the matrix. With these two indicators, we measure the performances of 2-means, 3-means, ..., 9-means. The graphs of two indicators are shown in Figure 2. From Figure 2(a), it is evident that CI is getting lower when k is getting bigger. However, it does not mean that the algorithm performs well with a large k. Suppose that k is large enough to be equal to the number of samples m. The CI will be zero, but it does not make sense. Usually, we choose the inflection point (e.g., $k = 5$). From Figure 2(b), it is evident that the point with $k = 5$ is a inflection point with max CHI value. Generally, the larger CHI value is, the better k-means performs. Because when CHI value is larger, within-class covariance is smaller and between-class covariance is larger, which means the current division is better.



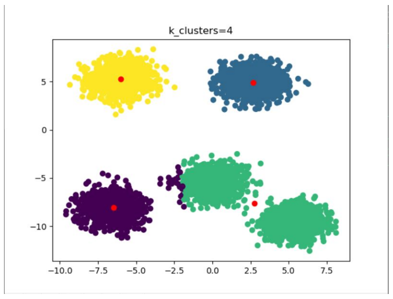
(a) initial



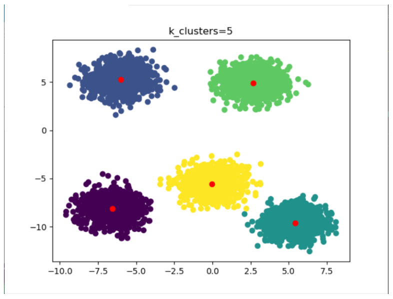
(b) $k = 2$



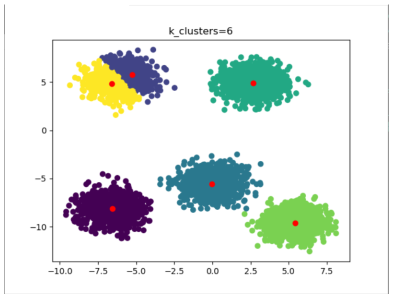
(c) $k = 3$



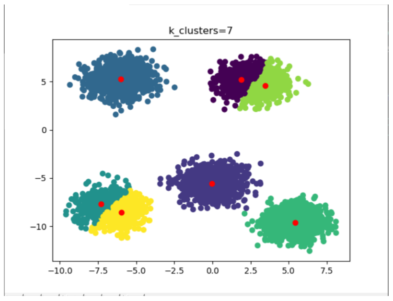
(d) $k = 4$



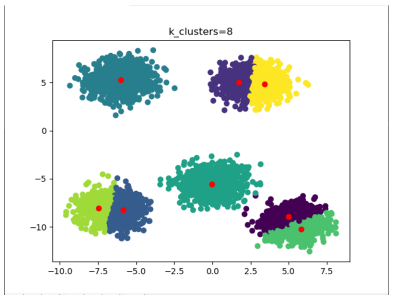
(e) $k = 5$



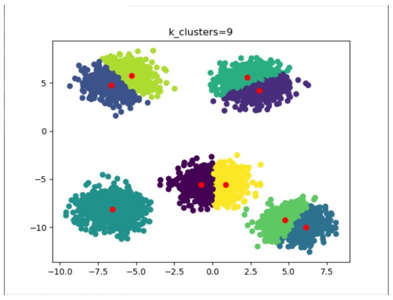
(f) $k = 6$



(g) $k = 7$



(h) $k = 8$

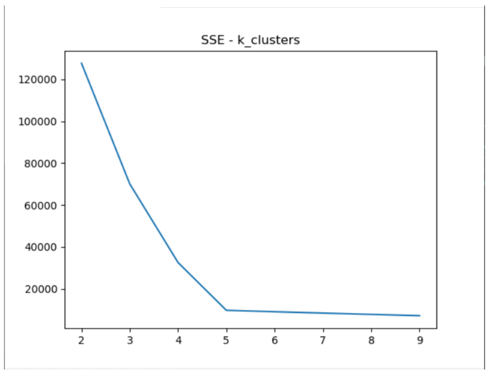


(i) $k = 9$

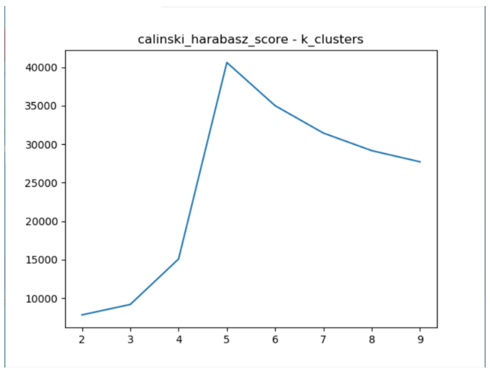
Figure 1: Random samples are shown in (a). (b) $\sim$ (i) are the results of 2-means $\sim$ 9-means.

After experiencing the effect of algorithm intuitively, we realize that the essence of k-means is to divide each point into the nearest cluster, of which center is probably changed in each loop. Then, we apply the algorithm to the practical experiment. We first implement image segmentation for a gray image. Following is the program.

```python
import cv2
import matplotlib.pyplot as plt
import numpy as np


def seg_kmeans_gray(img, k):
    img_flat = img.reshape((img.shape[0] * img.
        shape[1], 1))
    img_flat = np.float32(img_flat)

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.
        TermCriteria_MAX_ITER, 20, 0.5)
    flags = cv2.KMEANS_RANDOM_CENTERS

    compactness, labels, centers = cv2.kmeans(
        img_flat, k, None, criteria, 10, flags)

    img_output = labels.reshape((img.shape[0],
        img.shape[1]))
    plt.figure()
    plt.subplot(111), plt.imshow(img_output, '
        gray'), plt.title('k=%s'%k)

if __name__ == '__main__':
    img = cv2.imread('LENA.png', cv2.
        IMREAD_GRAYSCALE)
    plt.figure()
    plt.subplot(111), plt.imshow(img, 'gray'),
        plt.title('initial')
```

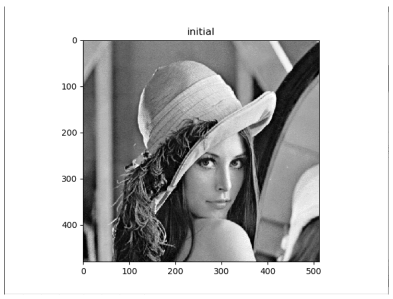


(a) cluster inertia (CI)

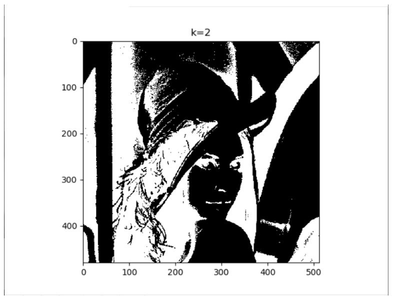


(b) Calinski-Harabasz index (CHI)

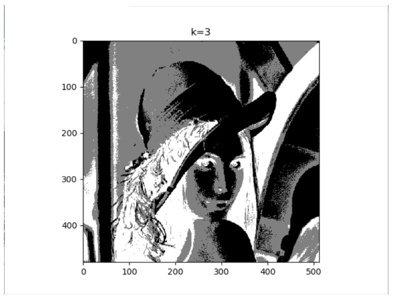
Figure 2: The graphs of CI and CHI.



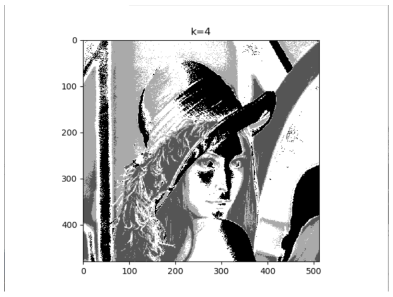
(a) initial



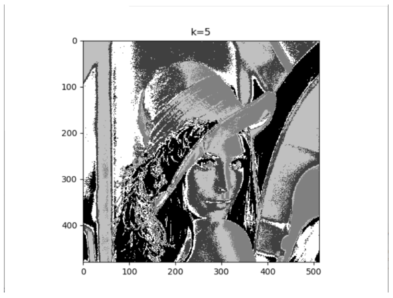
(b) $k = 2$



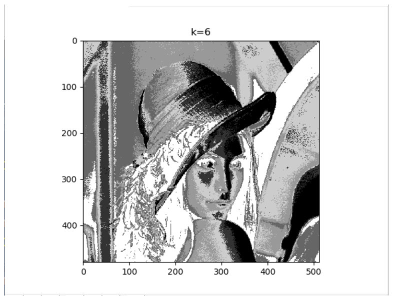
(c) $k = 3$



(d) $k = 4$



(e) $k = 5$



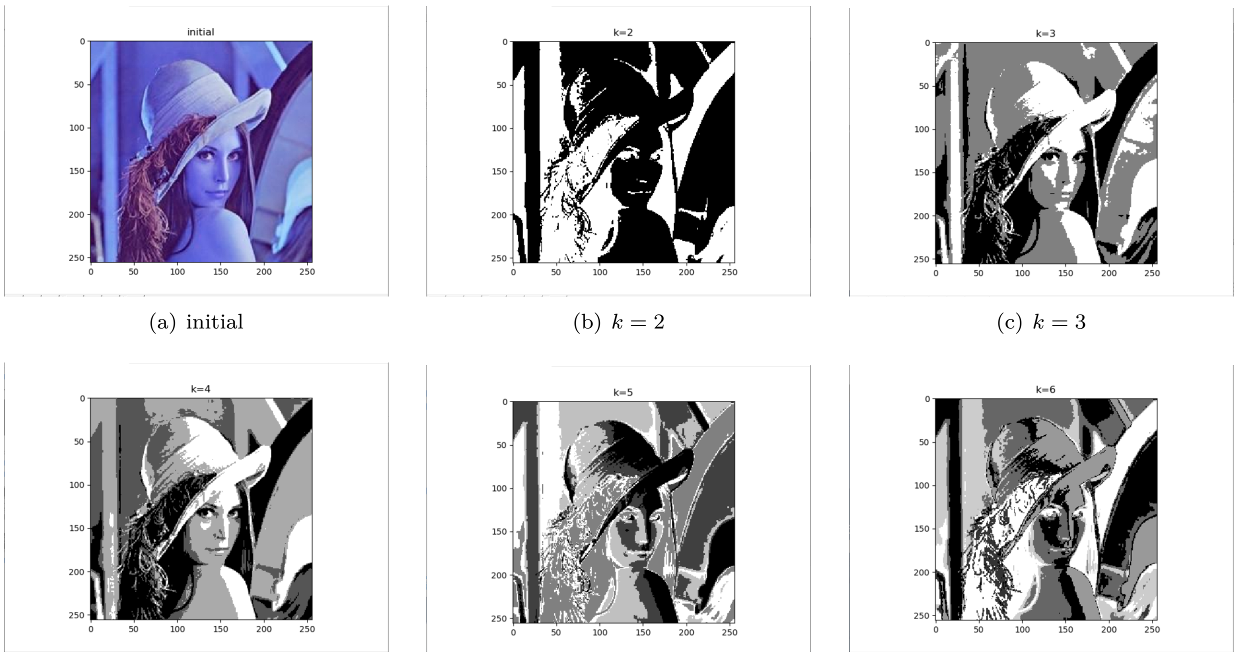
(f) $k = 6$

Figure 3: Image segmentation for a gray image.

```
    for k in range(2, 7):
        seg_kmeans_gray(img, k)
    plt.show()
```

We use cv2 module to read a gray image. As known to us, an gray image is made up of a 2-dimension matrix with gray values. We change the matrix into a 1-dimension vector, in which each gray value is regarded as a point with its feature. Then, we apply the algorithm to divide these points, which makes each point labeled. Points with the same label have the same color, otherwise they have different color. In this way, we implement image segmentation of a gray image. The corresponding results are displayed in Figure 3. It is found that the color of Lena's face and shoulder is the same, which is meant that the algorithm classifies them into the same cluster. Evidently, this is right.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```



(a) initial (b) $k = 2$ (c) $k = 3$

(d) $k = 4$ (e) $k = 5$ (f) $k = 6$

Figure 4: Image segmentation for a color image.

```
def seg_kmeans_color(img, k):
    b, g, r = cv2.split(img)
    img = cv2.merge([r, g, b])

    img_flat = img.reshape((img.shape[0] * img.
        shape[1], 3))
    img_flat = np.float32(img_flat)

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.
        TermCriteria_MAX_ITER, 20, 0.5)
    flags = cv2.KMEANS_RANDOM_CENTERS

    compactness, labels, centers = cv2.kmeans(
        img_flat, k, None, criteria, 10, flags)

    img_output = labels.reshape((img.shape[0],
        img.shape[1]))
    plt.figure()
    plt.subplot(111), plt.imshow(img_output, '
        gray'), plt.title('k=%s'%k)


if __name__ == "__main__":
    img = cv2.imread('LENA.jpg', cv2.IMREAD_COLOR
        )
    plt.figure()
    plt.subplot(111), plt.imshow(img), plt.title(
        'initial')

    for k in range(2, 7):
        seg_kmeans_color(img, k);
    plt.show()
```

For a color image, the corresponding is 3-dimension. When we change it into a vector, each value of the vector is a combination (i.e., vector) of 3 numbers which represent the values of red, green and blue channels. Each pixel of the color image has its combination of the value of red, green and blue channels, which is the feature. Based on these feature, k-means classifies these pixels. The corresponding results are shown in Figure 4.

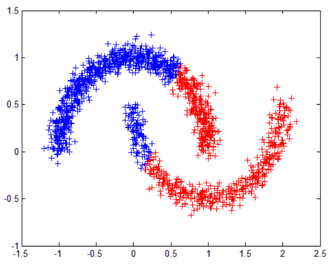
2.5. Analysis

In this subsection, we analyze the complexity, pros and cons of k-means.

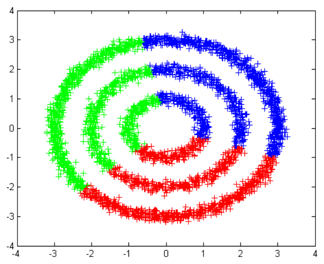
Assume the number of samples is m, the number of cluster is k, the number of loops is t. In each loop, we need to calculate the distance of each sample and each cluster. Thus, the time complexity of the algorithm is $O(mkt)$. Because we need to store the samples and the centers of clusters, the space complexity of the algorithm is $O(k+m)$. Considering that $k \ll m$ generally, the more general space complexity is $O(m)$.

The pros of the algorithm are evident. Firstly, the algorithm is simple and fast. Besides, it is easy to implement. In addition, it finds cluster centers that minimize conditional variance.

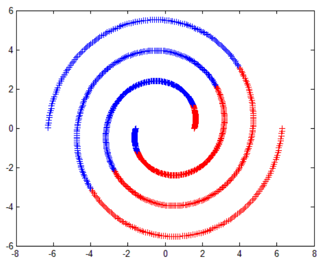
In the same time, the cons of the algorithm are evident. It is time-consuming to choose an appropriate k



(a)



(b)



(c)

Figure 5: Data sets with non spherical distribution.

value. However, the value of k makes difference. Besides, it is sensitive to outliers. Once there exist some outliers, the algorithm performs fairly bad. In addition, the algorithm only guarantees the local optimal solution, not the global optimal solution. Lastly, it fits bad for data sets with non spherical distribution, such as Figure 6

Finally, we make a summary for this subsection as follows.

Complexity:

- Time complexity is $O(kmt)$.
- Space complexity is $O(m)$.

Pros:

- Difficult to select an appropriate k.
- Sensitive to outliers.
- Finds cluster centers that minimize conditional variance.

Pros:

- Simple and fast.
- Easy to implement.
- Only guarantees the local optimal solution, not the global optimal solution.
- Fits bad for data sets with non spherical distribution.

3. K-Means++

In this section, we introduce an improved algorithm named k-means++ which is derived from k-means. We also provide the principle of k-means++ in Subsection 3.1 and display the pseudocode in Subsection 3.2. Then, we make a comparison between k-means and k-means++ in Subsection 3.3.

3.1. Principle

In this subsection, we explain the core of the k-means++ algorithm as follows.

The traditional k-means algorithm is sensitive to the initial centers of clusters, and the clustering results fluctuate with different initial centers. To overcome this shortcoming, some researchers propose a new method to select initial centers of clusters based on data distribution, which is known as the k-means++ algorithm [4]. The difference between k-means and k-means++ is that k-means++ does not select k samples randomly as the initial centers of clusters. The loop part of k-means is the same as k-means++'s one. The rule that the initial selection of k-means++ obeys is that the distance between the initial centers of clusters should be as far as possible. Assume that $n \quad (0 < n < k)$ initial centers of clusters have been selected. Then, the sample which is further away from the current n centers will be chosen as $n + 1$-th center with higher probability. The first center is chosen in all the samples with the same probability. This is the core of k-means++.

3.2. Pseudocode

In this subsection, we display the pseudocode of k-means++ .

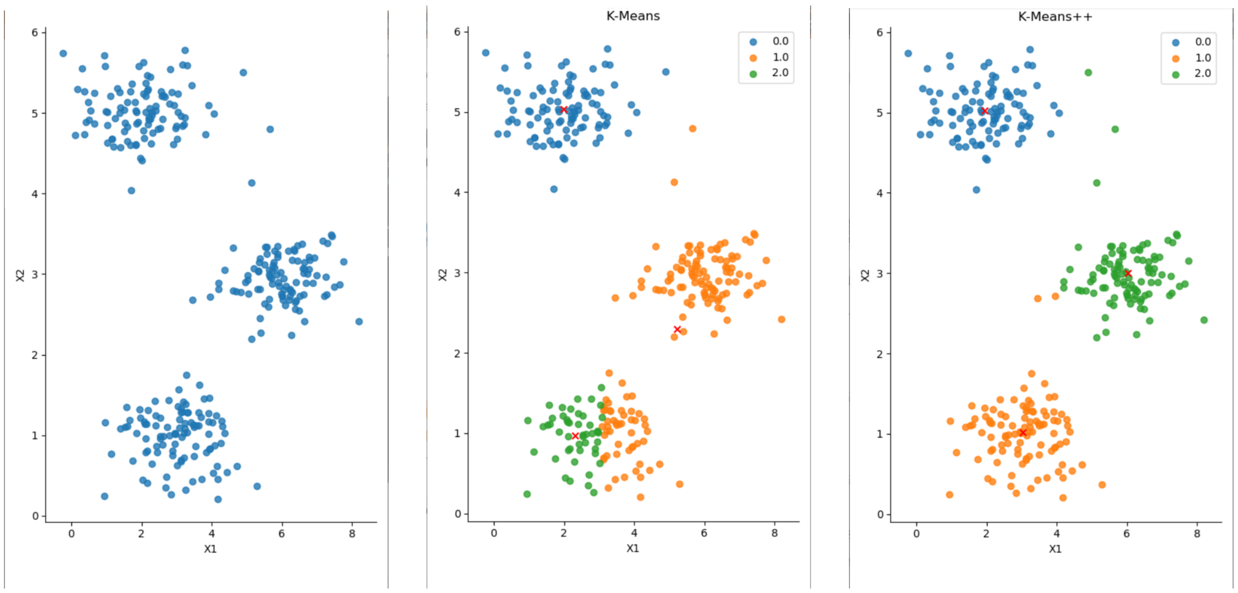
It is evident that the k-means++ algorithm is quite similar to the k-means algorithm except the selections of centers of clusters.

3.3. Comparison

In this subsection, we make a comparison between k-means and k-means++. At first, we generate a fixed data set, which is shown in Figure 6(a). As shown in Figure 6(a), the data set can be divided into 3 clusters roughly. Then, we apply the k-means algorithm and k-means++ algorithm on the data set with $k = 3$. In

Table 1: Comparison of CI values of two algorithms

The number of loops	CI of k-means	CI of k-means++
2	872.47	266.82
3	851.15	266.66
4	690.14	266.66
6	276.68	266.66
8	266.66	266.66
8	266.66	266.66



(a) initial (b) k-mean (c) k-means++

Figure 6: The effect of k-means and k-means++ on the data set when the number of loops is 4.

Section 2, we present two indicators of the algorithm, which is CI and CHI. Here, we compare the CI values of two algorithms, which is displayed in Table 1. From Table 1, as we can see, the CI value of k-means algorithm converges to the minimum 266.66 when the number of loops is 8. However, the CI value of k-means++ converges to the minimum 266.66 when the number of loops is 3. Therefore, it is proved that the k-means++ algorithm is more efficient that the k-means algorithm. The convergence speed of CI value of k-means++ is much faster and more stable. However, it is admitted that the effect of two algorithms are the same when the number of loops is large enough. Finally, we display the effect of two algorithms on data set when the number of loops is 4 in Figure 6. It is evident that the effect of k-means++ is better.

4. Conclusion

In this paper, we introduce a clustering algorithm, which is called k-means. We present the principle and pseudocode of the k-means algorithm. Then, we do

```
Algorithm 2 K-Means++
Input: Sample set $\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$
       Number of clusters $k$
Output: Cluster $\mathbf{C} = \{C_1, C_2, ..., C_n\}$
Procedure:
 1: Randomly choose 1 sample from $\mathbf{D}$ as an initial mean vector $\boldsymbol{\mu}_1$ of cluster $C_1$;
 2: for $i = 1, 2, ..., k$ do
 3:     for $j = 1, 2, ..., m$ do
 4:         Calculate the distance of sample $\mathbf{x}_j$ and the
 5:         current determined centers;
 6:         The nearest distance is recorded as $d_j$;
 7:     end for
 8:     Select a sample $\mathbf{x}_j$ $(1 \le j \le m)$ from $D$ as the
 9:     mean vector $\boldsymbol{\mu}_i$ of cluster $C_i$ according to the
10:     rule that the smaller $d_j$, the higher the
11:     probability;
12: end for
13: repeat
14:     Let $C_i = \emptyset$ $(1 \le i \le k)$;
15:     for $j = 1, 2, ..., m$ do
16:         Calculate the distance of sample $\mathbf{x}_j$ and each
17:         mean vector $\boldsymbol{\mu}_i$ $(1 \le i \le k)$: $d_{ji} =$
18:         $\|\mathbf{x}_j - \boldsymbol{\mu}_i\|_2$;
19:         Determine the cluster label of $\mathbf{x}_j$ according
20:         to the nearest mean vector: $\lambda_j =$
21:         $\arg\min_{i \in \{1,2,...,k\}} d_{ji}$;
22:         Add sample $\mathbf{x}_j$ into the corresponding
23:         cluster division: $C_{\lambda_j} = C_{\lambda_j} \cup \{\mathbf{x}_j\}$;
24:     end for
25:     for $i = 1, 2, ..., k$ do
26:         Calculate new mean vector: $\boldsymbol{\mu}'_i = \frac{1}{|C_i|}$
27:         $\sum_{\mathbf{x} \in C_i} \mathbf{x}$;
28:         if $\boldsymbol{\mu}_i \ne \boldsymbol{\mu}'_i$ then
29:             Update the current mean vector $\boldsymbol{\mu}_i$ to
30:             $\boldsymbol{\mu}'_i$;
31:         else
32:             Keep the current mean vector
33:             unchanged;
34:         end if
35:     end for
36: until The current mean vectors have not changed
```

some experiments about k-means and experience the effect of the algorithm. Besides, we analyze the complexity, pros and cons of the k-means algorithm. To overcome the shortcoming of k-means. We also introduce the improved algorithm which is named k-means++. We also present the principle and pseudocode of the k-means++ algorithm. Finally, we make a comparison between two algorithms and make a conclusion. We conclude that the k-means++ algorithm is more efficient that the k-means algorithm, but the effect of two algorithms are the same when the number of loops is large enough.

References

[1] Salem Saleh Al-Amri, N. V. Kalyankar, and SD Khamitkar. Image segmentation by using threshold techniques. Computer Science, 2(5), 2010.

[2] Bin Lin. Experimental comparison of k-means text clustering by varied distance calculation methods. Journal of Fujian University of Technology, 2016.

[3] P. A. V Miranda. Riverbed: A novel user-steered image segmentation method based on optimum boundary tracking. IEEE Transactions on Image Processing, 21(6):p.3042–3052, 2012.

[4] Xue Mei Wang and Jin Bo Wang. Research and improvement on k-means clustering algorithm. Computer & Digital Engineering, 756-759:3231–3235, 2013.