Name: Jianyuan Lu GT ID: 903633385

1.Clustering

1.1

Solution:

In general, J is the linear equation of r^{ij}, and for each i in m, the minimizing J process is independent, so we could let r^{ij} be fixed, and the expression will be the same for each i in m.

So $J = \sum_{i=1}^{m} r^{ij} \left\| x^i - \mu^j \right\|^2$ for this fixed r^{ij}.

$$dJ = \sum_{i=1}^{m} r^{ij} \left[d\left(x^i - \mu^j\right)^T \left(x^i - \mu^j\right) + \left(x^i - \mu^j\right)^T d\left(x^i - \mu^j\right) \right]$$

$$= \sum_{i=1}^{m} r^{ij} \left[-d\left(\mu^j\right)^T \left(x^i - \mu^j\right) - \left(x^i - \mu^j\right)^T d\left(\mu^j\right) \right]$$

Since dJ is a scalar, so $dJ = tr(dJ)$

So, the upper equation can equate following:

$$dJ = tr(dJ)$$

$$= \mathbf{tr}\left(\sum_{i=1}^{m} r^{ij} \left[-d\left(\mu^j\right)^T \left(x^i - \mu^j\right) - \left(x^i - \mu^j\right)^T d\left(\mu^j\right) \right] \right)$$

$$= \mathbf{tr}\left(\sum_{i=1}^{m} r^{ij} \left[-\left(x^i - \mu^j\right) d\left(\mu^j\right)^T - \left(x^i - \mu^j\right)^T d\left(\mu^j\right) \right] \right)$$

$$= \mathbf{tr}\left(\sum_{i=1}^{m} r^{ij} \left[-2\left(x^i - \mu^j\right)^T d\left(\mu^j\right) \right] \right)$$

Since $dJ = tr\left(\frac{\partial J}{\partial \mu^j}^T d\mu^j \right)$

So:

$$\frac{\partial J}{\partial \mu^j} = \sum_{i=1}^{m} r^{ij} \left[-2\left(x^i - \mu^j\right) \right]$$

Let $\frac{\partial J}{\partial \mu^j} = 0$, so $\mu^j = \frac{\sum_i r^{ij} x^i}{\sum_i r^{ij}}$

1.2.(i)

In general, we could let r^{ij} first to be fixed.

So $J = \sum_{i=1}^m r^{ij}(x^i - \mu^j)^T \Sigma (x^i - \mu^j)$ for this fixed r^{ij}.

$$dJ = \sum_{i=1}^m r^{ij} \left[d(x^i - \mu^j)^T \Sigma (x^i - \mu^j) + (x^i - \mu^j)^T \Sigma\, d(x^i - \mu^j) \right]$$

$$dJ = \sum_{i=1}^m r^{ij} \left[-d(\mu^j)^T \Sigma (x^i - \mu^j) - (x^i - \mu^j)^T \Sigma\, d(\mu^j) \right]$$

Since dJ is a scalar, so $dJ = tr(dJ)$

So, the upper equation can equate following:

$$dJ = tr(dJ)$$

$$tr\left(\sum_{i=1}^m r^{ij} \left[-d(\mu^j)^T \Sigma (x^i - \mu^j) - (x^i - \mu^j)^T \Sigma d(\mu^j) \right] \right)$$

$$= \mathbf{tr}\left(\sum_{i=1}^m r^{ij} \left[-(x^i - \mu^j) \Sigma\, d(\mu^j)^T - (x^i - \mu^j)^T \Sigma\, d(\mu^j) \right] \right)$$

$$= \mathbf{tr}\left(\sum_{i=1}^m r^{ij} \left[-2 (x^i - \mu^j)^T \Sigma^T d(\mu^j) \right] \right)$$

Since $dJ = tr\left(\frac{\partial J}{\partial \mu^j}^T d\mu^j \right)$

So:

$$\frac{\partial J}{\partial \mu^j} = \sum_{i=1}^m r^{ij} \left[-2 \Sigma (x^i - \mu^j) \right]$$

Let $\frac{\partial J}{\partial \mu^j} = 0$, since in order to let $\sum_{i=1}^m r^{ij}\left[-2\Sigma(x^i - \mu^j)\right] = 0$, and Σ is a potive matrix and has nothing to do with i, so we can reduce it. So equally,

$$\frac{\partial J}{\partial \mu^j} = 0$$

$$\Rightarrow \quad \sum_{i=1}^m r^{ij} \left[(x^i - \mu^j) \right] = 0$$

$$\Rightarrow \quad =\mu^j = \frac{\sum_i r^{ij} x^i}{\sum_i r^{ij}}$$

1.2.(ii)

Since $J = \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} \left(x^i - \mu^j\right)^T \Sigma \left(x^i - \mu^j\right)$, and in general r^{ij} is to define which cluster centroid μ^j will be the closest for point x^i.

So:

$$r^{ij} = \begin{cases} 1 & if\ j = \arg min_k \left(x^i - \mu^k\right)^T \Sigma \left(x^i - \mu^k\right) \\ 0 & otherwise \end{cases}$$

1.3

Proof:

Since $J = \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} \left\|x^i - \mu^j\right\|^2$, and define S to be one kind of division of the total *m* points. For example, S_1 to be a division that define the position of the *k* centroids and which cluster should each specific number belongs.

So, define the function $f(S_1) = \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} \left\|x^i - \mu^j\right\|^2$ to be the distortion value of this specific division of S_1.

As we know, in the K-means iteration, each adjustment will decrease the value J, which means $f(S)$ is a Monotone decreasing function. Besides, $f(S)$ is bounded, and its lower limit is 0.

So, according to the Monotone convergence theorem, so $\lim_{n \to \infty} f(S_n)$ exists, which means $f(S_n)$ converges. Besides, since we have the limit point number *m* and limit cluster number *k*, so we only have a limit possibility, which is k^m. So the maximum number $n = k^m$, which means $f(S_n)$ will converge in limit n steps.

1.4

(a): By simple calculation, in the first iteration, point 1,2,3 belong to cluster B, point 4,5 belong to cluster A.

(b): For centroid A: $(x_A,\ y_A) = \frac{1}{2}[(0,-1)+(-2,-2)] = (-1,-1.5)$

For centroid B: $(x_B,\ y_B) = \frac{1}{3}[(2,2)+(3,1)+(-1,1)] = \left(\frac{4}{3},\frac{4}{3}\right)$

(c): Yes, it will terminate in one step. Because in the next step we will do the point assignment, and there is no more change, which mean the objective function has reached the local minimum.

2. Image compression

1.

For the K-metroids, the iteration process is quite the same with K-means except for choosing each cluster's centroid. In K-metroids, we must select a true point c, which can also get the minimize value of $\sum_{i:\,\pi(i)=j}\left\|x^i - c\right\|^2$. In my code, I still compute the K-means centroid c. Then, choose the data x from the c's clustering, which has the smallest $d(x,c)$. Then, replace c with the x RGB value.

I choose the Euclidian and Manhattan distance respectively. The picture with different clustering numbers and different distance are in the homework file package. In gerenal, the runtime for K-metroids is longer than that of K-means, simply because we need to do one more iteration, which is to find the appropriate x to represent c. Besides, the runtime for Manhattan distance is slightly slower than that of Euclidian.

I use two conditions to stop iteration, one is the maximum iteration step is 200, and the other is $\|c\_new - c\|^2$ must be less than 10.

Besides, for the first assignment of initial centroid c, instead of using any random numbers which are not guaranteed to be a true point in the image, I randomly pick the true points in the image to initialize my centroid c.

2.

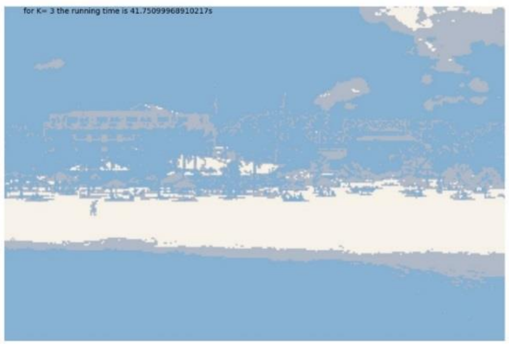
The K means how many colors I use to represent this new Image. So apparently, with the K increase, the image will become more vivid and similar with the original one. The runtime with different K is shown as below table using K-metroid method:

Picture name	K number	Distance	Runtime
Fengjing	2	Euclidian	72.31s
	3	Euclidian	73.19s
	10	Euclidian	81.49s
	20	Euclidian	93.00s
	50	Euclidian	125.07s
Beach	2	Euclidian	41.02s
	3	Euclidian	41.75s
		Manhattan	60.03s
	10	Euclidian	46.26s
	20	Euclidian	52.53s
	50	Euclidian	69.34s
		Manhattan	72.36s
Football	2	Euclidian	138.21s

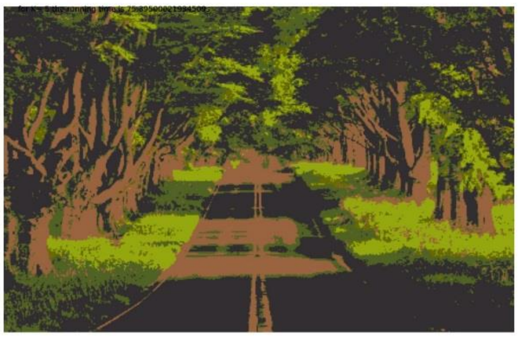
	3	Euclidian	142.06s
	10	Euclidian	158.10s
	20	Euclidian	182.59s
	50	Euclidian	242.85s

The following pictures are derived by using K-centroid method and Euclidian distance.

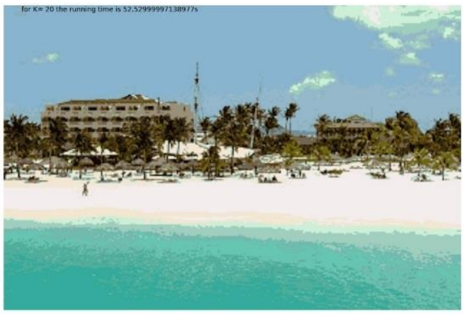
K=3



K=5

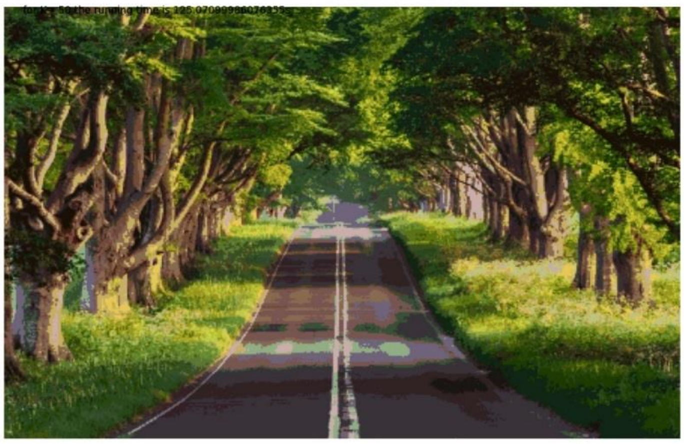


K=20



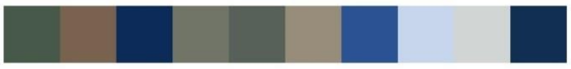
```
for K= 20 the running time is 52.52999997138977s
```

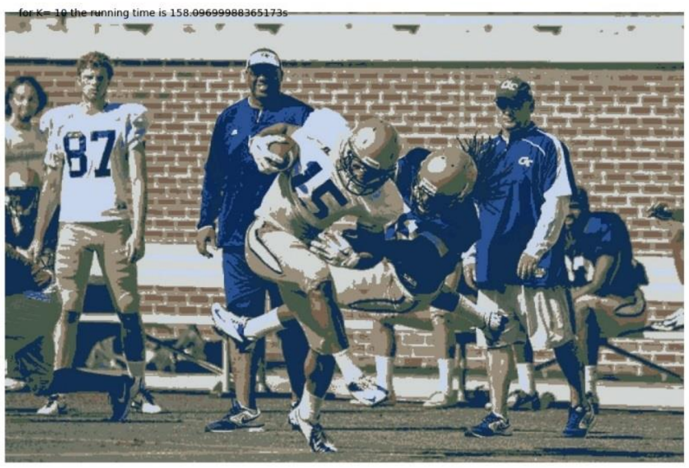
K=50



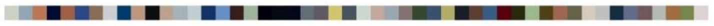
```
for K= 50 the running time is 125.07099986076355s
```

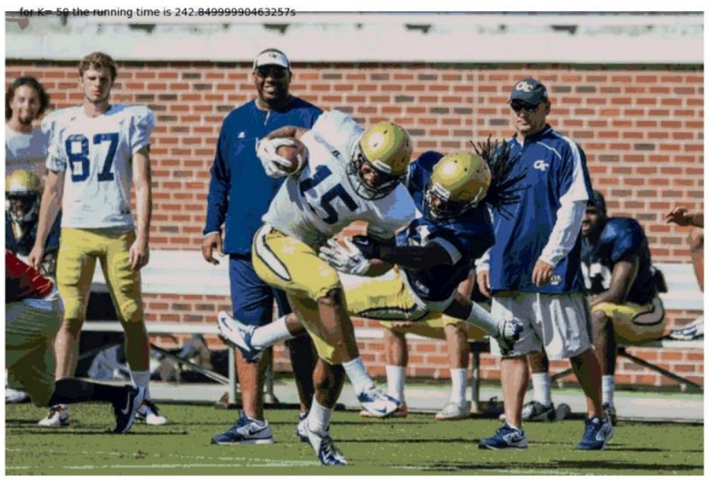
K=10




for K= 10 the running time is 158.09699988365173s

K=50

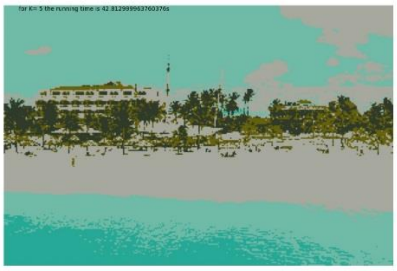



for K= 50 the running time is 242.84999990463257s

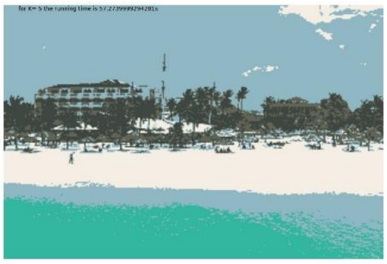
The following is the comparison of Euclidian distance and Manhattan distance in K-Metroid method.

K=5

Euclidian

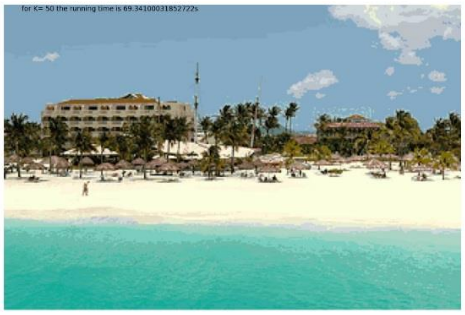

for K= 5 the running time is 42.812999963760376s

Manhattan

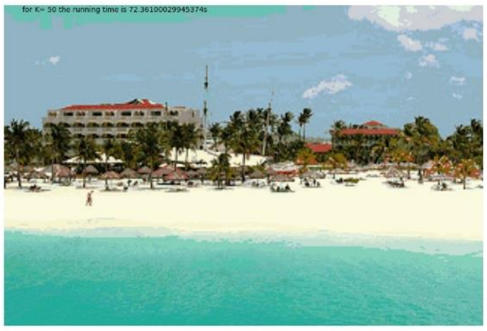

for K= 5 the running time is 57.2739999294281s

K=50

Euclidian


for K= 50 the running time is 69.34100031852722s

Manhattan


for K= 50 the running time is 72.36100029945374s

The qualities are roughly the same with each other, and the runtime of Euclidian is roughly the same with that of Manhattan.

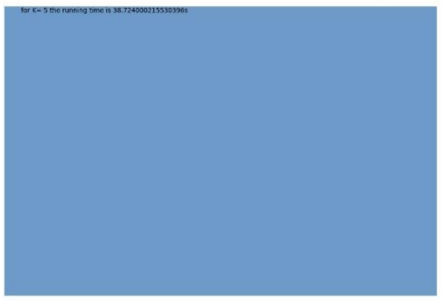
3.

Previously, I use a random picked initial centroid to start the iteration. In the poor assignment, I choose K centroids which are very similar with each other. The following pictures are derived by using K-centroid method and Euclidian distance, which are shown as below:

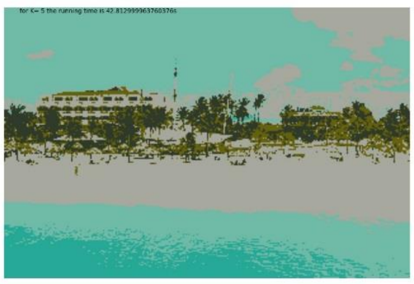
K=5

Poor pick

Random pick



```
for K= 5 the running time is 38.724000215530396s
```

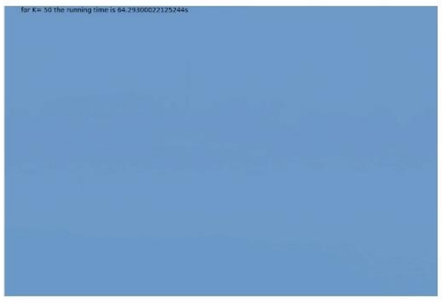


```
for K= 5 the running time is 42.812999963760376s
```

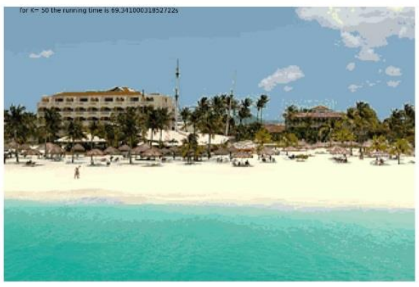
K=50

Poor pick

Random pick



```
for K= 50 the running time is 64.29300022125244s
```



```
for K= 50 the running time is 69.34100031852722s
```

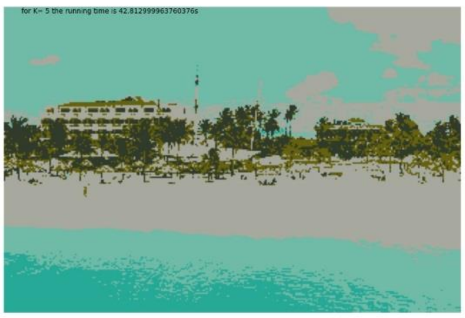
So, as what we can see, if we poorly choose the initial centroids c, the quality is poor even though with a high value K. This is because the initial value of c is very close to each other, and it won't change once it reach a local minimum value. Besides, the runtime of poor pick is slightly faster than that of random pick.

4.

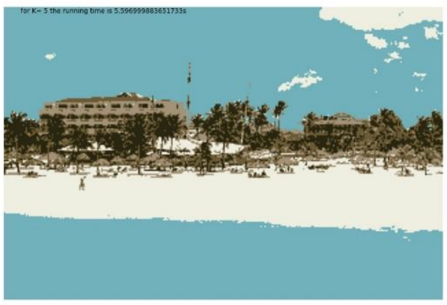
In general comparison between K-means and K-metroid, the Robust of K-metroid is much higher than that of K-means. However, since in the K-metroid method, I first compute the K-means centroid, then do the point assignment, last compute the new centroid which is the closest point in each clustering, so the runtime for K-metroid is much higher than that of K-means, which is about 5~10 times of K-means runtime. As for the quality, when K is small, the quality of K-means is better than that of K-metroid. With the K becomes larger, the significance will decrease, but the quality of K-means is still higher than that of K-metroid.

K=5, L2 distance

K-metroid



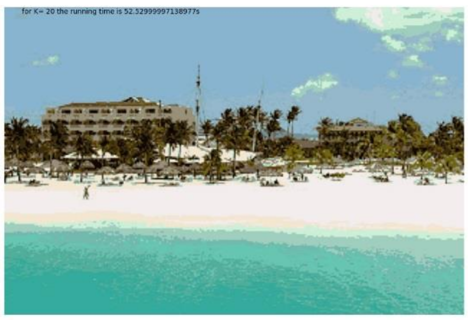
K-means



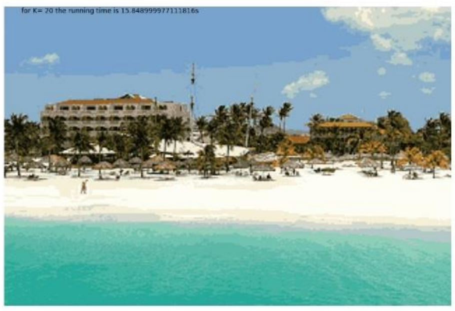
K=20, L2 distance

K-metroid

K-means



```
for K= 20 the running time is 52.52999997138977s
```

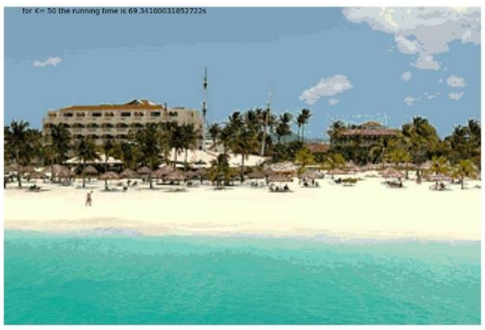


```
for K= 20 the running time is 15.848999977111816s
```

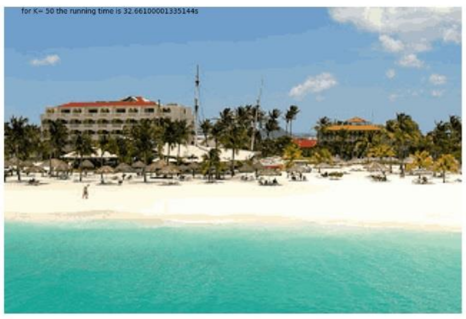
K=50, L2 distance

K-metroid

K-means



```
for K= 50 the running time is 69.34100031852722s
```



```
for K= 50 the running time is 32.66100001335144s
```

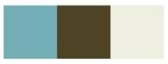
The K means how many colors I use to represent this new Image. So apparently, with the K increase, the image will become more vivid and similar with the original one. The runtime with different K is shown as below table:

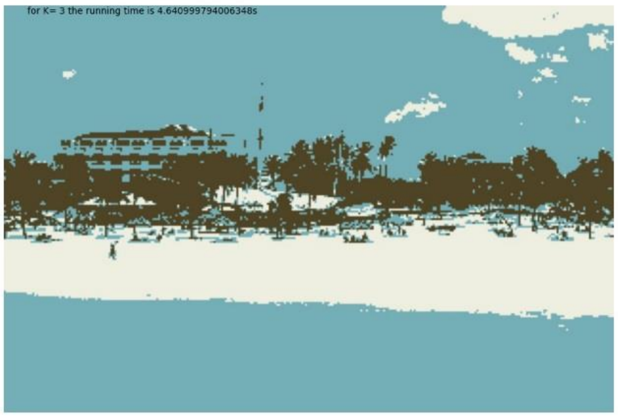
Picture name	K number	Distance	Runtime
Fengjing	2	Euclidian	8.95s
	3	Euclidian	9.98s
	10	Euclidian	17.56s
	20	Euclidian	27.94s
	50	Euclidian	63.42s
Beach	2	Euclidian	4.54s
	3	Euclidian	4.64s
		Manhattan	2.5s
	10	Euclidian	8.39s
	20	Euclidian	15.85s
	50	Euclidian	32.66s
		Manhattan	31.51s
Football	2	Euclidian	15.17s
	3	Euclidian	17.02s
	10	Euclidian	31.34s
	20	Euclidian	59.70s

	50	Euclidian	117.94s

The following pictures are derived by using K-means method and Euclidian distance.

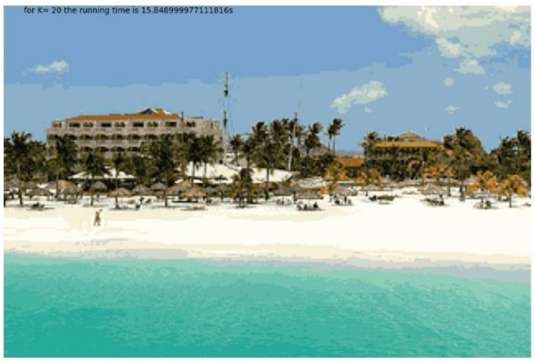
K=3



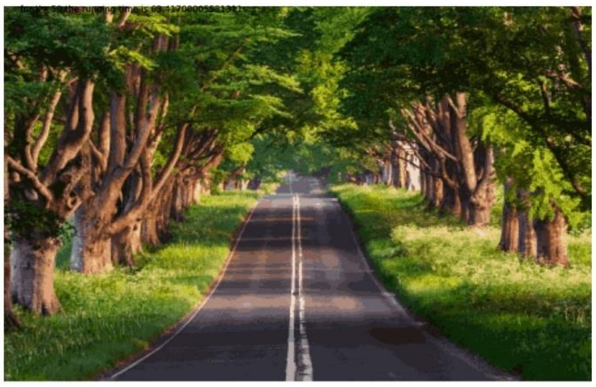


```
for K= 3 the running time is 4.640999794006348s
```

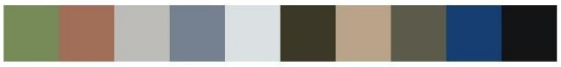
K=20

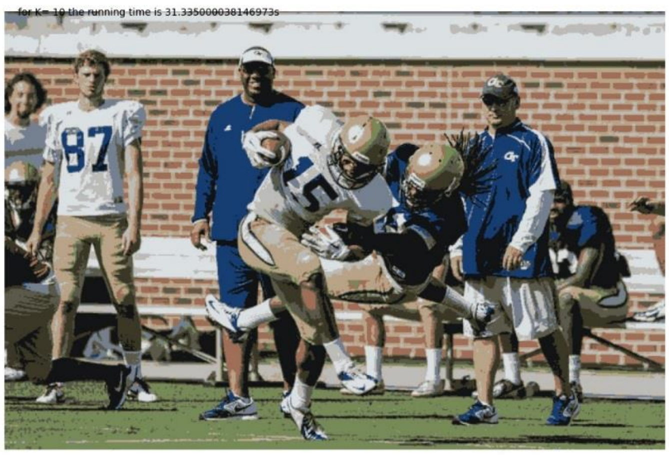

for K= 20 the running time is 15.848999977111816s

K=50

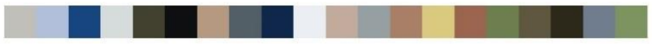

for K= 50 the running time is 69.41700005531311s

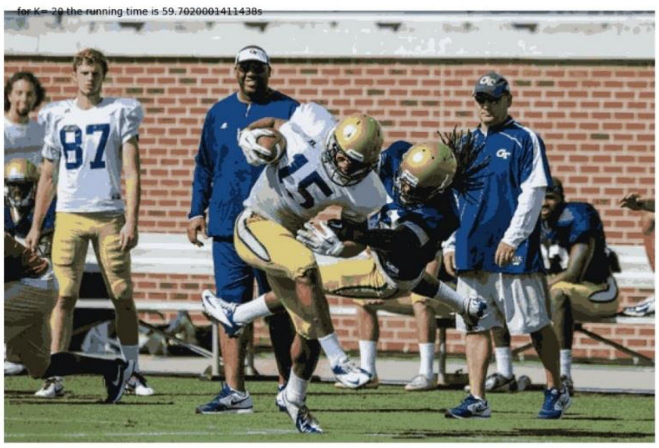
K=10



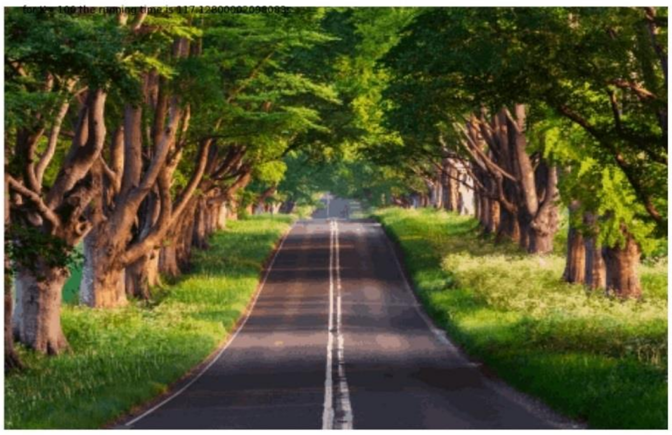

for K= 10 the running time is 31.335000038146973s

K=20




for K= 20 the running time is 59.7020001411438s

K=100



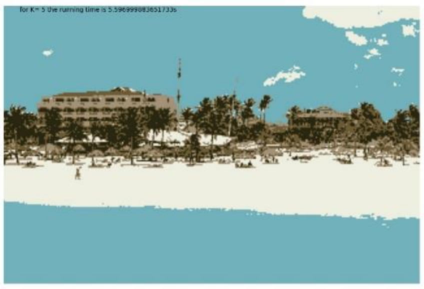
```
for K= 100 the running time is 117.12800002098083s
```

The following is the comparison of Euclidian distance and Manhattan distance in K-Means method.

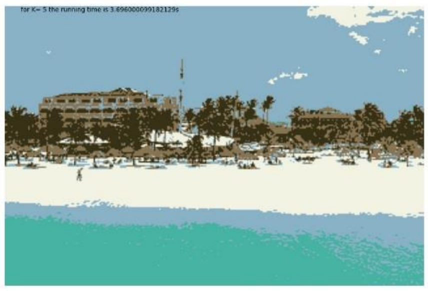
K=5

Euclidian

Manhattan



```
for K= 5 the running time is 5.596999883651733s
```



```
for K= 5 the running time is 3.696000099182129s
```

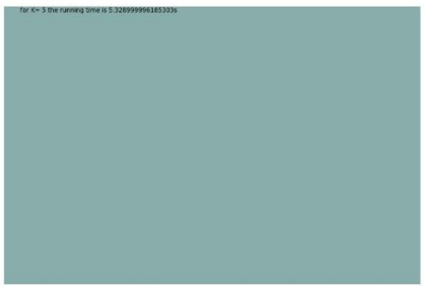
The qualities are roughly the same with each other, and the runtime of Euclidian is roughly the same with that of Manhattan.

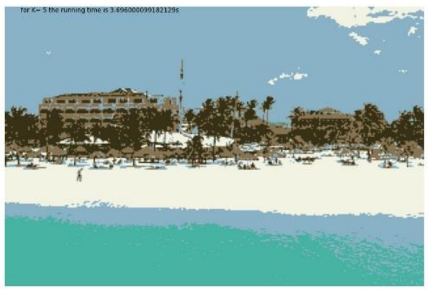
Previously, I use a random picked initial centroid to start the iteration. In the poor assignment, I choose K centroids which are very similar with each other. The following pictures are derived by using K-means method and Euclidian distance, which are shown as below:

K=5

Poor pick

Random pick

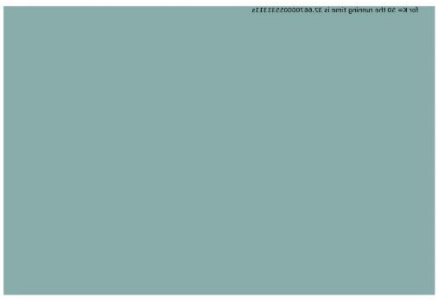


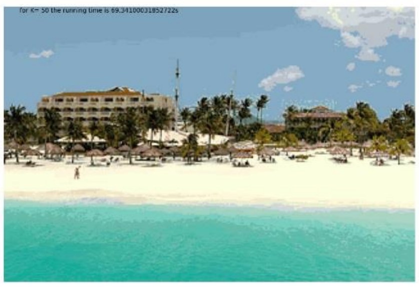


K=50

Poor pick

Random pick





So, as what we can see, if we poorly choose the initial centroids c, the quality is poor even though with a high value K. This is because the initial value of c is very close to each other, and it won't change once it reach a local minimum value.

1.3 Political blogs

1.

From the Graph Laplacian aspect, k means the number of blocks. From the eigenvalue aspect, k means we choose the k smallest eigenvalues to form the $\mathbb{R}^{\boldsymbol{m}\times \mathrm{k}}$ eigenvector matrix, where each row is the feature vector, which is used to divide the node into different communities. From the Nodes aspect, k means the Nodes are separated into k communities, where the Nodes are strongly connected with each other, and intuitively the nodes in the same community share the same political orientation.

2,3.

The original total node number is 1490, after removing the isolated data, the left total node number is 1224.

Since $\mathbf{L} = \mathbf{D} - \mathbf{A} = \mathbf{D}^{\frac{1}{2}}\left[\mathbf{I} - \mathbf{D}^{\frac{1}{2}}\,\mathbf{A}\,\mathbf{D}^{\frac{1}{2}}\right]\boldsymbol{D}^{\frac{1}{2}}$

Let $\mathbf{B} = \mathbf{D}^{\frac{1}{2}}\,\mathbf{A}\,\mathbf{D}^{\frac{1}{2}}$, finding smallest eigenvalues for $\mathbf{L}$ equals finding largest eigenvalues for $\mathbf{B}$, and the top ten largest eigenvalues are listed as below:

No	eigenvalue
1	1
2	1
3	0.934
4	0.899
5	0.863
6	0.714

7	0.711
8	0.636
9	0.611
10	0.589

We main discover that the largest eigenvalue gap occurs between row K=5 and K=6.

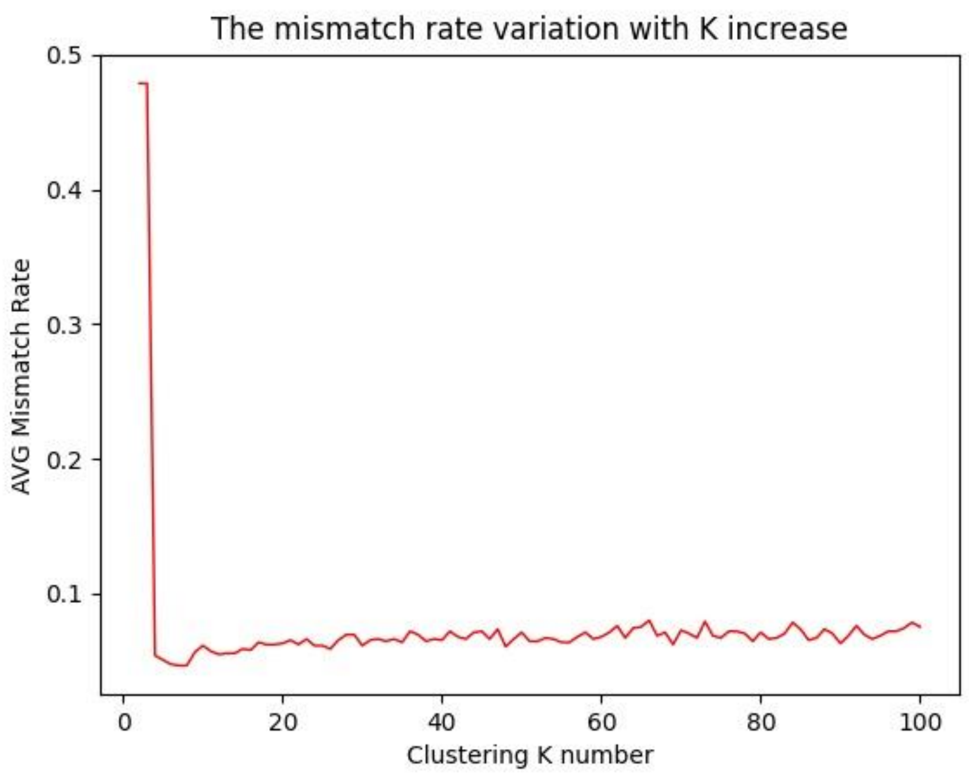
The AVG Mismatch Rate means the total mismatch nodes number divide the total node number. The following table shows the detailed information when the Total Cluster number **K =2,3,4** respectively.

Total Cluster number **K**	Cluster Number	Node number in Cluster	Majority Label	Mismatch Rate	AVG Mismatch Rate
2	0	1222	1	48.0%	47.9%
	1	2	0	0.0%	
3	0	1216	1	48.0%	47.9%
	1	6	1	33.3%	
	2	2	0	0.0%	
4	0	78	1	44.9%	5.4%
	1	546	0	2.2%	
	2	593	1	2.7%	

3	7	1	42.9%

4.

Firstly, I don't know how to tune the clustering number K value, so I do the iteration with K between 2 and 100 to find the smallest AVG Mismatch Rate. The following graph shows the iteration process:



From the graph, we find that the smallest AVG Mismatch Rate occurs when K = 5, with the smallest value which is 4.6%. And no surprisingly, the largest eigenvalue gap occurs between K=5 and K=6. So, in conclusion, in order to find the smallest AVG mismatch rate, the better to define the cluster's number K value is to set it equals to the K when the largest eigenvalue gap occurs. In this example, the largest eigenvalue gap occurs between K=5 and K=6, so K=5 has the smallest AVG mismatch rate, which means we have a better clustering result.

5.

1. imperfect clustering data, which have some noise, how to define the clustering number K is a big problem. And it is not always good when using the larger K number. Using the biggest eigenvalue gap is a better way to define K number(this case, K=5), which is proved by the graph that the AVG mismatch rate is lower than that when K is larger than 5.

2. It not always true node within same community tend to share the same political view, even though we use the best K number. In the best K(K=5), For some clusters with large Node number, it is true, which has a much lower Mismatch Rate. But for the smaller clusters, the result is not good. Node in smaller clusters tend to have different political view.

Total Cluster number **K**	Cluster Number	Node number in Cluster	Majority Label	Mismatch Rate	AVG Mismatch Rate
	0	581	1	2.4%	
	1	500	0	2.2%	
5	2	65	0	9.2%	4.6%
	3	68	1	33.8%	
	4	10	1	30.0%	